A

ABS Algorithms for Linear Equations and Linear Least Squares

EMILIO SPEDICATO
Department Math., University Bergamo,
Bergamo, Italy

MSC2000: 65K05, 65K10

Article Outline

Keywords

Linear algebraic equations; Linear least squares; ABS methods; Abaffian matrices; Huang algorithm; Implicit LU algorithm; Implicit LX algorithm

Synonyms

Abaffi–Broyden–Spedicato algorithms for linear equations and linear least squares

The Scaled ABS Class: General Properties

ABS methods were introduced by [1], in a paper dealing originally only with solving linear equations via what is now called the *basic* or *unscaled ABS class*. The basic ABS class was later generalized to the so-called *scaled ABS class* and subsequently applied to linear least squares, nonlinear equations and optimization problems, see [2]. Preliminary work has also been initiated concerning *Diophantine equations*, with possible extensions to combinatorial optimization, and the eigenvalue problem. There are presently (1998) over 350 papers in the ABS field, see [11]. In this contribution we will review the basic properties and results of ABS methods for solving linear determined or underdetermined systems and overdetermined linear systems in the least squares sense.

Let us consider the linear determined or underdetermined system, where rank(A) is arbitrary

$$Ax = b, \quad x \in \mathbb{R}^n, b \in \mathbb{R}^m, \quad m \leq n, \tag{1}$$

or

$$a_i^\top x - b_i = 0, \quad i = 1, \ldots, m, \tag{2}$$

where

$$A = \begin{pmatrix} a_1^\top \\ \vdots \\ a_m^\top \end{pmatrix}. \tag{3}$$

The steps of the scaled ABS class algorithms are as follows:

A) Let $x_1 \in \mathbf{R}^n$ be arbitrary, $H_1 \in \mathbf{R}^{n,n}$ be nonsingular arbitrary, v_1 be an arbitrary nonzero vector in $\mathbf{R}^m$; set $i = 1$.

B) Compute the residual $r_i = Ax_i - b$. If $r_i = 0$, stop (x_i solves the problem); else compute $s_i = H_i A^\top v_i$. If $s_i \neq 0$, then go to C). If $s_i = 0$ and $\tau = v_i^\top r_i = 0$, then set $x_{i+1} = x_i$, $H_{i+1} = H_i$ and go to F), else stop (the system has no solution).

C) Compute the search vector p_i by

$$p_i = H_i^\top z_i, \tag{4}$$

where $z_i \in \mathbf{R}^n$ is arbitrary save for the condition

$$v_i^\top A H_i^\top z_i \neq 0. \tag{5}$$

D) Update the estimate of the solution by

$$x_{i+1} = x_i - \alpha_i p_i, \tag{6}$$

where the stepsize α_i is given by

$$\alpha_i = \frac{v_i^\top r_i}{r_i^\top A p_i}. \tag{7}$$

E) Update the matrix H_i by

$$H_{i+1} = H_i - \frac{H_i A^\top v_i w_i^\top H_i}{w_i^\top H_i A^\top v_i}, \tag{8}$$

where $w_i \in \mathbf{R}^n$ is arbitrary save for the condition

$$w_i^\top H_i A^\top v_i \neq 0. \tag{9}$$

F) If $i = m$, then stop (x_{m+1} solves the system), else define v_{i+1} as an arbitrary vector in $\mathbf{R}^m$ but linearly independent from $v_1, \ldots, v_i$, increment i by one and go to B).

The matrices H_i appearing in step E) are generalizations of (*oblique*) *projection matrices*. They probably first appeared in [16]. They have been named *Abaffians* since the first international conference on ABS methods (Luoyang, China, 1991) and this name will be used here.

The above recursion defines a class of algorithms, each particular method being determined by the choice of the parameters H_1, v_i, z_i, w_i. The *basic ABS class* is obtained by taking $v_i = e_i$, e_i being the ith unitary vector in $\mathbf{R}^m$. The parameters w_i, z_i, H_1 have been introduced respectively by J. Abaffy, C.G. Broyden and E. Spedicato, whose initials are referred to in the name of the class. It is possible to show that the scaled ABS class is a complete realization of the so-called *Petrov–Galerkin iteration* for solving a linear system (but the principle can be applied to more general problems), where the iteration has the form $x_{i+1} = x_i - \alpha_i p_i$ with α_i, p_i chosen so that the orthogonality relation $r_{i+1}^\top v_j = 0$, $j = 1, \ldots, i$, holds, the vectors v_j being arbitrary linearly independent. It appears that all deterministic algorithms in the literature having finite termination on a linear system are members of the scaled ABS class (this statement has been recently shown to be true also for the *quasi-Newton methods*, which are known to have under some conditions termination in at most $2n$ steps: the iterate of index $2i - 1$ generated by Broyden's iteration corresponds to the ith iterate of a certain algorithm in the ABS class).

Referring [2] for proofs, we give some of the general properties of methods of the scaled ABS class, assuming, for simplicity, that A has full rank.

- Define $V_i = (v_1, \ldots, v_i)$, $W_i = (w_1, \ldots, w_i)$. Then $H_{i+1} A^\top V_i = 0$, $H_{i+1}^\top W_i = 0$, meaning that vectors $A^\top v_j$, w_j, $j = 1, \ldots, i$, span the null spaces of H_{i+1} and its transpose, respectively.
- The vectors $H_i A^\top v_i$, $H_i^\top w_i$ are nonzero if and only if a_i, w_i are linearly independent from $a_1, \ldots, a_{i-1}$, $w_1, \ldots, w_{i-1}$, respectively.
- Define $P_i = (p_1, \ldots, p_i)$. Then the implicit factorization $V_i^\top A_i^\top P_i = L_i$ holds, where L_i is nonsingular lower triangular. From this relation, if $m = n$, one obtains the following semi-explicit factorization of the inverse, with $P = P_n$, $V = V_n$, $L = L_n$

$$A^{-1} = P L^{-1} V^\top. \tag{10}$$

For several choices of the matrix V the matrix L is diagonal, hence formula (10) gives a fully explicit factorization of the inverse as a byproduct of the ABS solution of a linear system, a property that does not hold for the classical solvers. It can also be shown that all possible factorizations of the form (10) can be obtained by proper parameter choices in the scaled ABS class, another completeness result.

- Define S_i and R_i by $S_i = (s_1, \ldots, s_i)$, $R_i = (r_1, \ldots, r_i)$, where $s_i = H_i A^\top v_i$, $r_i = H_i^\top w_i$. Then the Abaffian can be written in the form $H_{i+1} = H_1 - S_i R_i^\top$ and the vectors s_i, r_i can be built via a *Gram–Schmidt type iterations* involving the previous vectors (the search vector p_i can be built in a similar way). This representation of the Abaffian in terms of $2i$ vectors is computationally convenient when the number of equations is much less than the number of variables. Notice that there is also a representation in terms of $n - i$ vectors.

- A compact formula of the Abaffian in terms of the parameter matrices is the following

$$H_{i+1} = H_1 - H_1 A^\top V_i (W_i^\top H_1 A^\top V_i)^{-1} W_i^\top H_1. \tag{11}$$

Letting $V = V_m$, $W = W_m$, one can show that the parameter matrices H_1, V, W are admissible (i. e. are such that condition (9) is satisfied) if and only if the matrix $Q = V^\top A H_1^\top W$ is *strongly nonsingular* (i. e. is LU factorizable). Notice that this condition can always be satisfied by suitable exchanges of the columns of V or W, equivalent to a row or a column pivoting on the matrix Q. If Q is strongly nonsingular and we take, as is done in all algorithms insofar considered, $z_i = w_i$, then condition (5) is also satisfied.

It can be shown that the *scaled ABS class* corresponds to applying (implicitly) the unscaled ABS algorithm to the scaled (or preconditioned) system $V^\top Ax = V^\top b$, where V is an arbitrary nonsingular matrix of order m. Therefore we see that the scaled ABS class is also complete with respect to all possible left preconditioning matrices, which in the ABS context are defined implicitly and dynamically (only the ith column of V is needed at the ith iteration, and it can also be a function of the previous column choices).

Subclasses of the ABS Class

In [1], nine subclasses are considered of the scaled ABS class. Here we quote three important subclasses.

- The *conjugate direction subclass*. This class is well defined under the condition (sufficient but not necessary) that A is symmetric and positive definite. It contains the *implicit Choleski algorithm*, the *Hestenes–Stiefel* and the *Lanczos algorithms*. This class generates all possible algorithms whose search directions are A-conjugate. The vector x_{i+1} minimizes the *energy* or *A-weighted Euclidean norm* of the error over $x_1 + \mathrm{Span}(p_1, \ldots, p_i)$. If $x_1 = 0$, then the solution is approached monotonically from below in the energy norm.
- The *orthogonally scaled subclass*. This class is well defined if A has full column rank and remains well defined even if m is greater than n. It contains the ABS formulation of the QR algorithm (the so-called *implicit QR algorithm*), of the *GMRES* and of the *conjugate residual algorithms*. The scaling vectors are orthogonal and the search vectors are $AA^\top$-conjugate. The vector x_{i+1} minimizes the Euclidean norm of the residual over $x_1 + \mathrm{Span}(p_1, \ldots, p_i)$. In general, the methods in this class can be applied to overdetermined systems to obtain the solution in the least squares sense.
- The *optimally scaled subclass*. This class is obtained by the choice $v_i = A^{-\top} p_i$. The inverse of $A^\top$ disappears in the actual formulas, if we make the change of variables $z_i = A^\top u_i$, u_i being now the parameter that defines the search vector. For $u_i = e_i$ the *Huang method* is obtained and for $u_i = r_i$ a method equivalent to *Craig's conjugate gradient type algorithm*. From the general implicit factorization relation one obtains $P^\top P = D$ or $V^\top AA^\top V = D$, a relation which was shown in [5] to characterize the optimal choice of the parameters in the general Petrov–Galerkin process in terms of minimizing the effect of a single error in x_i on the final computed solution. Such a property is therefore satisfied by the Huang (and the Craig) algorithm, but not, for instance, by the implicit LU or the implicit QR algorithms. A. Galantai [8] has shown that the condition characterizing the optimal choice of the scaling parameters in terms of minimizing the final residual Euclidean norm is $V^\top V = D$, a condition satisfied by the implicit QR algorithm, the GMRES method, the implicit LU algorithm and again by the Huang algorithm, which therefore satisfies both conditions). The methods in the optimally stable subclass have the property that x_{i+1} minimizes the Euclidean norm of the error over $x_1 + \mathrm{Span}(p_1, \ldots, p_i)$.

The Implicit LU Algorithm and the Huang Algorithm

Specific algorithms of the scaled ABS class are obtained by choosing the available parameters. The *implicit LU algorithm* is given by the choices $H_1 = I$, $z_i = w_i = v_i = e_i$. We quote the following properties of the implicit LU algorithm.

a) The algorithm is well defined if and only if A is *regular* (i. e. all principal submatrices are nonsingular). Otherwise column pivoting has to be performed (or, if $m = n$, equations pivoting).

b) The Abaffian H_{i+1} has the following structure, with $K_i \in \mathbf{R}^{n-i,\,i}$:

$$H_{i+1} = \begin{pmatrix} 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ K_i & I_{n-i} \end{pmatrix}. \tag{12}$$

c) Only the first i components of p_i can be nonzero and the ith component is one. Hence the matrix P_i is unit upper triangular, so that the implicit factorization $A = LP^{-1}$ is of the LU type, with units on the diagonal, justifying the name.

d) Only K_i has to be updated. The algorithm requires $nm^2 - 2m^3/3$ multiplications plus lower order terms, hence, for $m = n$, $n^3/3$ multiplications plus lower order terms. This is the same overhead required by the *classical LU factorization* or *Gaussian elimination* (which are two essentially equivalent processes).

e) The main storage requirement is the storage of K_i, whose maximum value is $n^2/4$. This is two times less than the storage needed by Gaussian elimination and four times less than the storage needed by the LU factorization algorithm (assuming that A is not overwritten). Hence the implicit LU algorithm is computationally better than the classical Gaussian elimination or LU algorithm, having the same overhead but less memory cost.

The implicit LU algorithm, implemented in the case $m = n$ with row pivoting, has been shown in experiments of M. Bertocchi and Spedicato [3] to be numerically stable and in experiments of E. Bodon [4] on the vector processor Alliant FX 80 with 8 processors to be about twice faster than the *LAPACK* implementation of the classical LU algorithm.

The *Huang algorithm* is obtained by the parameter choices $H_1 = I$, $z_i = w_i = a_i$, $v_i = e_i$. A mathematically equivalent, but numerically more stable, formulation of this algorithm is the so-called *modified Huang algorithm* where the search vectors and the Abaffians are given by formulas $p_i = H_i(H_i a_i)$ and $H_{i+1} = H_i - p_i p_i^\top / p_i^\top p_i$. Some properties of this algorithm follow.

- The search vectors are orthogonal and are the same vectors obtained by applying the classical Gram–Schmidt orthogonalization procedure to the rows of A. The modified Huang algorithm is related, but is not numerically identical, with the *Daniel–Gragg–Kaufmann–Stewart reorthogonalized Gram–Schmidt algorithm* [6].
- If x_1 is the zero vector, then the vector x_{i+1} is the solution with least Euclidean norm of the first i equations and the solution x^+ of least Euclidean norm of the whole system is approached monotonically and from below by the sequence x_i. L. Zhang [17] has shown that the Huang algorithm can be applied, via the *Goldfarb–Idnani active set strategy* [9], to systems of linear inequalities. The process in a finite number of steps either finds the solution with least Euclidean norm or determines that the system has no solution.
- While the error growth in the Huang algorithm is governed by the square of the number $\eta_i = \| a_i \| / \| H_i a_i \|$, which is certainly large for some i if A is ill conditioned, the error growth depends only on η_i if p_i or H_i are defined as in the modified Huang algorithm and, at first order, there is no error growth for the modified Huang algorithm.
- Numerical experiments, see [15], have shown that the modified Huang algorithm is very stable, giving usually better accuracy in the computed solution than both the implicit LU algorithm and the classical LU factorization method.

The *implicit LX algorithm* is defined by the choices $H_1 = I$, $v_i = e_i$, $z_i = w_i = e_{k_i}$, where k_i is an integer, $1 \le k_i \le n$, such that

$$e_{k_i}^\top H_i a_i \neq 0. \tag{13}$$

Notice that by a general property of the ABS class for A with full rank there is at least one index k_i such that (13) is satisfied. For stability reasons it may be recommended to select k_i such that $\eta_i = |e_{k_i}^\top H_i a_i|$ is maximized.

The following properties are valid for the implicit LX algorithm. Let N be the set of integers from 1 to n, $N = (1, \ldots, n)$. Let B_i be the set of indexes $k_1, \ldots, k_i$ chosen for the parameters of the implicit LX algorithm up to the step i. Let N_i be the set $N \setminus B_i$. Then:

- The index k_i is selected in the set N_{i-1}.
- The rows of H_{i+1} of index $k \in B_i$ are null rows.
- The vector p_i has $n - i$ zero components; its k_ith component is equal to one.
- If $x_1 = 0$, then x_{i+1} is a basic type solution of the first i equations, whose nonzero components may lie

only in the positions corresponding to the indices $k \in B_i$.

- The columns of H_{i+1} of index $k \in N_i$ are the unit vectors e_k, while the columns of H_{i+1} of index $k \in B_i$ have zero components in the jth position, with $j \in B_i$, implying that only $i(n-i)$ elements of such columns have to be computed.
- At the ith step $i(n-i)$ multiplications are needed to compute $H_i a_i$ and $i(n-i)$ to update the nontrivial part of H_i. Hence the total number of multiplications is the same as for the implicit LU algorithm (i. e. $n^3/3$), but no pivoting is necessary, reflecting the fact that no condition is required on the matrix A.
- The storage requirement is the same as for the implicit LU algorithm, i. e. at most $n^2/4$. Hence the implicit LX algorithm shares the same storage advantage of the implicit LU algorithm over the classical LU algorithm, with the additional advantage of not requiring pivoting.
- Numerical experiments by K. Mirnia [10] have shown that the implicit LX method gives usually better accuracy, in terms of error in the computed solution, than the implicit LU algorithm and often even than the modified Huang algorithm. In terms of size of the final residual, its accuracy is comparable to that of the LU algorithm as implemented (with row pivoting) in the MATLAB or LAPACK libraries, but it is better again in terms of error in the solution.

Other ABS Linear Solvers

ABS reformulations have been obtained for most algorithms proposed in the literature. The availability of several formulations of the linear algebra of the ABS process allows alternative formulations of each method, with possibly different values of overhead, storage and different properties of numerical stability, vectorization and parallelization. The reprojection technique, already seen in the case of the modified Huang algorithm and based upon the identities $H_i q = H_i(H_i q)$, $H_i^\top = H_i^\top (H_i^\top q)$, valid for any vector q if $H_1 = I$, remarkably improves the stability of the algorithm. The ABS versions of the *Hestenes–Stiefel* and the *Craig algorithms* for instance are very stable under the above reprojection. The *implicit QR algorithm*, defined by the choices $H_1 = I$, $v_i = Ap_i$, $z_i = w_i = e_i$ can be implemented in a very stable way using the reprojection in both the definition of the search vector and the scaling vector. It should also be noticed that the classical iterative refinement procedure, which amounts to a Newton iteration on the system $Ax - b = 0$ using the approximate factors of A, can be reformulated in the ABS context using the previously defined search vectors p_i. Experiments of Mirnia [11] have shown that ABS refinement works excellently.

For problems with special structure ABS methods can often be implemented taking into account the effect of the structure on the Abaffian matrix, which often tends to reflect the structure of the matrix A. For instance, if A has a banded structure, the same is true for the Abaffian matrix generated by the implicit LU, the implicit QR and the Huang algorithm, albeit the band size is increased. If A is SPD and has a *ND* structure, the same is true for the Abaffian matrix. In this case the implementation of the implicit LU algorithm has much less storage cost, for large n, than the cost required by an implementation of the Choleski algorithm. For matrices having the Kuhn–Tucker structure (KT structure) large classes of ABS methods have been devised, see ► ABS algorithms for optimization. For matrices with general sparsity patterns little is presently known about minimizing the fill-in in the Abaffian matrix. Careful use of BLAS4 routines can however substantially reduce the number of operations and make the ABS implementation competitive with a sparse implementation of say the LU factorization (e. g. by the code MA28) for values of n not too big.

It is possible to implement the ABS process also in block form, where several equations, instead of just one, are dealt with at each step. The block formulation does not deteriorate the numerical accuracy and can lead to reduction of overhead on special problems or to faster implementations on vector or parallel computers.

Finally infinite iterative methods can be obtained by the finite ABS methods via two approaches. The first one consists in restarting the iteration after $k < m$ steps, so that the storage will be of order $2kn$ if the representation of the Abaffian in terms of $2i$ vectors is used. The second approach consists in using only a limited number of terms in the Gram–Schmidt type processes that are alternative formulations of the ABS procedure. For both cases convergence at a linear rate has been established using the technique developed in [7]. The infinite

iteration methods obtained by these approaches define a very large class of methods, that contains not only all *Krylov space type methods* of the literature, but also non-Krylov type methods as the *Gauss–Seidel*, the *De La Garza* and the *Kackmartz methods*, with their generalizations.

ABS Methods for Linear Least Squares

There are several ways of using ABS methods for solving in the least squares sense an overdetermined linear system without forming the normal equations of Gauss, which are usually avoided on the account of their higher conditioning. One possibility is to compute explicitly the factors associated with the implicit factorization and then use them in the standard way. From results of [14] the obtained methods work well, giving usually better results than the methods using the QR factorization computed in the standard way. A second possibility is to use the representation of the *Moore–Penrose pseudo-inverse* that is provided explicitly by the ABS technique described in [13]. Again this approach has given very good numerical results. A third possibility is based upon the equivalence of the normal system $A^\top Ax = A^\top b$ with the extended system in the variables $x \in \mathbf{R}^n$, $y \in \mathbf{R}^m$, given by the two subsystems $Ax = y$, $A^\top y = A^\top b$. The first of the subsystems is overdetermined but must be solvable. Hence y must lie in the range of $A^\top$, which means that y must be the solution of least Euclidean norm of the second underdetermined subsystem. Such a solution is computed by the Huang algorithm. Then the ABS algorithm, applied to the first subsystem, in step B) recognizes and eliminates the $m - k$ dependent equations, where k is the rank of A. If $k < n$ there are infinite solutions and the one of least Euclidean norm is obtained by using again the Huang algorithm on the first subsystem.

Finally a large class of ABS methods can be applied directly to an overdetermined system stopping after n iterations in a least squares solution. The class is obtained by defining $V = AU$, where U is an arbitrary nonsingular matrix in $\mathbf{R}^n$. Indeed at the point x_{n+1} the satisfied Petrov–Galerkin condition is just equivalent to the normal equations of Gauss. If $U = P$ then the orthogonally scaled class is obtained, implying, as already stated in section 2, that the methods of this class can be applied to solve linear least squares (but a suitable modification has to be made for the *GMRES* method). A version of the implicit QR algorithm, with reprojection on both the search vector and the scaling vector, tested in [12], has outperformed other ABS algorithms for linear least squares methods as well as methods in the *LINPACK* and *NAG library* based upon the classical QR factorization via the Householder matrices.

See also

- ▶ ABS Algorithms for Optimization
- ▶ Cholesky Factorization
- ▶ Gauss–Newton Method: Least Squares, Relation to Newton's Method
- ▶ Generalized Total Least Squares
- ▶ Interval Linear Systems
- ▶ Large Scale Trust Region Problems
- ▶ Large Scale Unconstrained Optimization
- ▶ Least Squares Orthogonal Polynomials
- ▶ Least Squares Problems
- ▶ Linear Programming
- ▶ Nonlinear Least Squares: Newton-type Methods
- ▶ Nonlinear Least Squares Problems
- ▶ Nonlinear Least Squares: Trust Region Methods
- ▶ Orthogonal Triangularization
- ▶ Overdetermined Systems of Linear Equations
- ▶ QR Factorization
- ▶ Solving Large Scale and Sparse Semidefinite Programs
- ▶ Symmetric Systems of Linear Equations

References

1. Abaffy J, Broyden CG, Spedicato E (1984) A class of direct methods for linear systems. Numerische Math, 45:361–376
2. Abaffy J, Spedicato E (1989) ABS projection algorithms: Mathematical techniques for linear and nonlinear equations. Horwood, Westergate
3. Bertocchi M, Spedicato E (1989) Performance of the implicit Gauss–Choleski algorithm of the ABS class on the IBM 3090 VF. In: Proc. 10th Symp. Algorithms, Strbske Pleso, pp 30–40
4. Bodon E (1993) Numerical experiments on the ABS algorithms for linear systems of equations. Report DMSIA Univ Bergamo 93(17)
5. Broyden CG (1985) On the numerical stability of Huang's and related methods. JOTA 47:401–412
6. Daniel J, Gragg WB, Kaufman L, Stewart GW (1976) Reorthogonalized and stable algorithms for updating

the Gram–Schmidt QR factorization. Math Comput 30: 772–795
7. Dennis J, Turner K (1987) Generalized conjugate directions. Linear Alg & Its Appl 88/89:187–209
8. Galantai A (1991) Analysis of error propagation in the ABS class. Ann Inst Statist Math 43:597–603
9. Goldfarb D, Idnani A (1983) A numerically stable dual method for solving strictly convex quadratic programming. Math Program 27:1–33
10. Mirnia K (1996) Numerical experiments with iterative refinement of solutions of linear equations by ABS methods. Report DMSIA Univ Bergamo 32/96
11. Nicolai S, Spedicato E (1997) A bibliography of the ABS methods. OMS 8:171–183
12. Spedicato E, Bodon E (1989) Solving linear least squares by orthogonal factorization and pseudoinverse computation via the modified Huang algorithm in the ABS class. Computing 42:195–205
13. Spedicato E, Bodon E (1992) Numerical behaviour of the implicit QR algorithm in the ABS class for linear least squares. Ricerca Oper 22:43–55
14. Spedicato E, Bodon E (1993) Solution of linear least squares via the ABS algorithm. Math Program 58:111–136
15. Spedicato E, Vespucci MT (1993) Variations on the Gram-Schmidt and the Huang algorithms for linear systems: A numerical study. Appl Math 2:81–100
16. Wedderburn JHM (1934) Lectures on matrices. Colloq Publ Amer Math Soc
17. Zhang L (1995) An algorithm for the least Euclidean norm solution of a linear system of inequalities via the Huang ABS algorithm and the Goldfarb–Idnani strategy. Report DMSIA Univ Bergamo 95/2

ABS Algorithms for Optimization

Emilio Spedicato[1], Zunquan Xia[2], Liwei Zhang[2]
[1] Department Math., University Bergamo, Bergamo, Italy
[2] Department Applied Math., Dalian University Technol., Dalian, China

MSC2000: 65K05, 65K10

Article Outline

Keywords

Linear equations; Optimization; ABS methods; Quasi-Newton methods; Linear programming; Feasible direction methods; KT equations; Interior point methods

The *scaled ABS* (Abaffy–Broyden–Spedicato) class of algorithms, see [1] and ► ABS algorithms for linear equations and linear least squares, is a very general process for solving linear equations, realizing the so-called *Petrov–Galerkin approach*. In addition to solving general determined or underdetermined linear systems $Ax = b$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $m \leq n$, $\text{rank}(A) \leq m$, $A = [a_1, \ldots a_m]^\mathsf{T}$, ABS methods can also solve linear least squares problems and nonlinear algebraic equations. In this article we will consider applications of ABS methods to optimization problems. We will consider only the so-called *basic ABS class*, defined by the following procedure for solving $Ax = b$:

A) Let $x_1 \in \mathbb{R}^n$ be arbitrary, $H_1 \in \mathbb{R}^{n,n}$ be nonsingular arbitrary, set $i = 1$.
B) Compute $s_i = H_i\, a_i$. IF $s_i \neq 0$, go to C).
IF $s_i = 0$ and $\tau = a_i^\top x_i - b_i = 0$, THEN set $x_{i+1} = x_i$, $H_{i+1} = H_i$ and go to F), ELSE stop, the system has no solution.
C) Compute the search vector p_i by $p_i = H_i^\top\, z_i$, where $z_i \in \mathbb{R}^n$ is arbitrary save for the condition $a_i^\top\, H_i^\top\, z_i \neq 0$.
D) Update the estimate of the solution by $x_{i+1} = x_i - \alpha_i p_i$, where the stepsize α_i is given by $\alpha_i = (a_i^\top p_i - b_i)/a_i^\top p_i$.
E) Update the matrix H_i by $H_{i+1} = H_i - H_i a_i w_i^\top H_i / w_i^\top H_i a_i$, where $w_i \in \mathbb{R}^n$ is arbitrary save for the condition $w_i^\top H_i a_i \neq 0$.
F) IF $i = m$, THEN stop; x_{m+1} solves the system, ELSE increment i by one and go to B).

Among the properties of the ABS class the following is fundamental in the applications to optimization. Let

$m < n$ and, for simplicity, assume that rank$(A) = m$. Then the linear variety containing all solutions of the underdetermined system $Ax = b$ is represented by the vectors x of the form

$$x = x_{m+1} + H_{m+1}^{\top} q, \tag{1}$$

where $q \in \mathbb{R}^n$ is arbitrary. In the following the matrices generated by the ABS process will be called *Abaffians*. It is recalled that the matrix H_{i+1} can be represented in terms of either $2i$ vectors or of $n - i$ vectors, which is also true for the representation of the search vector p_i. The first representation is computationally convenient for systems where the number of equations is small (less than $n/2$), while the second one is suitable for problems where m is close to n. In the applications to optimization, the first case corresponds to problems with few constraints (many degrees of freedom), the second case to problems with many constraints (few degrees of freedom).

Among the algorithms of the basic ABS class, the following are particularly important.

a) The *implicit LU algorithm* is given by the choices $H_1 = I$, $z_i = w_i = e_i$, where e_i is the ith unit vector in $\mathbb{R}^n$. This algorithm is well defined if and only if A is regular (otherwise pivoting of the columns has to be performed, or of the equations, if $m = n$). Due to the special structure of the Abaffian induced by the parameter choices (the first i rows of H_{i+1} are identically zero, while the last $n - i$ columns are unit vectors) the maximum storage is $n^2/4$, hence 4 times less than for the classical LU factorization or twice less than for *Gaussian elimination*; the number of multiplications is $nm^2 - 2m^3/3$, hence, for $m = n$, $n^3/3$, i. e. the same as for Gaussian elimination or the LU factorization algorithm.

b) The *Huang algorithm* is obtained by the parameter choices $H_1 = I$, $z_i = w_i = a_i$. A mathematically equivalent, but numerically more stable, formulation of this algorithm is the so-called *modified Huang algorithm* where the search vectors and the Abaffians are given by formulas $p_i = H_i(H_i a_i)$ and $H_{i+1} = H_i - p_i p_i^{\top}/p_i^{\top} p_i$. The search vectors are orthogonal and are equal to the vectors obtained by applying the classical *Gram–Schmidt orthogonalization* procedure to the rows of A. If x_1 is the zero vector, then the vector x_{i+1} is the solution of least Euclidean norm of the first i equations and the solution x^+ of least Euclidean norm of the whole system is approached monotonically and from below by the sequence x_i.

c) The *implicit LX algorithm*, where 'L' refers to the lower triangular left factor while 'X' refers to the right factor, which is a matrix obtainable after row permutation of an upper triangular matrix, considered by Z. Xia, is defined by the choices $H_1 = I$, $z_i = w_i = e_{k_i}$ where k_i is an integer, $1 \le k_i \le n$, such that

$$e_{k_i}^{\top} H_i a_i \neq 0. \tag{2}$$

If A has full rank, from a property of the basic ABS class the vector $H_i a_i$ is nonzero, hence there is at least one index k_i such that (2) is satisfied. The implicit LX algorithm has the same overhead as the implicit LU algorithm, hence the same as Gaussian elimination, and the same storage requirement, i. e. less than Gaussian elimination or the LU factorization algorithm. It has the additional advantage of not requiring any condition on the matrix A, hence pivoting is not necessary. The structure of the Abaffian matrix is somewhat more complicated than for the implicit LU algorithm, the zero rows of H_{i+1} being now in the positions $k_1, \ldots, k_i$ and the columns that are unit vectors being in the positions that do not correspond to the already chosen indices k_i.

The vector p_i has $n - i$ zero components and its k_ith component is equal to one. It follows that if $x_1 = 0$, then x_{i+1} is a basic type solution of the first i equations, whose nonzero components correspond to the chosen indices k_i.

In this paper we will present the following applications of ABS methods to optimization problems. In Section 2 we describe a class of ABS related methods for the unconstrained optimization problem. In Section 3 we show how ABS methods provide the general solution of the quasi-Newton equation, also with sparsity and symmetry and we discuss how SPD solutions can be obtained. In Section 4 we present several special ABS methods for solving the Kuhn–Tucker equations. In Section 5 we consider the application of the implicit LX algorithm to the linear programming (LP) problem. In Section 6 we present ABS approaches to the general linearly constrained optimization problem, which unify linear and nonlinear problems.

A Class of ABS Projection Methods for Unconstrained Optimization

ABS methods can be applied directly to solve *unconstrained optimization* problems via the iteration $x_{i+1} = x_i - \alpha_i H_i^\top z_i$, where H_i is reset after n or less steps and z_i is chosen so that the descent condition holds, i. e. $g_i^\top H_i^\top z_i > 0$, with g_i the gradient of the function at x_i. If the function to be minimized is quadratic, one can identify the matrix A in the Abaffian update formula with the Hessian of the quadratic function. Defining a perturbed point x' by $x' = x_i - \beta\ v_i$ one has on quadratic functions $g' = g - \beta\ Av_i$, hence the update of the Abaffian takes the form $H_{i+1} = H_i - H_i y_i w_i^\top H_i / w_i^\top H_i y_i$, where $y_i = g' - g_i$. The above defined class has termination on quadratic functions and local superlinear (n-step Q-quadratic) rate of convergence on general functions. It is a special case of a class of projection methods developed in [7]. Almost no numerical results are available about the performance of the methods in this class.

Applications to Quasi-Newton Methods

ABS methods have been used to provide the general solution of the quasi-Newton equation, also with the additional conditions of symmetry, sparsity and positive definiteness. While the general solution of only the quasi-Newton equation was already known from [2], the explicit formulas obtained for the sparse symmetric case are new, and so is the way of constructing sparse SPD updates.

Let us consider the quasi-Newton equation defining the new approximation to a Jacobian or a Hessian, in the transpose form

$$d^\top B' = y^\top , \tag{3}$$

where $d = x' - x$, $y = g' - g$. We observe that (3) can be seen as a set of n linear underdetermined systems, each one having just one equation and differing only in the right-hand side. Hence the general solution can be obtained by one step of the ABS method. It can be written in the following way

$$B' = B - \frac{s(B^\top d - y)^\top}{d^\top s} + \left(I - \frac{sd^\top}{d^\top s}\right) Q\ , \tag{4}$$

where $Q \in \mathbb{R}^{n,\,n}$ is arbitrary and $s \in \mathbb{R}^n$ is arbitrary subject to $s^\intercal d \neq 0$. Formula (4), derived in [9], is equivalent to the formula in [2].

Now the conditions that some elements of B' should be zero, or have constant value or that B' should be symmetric can be written as the additional linear constraints, where b'_i is the ith column of B'

$$(b'_i)^\top e_k = \eta_{ij}\ , \tag{5}$$

where $\eta_{ij} = 0$ implies sparsity, $\eta_{ij} = \text{const}$ implies that some elements do not change their value and $\eta_{ij} = \eta_{ji}$ implies symmetry. The ABS algorithm can deal with these extra conditions, see [11], giving the solution in explicit form, columnwise in presence of symmetry. By adding the additional condition that the diagonal elements be sufficiently large, it is possible to obtain formulas where B' is quasi positive definite or quasi diagonally dominant, in the sense that the principal submatrix of order $n - 1$ is positive definite or diagonally dominant. It is not possible in general to force B' to be SPD, since SPD solutions may not exist, which is reflected in the fact that no additional conditions can be put on the last diagonal element, since the last column is fully determined by the $n - 1$ symmetry conditions and the quasi-Newton equation. This result can however be exploited to provide SPD approximations by imbedding the original minimization problem of n variables in a problem of $n + 1$ variables, whose solution with respect to the first n variables is the original solution (just set, for instance, $f(x') = f(x) + x_{n+1}^2$). This imbedding modifies the quasi-Newton equation so that SPD solutions exist.

ABS Methods for Kuhn–Tucker Equations

The *Kuhn–Tucker equations* (KT equations), which should more appropriately be named *Kantorovich–Karush–Kuhn–Tucker equations* (KKKT equations), are a special linear system, obtained by writing the optimality conditions of the problem of minimizing a quadratic function with Hessian G subject to the linear equality constraint $Cx = b$. They are the system $Ax = b$, where A is a symmetric indefinite matrix of the following form, with $G \in \mathbb{R}^{n,\,n}$, $C \in \mathbb{R}^{m,\,n}$

$$A = \begin{pmatrix} G & C^\top \\ C & 0 \end{pmatrix} . \tag{6}$$

If G is nonsingular, then A is nonsingular if and only if $CG^{-1}C^\intercal$ is nonsingular. Usually G is nonsingular, symmetric and positive definite, but this assumption, required by several classical solvers, is not necessary for the ABS solvers.

ABS classes for solving the KT problem can be derived in several ways. Observe that system (6) is equivalent to the two subsystems

$$Gp + C^\top z = g, \tag{7}$$
$$Cp = c, \tag{8}$$

where $x = (p^\intercal, z^\intercal)^\intercal$ and $b = (g^\intercal, C^\intercal)^\intercal$. The general solution of subsystem (8) has the form, see (1)

$$p = p_{m+1} + H_{m+1}^\top q, \tag{9}$$

with q arbitrary. The parameter choices made to construct p_{m+1} and H_{m+1} are arbitrary and define therefore a class of algorithms.

Since the KT equations have a unique solution, there must be a choice of q in (9) which makes p be the unique n-dimensional subvector defined by the first n components of the solution x. Notice that since H_{m+1} is singular, q is not uniquely defined (but would be uniquely defined if one takes the representation of the Abaffian in terms of $n - m$ vectors).

By multiplying equation (7) on the left by H_{m+1} and using the ABS property $H_{m+1}\ C^\intercal = 0$, we obtain the equation

$$H_{m+1}Gp = H_{m+1}g, \tag{10}$$

which does not contain z. Now there are two possibilities to determine p:

A1) Consider the system formed by equations (8) and (10). Such a system is solvable but overdetermined. Since $\text{rank}(H_{m+1}) = n - m$, m equations are recognized as dependent and are eliminated in step B) of any ABS algorithm applied to this system.

A2) In equation (10) substitute p with the expression of the general solution (9) obtaining

$$H_{m+1}GH_{m+1}^\top q = H_{m+1}g - H_{m+1}Gp_{m+1}. \tag{11}$$

The above system can be solved by any ABS method for a particular solution q, m equations being again removed at step B) of the ABS algorithm as linearly dependent.

Once p is determined, there are two approaches to determine z, namely:

B1) Solve by any ABS method the overdetermined compatible system

$$C^\top z = g - Gp \tag{12}$$

by removing at step B) of the ABS algorithm the $n - m$ dependent equations.

B2) Let $P = (p_1, \ldots p_m)$ be the matrix whose columns are the search vectors generated on the system $Cp = c$. Now $CP = L$, with L nonsingular lower diagonal. Multiplying equation (12) on the left by $P^\intercal$ we obtain a triangular system, defining z uniquely

$$L^\top z = P^\top g - P^\top Gp. \tag{13}$$

Extensive numerical testing has evaluated the accuracy of the above considered ABS algorithms for KT equations for certain choices of the ABS parameters (corresponding to the implicit LU algorithm with row pivoting and the modified Huang algorithm). The methods have been tested against classical methods, in particular the method of Aasen and methods using the QR factorization. The experiments have shown that some ABS methods are the most accurate, in both residual and solution error; moreover some ABS algorithms are cheaper in storage and in overhead, up to one order, especially for the case when m is close to n.

In many interior point methods the main computational cost is to compute the solution for a sequence of KT problems where only G, which is diagonal, changes. In such a case the ABS methods, which initially work on the matrix C, which is unchanged, are advantaged, particularly when m is large, where the dominant cubic term decreases with m and disappears for $m = n$, so that the overhead is dominated by second order terms. Again numerical experiments show that some ABS methods are more accurate than the classical ones. For details see [8].

Reformulation of the Simplex Method via the Implicit LX Algorithm

The implicit LX algorithm has a natural application to a reformulation of the simplex method for the LP prob-

lem in standard form, i. e. the problem

$$\begin{cases} \min & c^\top x \\ \text{s.t.} & Ax = b \\ & x \geq 0\,. \end{cases}$$

The applicability of the implicit LX method is a consequence of the fact that the iterate x_{i+1} generated by the method, started from the zero vector, is a basic type vector, with a unit component in the position k_i, non identically zero components corresponding to indices $j \in B_i$, where B_i is the set of indices of the unit vectors chosen as the z_i, w_i parameters, i. e. the set $B_i = (k_1, \ldots, k_i)$, while the components of x_{i+1} of indices in the set $N_i = N/B_i$ are identically zero, where $N = (1, \ldots n)$. Therefore, if the nonzero components are nonnegative, the point defines a vertex of the polytope containing the feasible points defined by the constraints of the LP problem.

In the simplex method one moves from a vertex to another one, according to some rules and usually reducing at each step the value of the function $c^\intercal x$. The direction along which one moves from a vertex to another one is an edge direction of the polytope and is determined by solving a linear system, whose coefficient matrix A_B, the *basic matrix*, is defined by m linearly independent columns of the matrix A, called the *basic columns*. Usually such a system is solved by the LU factorization method or occasionally by the *QR method*, see [5]. The new vertex is associated to a new basic matrix A_B', which is obtained by substituting one of the columns in A_B by a column of the matrix A_N, which comprises the columns of A that do not belong to A_B. The most efficient algorithm for solving the modified system, after the column interchange, is the *Forrest–Goldfarb method* [6], requiring m^2 multiplications. Notice that the classical simplex method requires m^2 storage for the matrix A_B plus mn storage for the matrix A, which must be kept in general to provide the columns for the exchange.

The application of the implicit LX method to the simplex method, developed in [4,10,13,17] exploits the fact that in the implicit LX algorithm the interchange of a jth column in A_B with a kth column in A_N corresponds to the interchange of a previously chosen parameter vector $z_j = w_j = e_j$ with a new parameter $z_k = w_k = e_k$. This operation is a special case of the perturbation of the Abaffian after a change in the parameters and can be done using a general formula of [15], without explicit use of the kth column in A_N. Moreover since all quantities which are needed for the construction of the search direction (the edge direction) and for the interchange criteria can as well be implemented without explicit use of the columns of A, it follows that the ABS approach needs only the storage of the matrix H_{m+1}, which, in the case of the implicit LX algorithm, has a cost of at most $n^2/4$. Therefore for values of m close to n the storage required by the ABS formulation is about 8 times less than for the classical simplex method.

Here we give the basic formulas of the simplex method in the classical and in the ABS formulation. The column in A_N substituting an old column in A_B is often taken as the column with minimal relative cost. In terms of the ABS formulation this is equivalent to minimize with respect to $i \in N_m$ the scalar $\eta_i = c^\intercal H^\intercal e_i$. Let N^* be the index chosen in this way. The column in A_B to be exchanged is usually chosen with the criterion of the maximum displacement along an edge which keeps the basic variables nonnegative. Define $\omega_i = x^\intercal e_i / e_i^\top H^\intercal e_{N^*}$, where x is the current basic feasible solution. Then the above criterion is equivalent to minimize ω_i with respect the set of indices $i \in B_m$ such that

$$e_i^\top H^\top e_{N^*} > 0. \tag{14}$$

Notice that $H^\intercal e_{N^*} \neq 0$ and that an index i such that (14) is satisfied always exists, unless x is a solution of the LP problem.

The update of the Abaffian after the interchange of the unit vectors, which corresponds to the update of the LU factors after the interchange of the basic with the nonbasic column, is given by the following formula

$$H' = H - (He_{B^*} - e_{B^*}) \frac{e_{N^*}^\top H}{e_{N^*}^\top H e_{B^*}}. \tag{15}$$

The search direction d, which in the classical formulation is obtained by solving the system $A_B d = -Ae_{N^*}$, is given by $d = H_{m+1}^\top e_{N^*}$, hence at no cost. Finally, the relative cost vector r, classically given by $r = c - A^\intercal A_B^{-1} c_B$, where c_B consists of the components of c with indices corresponding to those of the basic columns, is simply given by $r = H_{m+1} c$.

Let us now consider the computational cost of update (15). Since $H e_{B^*}$ has at most $n - m$ nonzero components, while $H^\intercal e_{N^*}$ has at most m, no more than $m(n - m)$ multiplications are required. The update is most expensive for $m = n/2$ and gets cheaper the smaller m is or the closer it is to n. In the dual steepest edge Forrest–Goldfarb method [6] the overhead for replacing a column is m^2, hence formula (15) is faster for $m > n/2$ and is recommended on overhead considerations for m sufficiently large. However we notice that ABS updates having a $O(m^2)$ cost can also be obtained by using the representation of the Abaffian in terms of $2m$ vectors. No computational experience has been obtained till now on the new ABS formulation of the simplex method.

Finally, a generalization of the *simplex method*, based upon the use of the Huang algorithm started with a suitable singular matrix, has been developed in [16]. In this formulation the solution is approached by points lying on a face of the polytope. Whenever the point hits a vertex the remaining iterates move among vertices and the method is reduced to the simplex method.

ABS Unification of Feasible Direction Methods for Minimization with Linear Constraints

ABS algorithms can be used to provide a unification of feasible point methods for nonlinear minimization with linear constraints, including as a special case the LP problem. Let us first consider the problem with only linear equality constraints:

$$\begin{cases} \min\limits_{x\in\mathbb{R}^n} & f(x) \\ \text{s.t.} & Ax = b \\ & A \in \mathbb{R}^{m,n}, \quad m \le n, \\ & \operatorname{rank}(A) = m. \end{cases}$$

Let x_1 be a feasible starting point; then for an iteration procedure of the form $x_{i+1} = x_i - \alpha_i d_i$, the search direction will generate feasible points if and only if

$$Ad_i = 0. \tag{16}$$

Solving the underdetermined system (16) for d_i by the ABS algorithm, the solution can be written in the following form, taking, without loss of generality, the zero vector as a special solution

$$d_i = H_{m+1}^\top q, \tag{17}$$

where the matrix H_{m+1} depends on the arbitrary choice of the parameters H_1, w_i and v_i used in solving (16) and $q \in \mathbb{R}^n$ is arbitrary. Hence the general feasible direction iteration has the form

$$x_{i+1} = x_i - \alpha_i H_{m+1}^\top q. \tag{18}$$

The search direction is a descent direction if and only if $d^\intercal \nabla f(x) = q^\intercal H_{m+1} \nabla f(x) > 0$. Such a condition can always be satisfied by choice of q unless $H_{m+1} \nabla f(x) = 0$, which implies, from the null space structure of H_{m+1}, that $\nabla f(x) = A^\intercal \lambda$ for some λ, hence that x_{i+1} is a KT point and λ is the vector of the Lagrange multipliers. When x_{i+1} is not a KT point, it is immediate to see that the search direction is a descent directions if we select q as $q = WH_{m+1} \nabla f(x)$, where W is a symmetric and positive definite matrix.

Particular well-known algorithms from the literature are obtained by the following choices of q, with $W = I$:

- The *Wolfe reduced gradient method*. Here, H_{m+1} is constructed by the implicit LU (or the implicit LX) algorithm.
- The *Rosen gradient projection method*. Here, H_{m+1} is built using the Huang algorithm.
- The *Goldfarb–Idnani method*. Here, H_{m+1} is built via the modification of the Huang algorithm where H_1 is a symmetric positive definite matrix approximating the inverse Hessian of $f(x)$.

If there are inequalities two approaches are possible:

A) The *active set* approach. In this approach the set of linear equality constraints is modified at every iteration by adding and/or dropping some of the linear inequality constraints. Adding or deleting a single constraint can be done, for every ABS algorithm, in order two operations, see [15]. In the ABS reformulation of the Goldfarb–Idnani method, the initial matrix is related to a quasi-Newton approximation of the Hessian and an efficient update of the Abaffian after a change in the initial matrix is discussed in [14].

B) The *standard form* approach. In this approach, by introducing slack variables, the problem with both types of linear constraints is written in the equivalent form

$$\begin{cases} \min & f(x) \\ \text{s.t.} & Ax = b \\ & x \geq 0. \end{cases}$$

The following general iteration, started with x_1 a feasible point, generates a sequence of feasible points for the problem in standard form

$$x_{i+1} = x_i - \alpha_i \beta_i H_{m+1} \nabla f(x), \tag{19}$$

where the parameter α_i can be chosen by a line search along the vector $H_{m+1} \nabla f(x)$, while the relaxation parameter $\beta_i > 0$ is selected to avoid that the new point has some negative components.

If $f(x)$ is nonlinear, then H_{m+1} can be determined once and for all at the first step, since $\nabla f(x)$ generally changes from iteration to iteration, therefore modifying the search direction. If, however, $f(x) = c^\intercal x$ is linear (we have then the LP problem) to modify the search direction we need to change H_{m+1}. As observed before, the simplex method is obtained by constructing H_{m+1} with the implicit LX algorithm, every step of the method corresponding to a change of the parameters e_{k_i}. It can be shown, see [13], that the *method of Karmarkar* (equivalent to an earlier *method of Evtushenko* [3]), corresponds to using the generalized Huang algorithm, with initial matrix $H_1 = \text{Diag}(x_i)$ changing from iteration to iteration. Another method, faster than Karmarkar's and having superlinear against linear rate of convergence and $O(\sqrt{n})$ against $O(n)$ complexity, again first proposed by Y. Evtushenko, is obtained by the generalized Huang algorithm with initial matrix $H_1 = \text{Diag}(x_i^2)$.

See also

- ▶ ABS Algorithms for Linear Equations and Linear Least Squares
- ▶ Gauss–Newton Method: Least Squares, Relation to Newton's Method
- ▶ Generalized Total Least Squares
- ▶ Least Squares Orthogonal Polynomials
- ▶ Least Squares Problems
- ▶ Nonlinear Least Squares: Newton-type Methods
- ▶ Nonlinear Least Squares Problems
- ▶ Nonlinear Least Squares: Trust Region Methods

References

1. Abaffy J, Spedicato E (1989) ABS projection algorithms: Mathematical techniques for linear and nonlinear equations. Horwood, Westergate
2. Adachi N (1971) On variable metric algorithms. JOTA 7:391–409
3. Evtushenko Y (1974) Two numerical methods of solving nonlinear programming problems. Soviet Dokl Akad Nauk 251:420–423
4. Feng E, Wang XM, Wang XL (1997) On the application of the ABS algorithm to linear programming and linear complementarity. Optim Methods Softw 8:133–142
5. Fletcher R (1997) Dense factors of sparse matrices. In: Buhmann MD, Iserles A (eds) Approximation Theory and Optimization. Cambridge Univ. Press, Cambridge, pp 145–166
6. Forrest JH, Goldfarb D (1992) Steepest edge simplex algorithms for linear programming. Math Program 57:341–374
7. Psenichny BN, Danilin YM (1978) Numerical methods in extremal problems. MIR, Moscow
8. Spedicato E, Chen Z, Bodon E (1996) ABS methods for KT equations. In: Di Pillo G, Giannessi F (eds) Nonlinear Optimization and Applications. Plenum, New York, pp 345–359
9. Spedicato E, Xia Z (1992) Finding general solutions of the quasi-Newton equation in the ABS approach. Optim Methods Softw 1:273–281
10. Spedicato E, Xia Z, Zhang L (1995) Reformulation of the simplex algorithm via the ABS algorithm. Preprint Univ Bergamo
11. Spedicato E, Zhao J (1992) Explicit general solution of the quasi-Newton equation with sparsity and symmetry. Optim Methods Softw 2:311–319
12. Xia Z (1995) ABS generalization and formulation of the interior point method. Preprint Univ Bergamo
13. Xia Z (1995) ABS reformulation of some versions of the simplex method for linear programming. Report DMSIA Univ Bergamo 10/95
14. Xia Z, Liu Y, Zhang L (1992) Application of a representation of ABS updating matrices to linearly constrained optimization. Northeast Oper Res 7:1–9
15. Zhang L (1995) Updating of Abaffian matrices under perturbation in W and A. Report DMSIA Univ Bergamo 95/16
16. Zhang L (1997) On the ABS algorithm with singular initial matrix and its application to linear programming. Optim Methods Softw 8:143–156
17. Zhang L, Xia ZH (1995) Application of the implicit LX algorithm to the simplex method. Report DMSIA Univ Bergamo 9/95

Adaptive Convexification in Semi-Infinite Optimization

Oliver Stein
School of Economics and Business Engineering, University of Karlsruhe, Karlsruhe, Germany

MSC2000: 90C34, 90C33, 90C26, 65K05

Article Outline

Synonyms

ACA

Introduction

The adaptive convexification algorithm is a method to solve semi-infinite optimization problems via a sequence of *feasible iterates*. Its main idea [6] is to adaptively construct convex relaxations of the lower level problem, replace the relaxed lower level problems equivalently by their Karush–Kuhn–Tucker conditions, and solve the resulting mathematical programs with complementarity constraints. The convex relaxations are constructed with ideas from the αBB method of global optimization.

Feasibility in Semi-Infinite Optimization

In a (standard) semi-infinite optimization problem a finite-dimensional decision variable is subject to infinitely many inequality constraints. For adaptive convexification one assumes the form

$$SIP\colon \min_{x\in X} f(x) \quad \text{subject to} \quad g(x,y)\le 0, \quad \text{for all } y\in[0,1]$$

with objective function $f\in C^2(\mathbb{R}^n,\mathbb{R})$, constraint function $g\in C^2(\mathbb{R}^n\times\mathbb{R},\mathbb{R})$, a box constraint set $X=[x^\ell,x^u]\subset\mathbb{R}^n$ with $x^\ell<x^u\in\mathbb{R}^n$, and the set of infinitely many indices $Y=[0,1]$. Adaptive convexification easily generalizes to problems with additional inequality and equality constraints, a finite number of semi-infinite constraints as well as higher-dimensional box index sets [6]. Reviews on semi-infinite programming are given in [8,13], and [9,14,15] overview the existing numerical methods.

Classical numerical methods for *SIP* suffer from the drawback that their approximations of the feasible set $X\cap M$ with

$$M = \{\, x\in\mathbb{R}^n \mid g(x,y)\le 0 \text{ for all } y\in[0,1]\,\}$$

may contain infeasible points. In fact, discretization and exchange methods approximate M by finitely many inequalities corresponding to finitely many indices in $Y=[0,1]$, yielding an outer approximation of M, and reduction based methods solve the Karush–Kuhn–Tucker system of *SIP* by a Newton-SQP approach. As a consequence, the iterates of these methods are not necessarily feasible for *SIP*, but only their limit might be. On the other hand, a first method producing feasible iterates for *SIP* was presented in the articles [3,4], where a branch-and-bound framework for the global solution of *SIP* generates convergent sequences of lower and upper bounds for the globally optimal value.

In fact, checking feasibility of a given point $\bar{x}\in\mathbb{R}^n$ is the crucial problem in semi-infinite optimization. Clearly we have $\bar{x}\in M$ if and only if $\varphi(\bar{x})\le 0$ holds with the function

$$\varphi\colon \mathbb{R}^n\to\mathbb{R},\ x\mapsto \max_{y\in[0,1]} g(x,y)\,.$$

The latter function is the optimal value function of the so-called lower level problem of *SIP*,

$$Q(x)\colon \max_{y\in\mathbb{R}} g(x,y) \quad \text{subject to} \quad 0\le y\le 1\,.$$

The difficulty lies in the fact that $\varphi(\bar{x})$ is the *globally* optimal value of $Q(\bar{x})$ which might be hard to determine numerically. In fact, standard NLP solvers can

only be expected to produce a *local* maximizer y_{loc} of $Q(\bar{x})$ which is not necessarily a global maximizer y_{glob}. Even if $g(\bar{x}, y_{\text{loc}}) \le 0$ is satisfied, $\bar{x}$ might be infeasible since $g(\bar{x}, y_{\text{loc}}) \le 0 < \varphi(\bar{x}) = g(\bar{x}, y_{\text{glob}})$ may hold.

Convex Lower Level Problems

Assume for a moment that $Q(x)$ is a convex optimization problem for all $x \in X$, that is, $g(x, \cdot)$ is concave on $Y = [0, 1]$ for these x. An approach developed for so-called generalized semi-infinite programs from [18,19] then takes advantage of the fact that the solution set of a differentiable convex lower level problem satisfying a constraint qualification is characterized by its first order optimality condition. In fact, *SIP* and the Stackelberg game

$$SG\colon \min_{x,y} f(x) \quad \text{subject to} \quad g(x, y) \le 0, \quad \text{and } y \text{ solves } Q(x)$$

are equivalent problems, and the restriction 'y solves $Q(x)$' in *SG* can be equivalently replaced by its Karush–Kuhn–Tucker condition. For this reformulation we use that the Lagrange function of $Q(x)$,

$$\mathcal{L}(x, y, \gamma_\ell, \gamma_u) \;=\; g(x, y) + \gamma_\ell\, y + \gamma_u(1 - y),$$

satisfies

$$\nabla_y \mathcal{L}(x, y, \gamma_\ell, \gamma_u) \;=\; \nabla_y g(x, y) + \gamma_\ell - \gamma_u$$

and obtain that the Stackelberg game is equivalent to the following mathematical program with complementarity constraints:

$$\begin{aligned} MPCC\colon \min_{x,y,\gamma_\ell,\gamma_u} \; & f(x) \text{ subject to } g(x, y) \le 0 \\ & \nabla_y g(x, y) + \gamma_\ell - \gamma_u = 0 \\ & \gamma_\ell\, y = 0 \\ & \gamma_u(1 - y) = 0 \\ & \gamma_\ell,\ \gamma_u \ge 0 \\ & y,\ 1 - y \ge 0\,. \end{aligned}$$

Overviews of solution methods for *MPCC* are given in [10,11,17]. One approach to solve *MPCC* is the reformulation of the complementarity constraints by a so-called NCP function, that is, a function $\phi\colon \mathbb{R}^2 \to \mathbb{R}$ with

$$\phi(a, b) \;=\; 0 \quad \text{if and only if} \quad a \ge 0\,, \quad b \ge 0, \quad a\, b = 0\,.$$

For numerical purposes one can regularize these nondifferentiable NCP functions. Although MPCC does not necessarily have to be solved via the NCP function formulation, in the following we will use NCP functions to keep the notation concise. In fact, *MPCC* can be equivalently rewritten as the nonsmooth problem

$$\begin{aligned} P\colon \min_{x,y,\gamma_\ell,\gamma_u} \; & f(x) \quad \text{subject to} \quad g(x, y) \le 0 \\ & \nabla_y g(x, y) + \gamma_\ell - \gamma_u = 0 \\ & \phi(\gamma_\ell, y) = 0 \\ & \phi(\gamma_u, 1 - y) = 0\,. \end{aligned}$$

The αBB Method

In αBB, a convex underestimator of a nonconvex function is constructed by decomposing it into a sum of nonconvex terms of special type (e. g., linear, bilinear, trilinear, fractional, fractional trilinear, convex, univariate concave) and nonconvex terms of arbitrary type. The first type is then replaced by its convex envelope or very tight convex underestimators which are already known. A complete list of the tight convex underestimators of the above special type nonconvex terms is provided in [5].

For the ease of presentation, here we will treat all terms as arbitrarily nonconvex. For these terms, αBB constructs convex underestimators by adding a quadratic relaxation function ψ. With the obvious modification we use this approach to construct a concave overestimator for a nonconcave function $g\colon [y^\ell, y^u] \to \mathbb{R}$ being C^2 on an open neighborhood of $[y^\ell, y^u]$. With

$$\psi(y; \alpha, y^\ell, y^u) \;=\; \frac{\alpha}{2}(y - y^\ell)(y^u - y) \tag{1}$$

we put

$$\tilde{g}(y; \alpha, y^\ell, y^u) \;=\; g(y) + \psi(y; \alpha, y^\ell, y^u)\,.$$

In the sequel we will suppress the dependence of $\tilde{g}$ on y^ℓ, y^u. For $\alpha \ge 0$ the function $\tilde{g}$ clearly is an overestimator of g on $[y^\ell, y^u]$, and it coincides with g at the endpoints y^ℓ, y^u of the domain. Moreover, $\tilde{g}$ is twice continuously differentiable with second derivative

$$\nabla_y^2 \tilde{g}(y; \alpha) \;=\; \nabla^2 g(y) - \alpha$$

on $[y^\ell, y^u]$. Consequently $\tilde{g}$ is concave on $[y^\ell, y^u]$ for

$$\alpha \geq \max_{y\in[y^\ell,y^u]} \nabla^2 g(y) \tag{2}$$

(cf. also [1,2]). The computation of α thus involves a global optimization problem itself. Note, however, that one may use *any* upper bound for the right-hand side in (2). Such upper bounds can be provided by interval methods (see, e. g., [5,7,12]). An α satisfying (2) is called *convexification parameter*.

Combining these facts shows that for

$$\alpha \geq \max\left(0, \max_{y\in[y^\ell,y^u]} \nabla^2 g(y)\right)$$

the function $\tilde{g}(y;\alpha)$ is a concave overestimator of g on $[y^\ell, y^u]$.

Formulation

For $N \in \mathcal{N}$ let $0 = \eta^0 < \eta^1 < \ldots < \eta^{N-1} < \eta^N = 1$ define a subdivision of $Y = [0,1]$, that is, with $K = \{1,\ldots,N\}$ and

$$Y^k = [\eta^{k-1}, \eta^k], \quad k \in K,$$

we have

$$Y = \bigcup_{k\in K} Y_k .$$

A trivial but very useful observation is that the single semi-infinite constraint

$$g(x,y) \leq 0 \quad \text{for all} \quad y \in Y$$

is equivalent to the finitely many semi-infinite constraints

$$g(x,y) \leq 0 \quad \text{for all} \quad y \in Y^k,\ k \in K .$$

Given a subdivision, one can construct concave overestimators for each of these finitely many semi-infinite constraints, solve the corresponding optimization problem, and adaptively refine the subdivision.

The following lemma formulates the obvious fact that replacing g by overestimators on each subdivision node Y^k results in an approximation of M by *feasible* points.

Lemma 1 *For each $k \in K$ let $g^k\colon X \times Y^k \to \mathbb{R}$, and let $\bar{x} \in X$ be given such that for all $k \in K$ and all $y \in Y^k$ we have $g(\bar{x},y) \leq g^k(\bar{x},y)$. Then the constraints*

$$g^k(\bar{x},y) \leq 0 \quad \textit{for all} \quad y \in Y^k,\ k \in K,$$

entail $\bar{x} \in M$.

αBB for the Lower Level

For the construction of these overestimators one uses ideas of the αBB method. In fact, for each $k \in K$ we put

$$g^k\colon X\times Y^k \to \mathbb{R}, (x,y) \mapsto g(x,y)+\psi(y;\alpha_k,\eta^{k-1},\eta^k) \tag{3}$$

with the quadratic relaxation function ψ from (1) and

$$\alpha_k > \max\left(0, \max_{(x,y)\in X\times Y^k} \nabla_y^2 g(x,y)\right). \tag{4}$$

Note that the latter condition on α_k is uniform in x. We emphasize that with the single bound

$$\bar{\alpha} > \max\left(0, \max_{(x,y)\in X\times Y} \nabla_y^2 g(x,y)\right) \tag{5}$$

the choices $\alpha_k := \bar{\alpha}$ satisfy (4) for all $k \in K$. Moreover, the α_k can always be chosen such that $\alpha_k \leq \bar{\alpha}$, $k \in K$.

The following properties of g^k are easily verified.

Lemma 2 ([6]) *For each $k \in K$ let g^k be given by (3). Then the following holds:*
(i) For all $(x,y) \in X \times Y^k$ we have $g(x,y) \leq g^k(x,y)$.
(ii) For all $x \in X$, the function $g^k(x,\cdot)$ is concave on Y^k.

Now consider the following approximation of the feasible set M, where $E = \{\eta^k \mid k \in K\}$ denotes the set of subdivision points, and α the vector of convexification parameters:

$$M_{\alpha BB}(E,\alpha) = \{\, x \in \mathbb{R}^n \mid g^k(x,y) \leq 0\,, \text{ for all } y \in Y^k,\ k \in K\,\}.$$

By Lemma 1 and Lemma 2(i) we have $M_{\alpha BB}(E,\alpha) \subset M$. This means that any solution concept for

$$SIP_{\alpha BB}(E,\alpha)\colon \min_{x\in X} f(x) \quad \text{subject to} \quad x \in M_{\alpha BB}(E,\alpha),$$

be it global solutions, local solutions or stationary points, will at least lead to feasible points of *SIP* (provided that $SIP_{\alpha BB}(E,\alpha)$ is consistent).

The problem $SIP_{\alpha BB}(E,\alpha)$ has finitely many lower level problems $Q^k(x)$, $k \in K$, with

$$Q^k(x)\colon \max_{y\in\mathbb{R}} g^k(x,y) \quad \text{subject to} \quad \eta^{k-1} \le y \le \eta^k\,.$$

Since the inequality (4) is strict, the convex problem $Q^k(x)$ has a unique solution $y^k(x)$ for each $k \in K$ and $x \in X$. Recall that $y \in Y^k$ is called active for the constraint $\max_{y\in Y^k} g^k(x,y) \le 0$ at $\bar{x}$ if $g^k(\bar{x},y) = 0$ holds. By the uniqueness of the global solution of $Q^k(\bar{x})$ there exists *at most one* active index for each $k \in K$, namely $y^k(\bar{x})$. Thus, one can consider the finite active index sets

$$\begin{aligned} K_0(\bar{x}) &= \{\, k \in K \mid g^k(\bar{x}, y^k(\bar{x})) = 0 \,\},\\ Y_0^{\alpha BB}(\bar{x}) &= \{\, y^k(\bar{x}) \mid k \in K_0(\bar{x}) \,\}\,. \end{aligned}$$

The MPCC Reformulation

Following the ideas to treat convex lower level problems, y^k solves $Q^k(x)$ if and only if $(x, y^k, \gamma_\ell^k, \gamma_u^k)$ solves the system

$$\begin{aligned} \nabla_y g^k(x,y) + \gamma_\ell - \gamma_u &= 0\\ \phi(\gamma_\ell, y - \eta^{k-1}) &= 0\\ \phi(\gamma_u, \eta^k - y) &= 0 \end{aligned}$$

with some γ_ℓ^k, γ_u^k, and ϕ denoting some NCP function. With

$$\begin{aligned} w &:= (x, y^k, \gamma_\ell^k, \gamma_u^k, k \in K)\\ F(w) &:= f(x)\\ G^k(w;E,\alpha) &:= g(x,y^k) + \frac{\alpha_k}{2}(y^k - \eta^{k-1})(\eta^k - y^k)\\ H^k(w;E,\alpha) &:= \begin{pmatrix} \nabla_y g(x,y^k) + \alpha_k\left(\frac{\eta^{k-1}+\eta^k}{2} - y^k\right) + \gamma_\ell^k - \gamma_u^k\\ \phi(\gamma_\ell^k, y^k - \eta^{k-1})\\ \phi(\gamma_u^k, \eta^k - y^k) \end{pmatrix} \end{aligned}$$

one can thus replace $SIP_{\alpha BB}(E,\alpha)$ equivalently by the nonsmooth problem

$$\begin{aligned} P(E,\alpha)\colon \min_w\ & F(w) \quad \text{subject to}\\ & G^k(w;E,\alpha) \le 0,\\ & H^k(w;E,\alpha) = 0, \quad k \in K\,. \end{aligned}$$

The latter problem can be solved to local optimality by MPCC algorithms [10,11,17]. For a local minimizer $\bar{w}$ of $P(E,\alpha)$ the subvector $\bar{x}$ of $\bar{w}$ is a local minimizer and, hence, a stationary point of $SIP_{\alpha BB}(E,\alpha)$.

Method

The main idea of the adaptive convexification algorithm is to compute a stationary point $\bar{x}$ of $SIP_{\alpha BB}(E,\alpha)$ by the approach from the previous section, and terminate if $\bar{x}$ is also stationary for *SIP* within given tolerances. If $\bar{x}$ is not stationary it refines the subdivision E in the spirit of exchange methods [8,15] by adding the active indices $Y_0^{\alpha BB}(\bar{x})$ to E, and constructs a refined problem $SIP_{\alpha BB}(E \cup Y_0^{\alpha BB}(\bar{x}), \tilde{\alpha})$ by the following procedure. Note that, in view of Carathéodory's theorem, the number of elements of $Y_0^{\alpha BB}(\bar{x})$ may be bounded by $n+1$.

Refinement Step

For any $\tilde{\eta} \in Y_0^{\alpha BB}(\bar{x})$, let $k \in K$ be the index with $\tilde{\eta} \in [\eta^{k-1}, \eta^k]$. Put $Y^{k,1} = [\eta^{k-1}, \tilde{\eta}]$, $Y^{k,2} = [\tilde{\eta}, \eta^k]$, let $\alpha_{k,1}$ and $\alpha_{k,2}$ be the corresponding convexification parameters, put

$$\begin{aligned} g^{k,1}(x,y) &= g(x,y) + \frac{\alpha_{k,1}}{2}(y - \eta^{k-1})(\tilde{\eta} - y),\\ g^{k,2}(x,y) &= g(x,y) + \frac{\alpha_{k,2}}{2}(y - \tilde{\eta})(\eta^k - y), \end{aligned}$$

and define $M_{\alpha BB}(E \cup \{\tilde{\eta}\}, \tilde{\alpha})$ by replacing the constraint

$$g^k(x,y) \le 0, \quad \text{for all } y \in Y^k$$

in $M_{\alpha BB}(E,\alpha)$ by the two new constraints

$$g^{k,i}(x,y) \le 0, \quad \text{for all} \quad y \in Y^{k,i}, \quad i = 1,2\,,$$

and by replacing the entry α_k of α by the two new entries $\alpha_{k,i}$, $i = 1,2$.

The Algorithm

The point $\bar{x}$ is stationary for $SIP_{\alpha BB}(E,\alpha)$ (in the sense of Fritz John) if $\bar{x} \in M_{\alpha BB}(E,\alpha)$ and if there exist $y^k \in Y_0^{\alpha BB}(\bar{x})$, $1 \le k \le n+1$, and $(\kappa,\lambda) \in S^{n+1}$ (the $(n+1)-$dimensional standard simplex) with

$$\begin{aligned} \kappa \nabla f(\bar{x}) + \sum_{k=1}^{n+1} \lambda_k \nabla_x g(\bar{x}, y^k) &= 0\\ \lambda_k \cdot g^k(\bar{x}, y^k) &= 0,\ 1 \le k \le n+1\,. \end{aligned}$$

For the adaptive convexification algorithm the notions of *active index*, *stationarity*, and *set unification* are relaxed by certain tolerances.

Definition 1 For $\varepsilon_{\text{act}}, \varepsilon_{\text{stat}}, \varepsilon_{\cup} > 0$ we say that

(i) y^k is ε_{act}-active for g^k at $\bar{x}$ if $g^k(\bar{x}, y^k) \in [-\varepsilon_{\text{act}}, 0]$,

(ii) $\bar{x}$ is $\varepsilon_{\text{stat}}$-stationary for *SIP* with ε_{act}-active indices if $\bar{x} \in M$ and if there exist $y^k \in Y$, $1 \le k \le n+1$, and $(\kappa, \lambda) \in S^{n+1}$ such that

$$\left\| \kappa \nabla f(\bar{x}) + \sum_{k=1}^{n+1} \lambda_k \nabla_x g(\bar{x}, y^k) \right\| \le \varepsilon_{\text{stat}}$$
$$\lambda_k \cdot g(\bar{x}, y^k) \in [-\lambda_k \cdot \varepsilon_{\text{act}}, 0], \quad 1 \le k \le n+1,$$

hold, and

(iii) the $\varepsilon_{\cup}$-union of E and $\tilde{\eta}$ is $E \cup \{\tilde{\eta}\}$ if

$$\min\{\tilde{\eta} - \eta^{k-1}, \eta^k - \tilde{\eta}\} > \varepsilon_{\cup} \cdot (\eta^k - \eta^{k-1})$$

holds for the $k \in K$ with $\tilde{\eta} \in [\eta^{k-1}, \eta^k]$, and E otherwise (i. e., $\tilde{\eta}$ is not unified with E if its distance from E is too small).

In [6] it is shown that Algorithm 1 is well-defined, convergent and finitely terminating. Furthermore, the following feasibility result holds.

Theorem 2 ([6]) *Let $(x^\nu)_\nu$ be a sequence of points generated by Algorithm 1. Then all x^ν, $\nu \in \mathbb{N}$, are feasible for SIP, the sequence $(x^\nu)_\nu$ has an accumulation point, each such accumulation point x^* is feasible for SIP, and $f(x^*)$ provides an upper bound for the optimal value of SIP.*

Numerical examples for the performance of the method from Chebyshev approximation and design centering are given in [6].

A Consistent Initial Approximation

Even if the feasible set M of *SIP* is consistent, there is no guarantee that its approximations $M_{\alpha BB}(E, \alpha)$ are also consistent. For Step 1 of Algorithm 1 [6] suggests the following phase I approach: use Algorithm 1 to construct adaptive convexifications of

$$SIP^{ph.I}: \min_{(x,z) \in X \times \mathbb{R}} z \quad \text{subject to} \quad g(x, y) \le z \quad \text{for all } y \in [0, 1]$$

Algorithm 1
(Adaptive convexification algorithm)

Step 1: Determine a uniform convexification parameter $\bar{\alpha}$ with (5), choose $N \in \mathbb{N}$, $\eta^k \in Y$ and $\alpha_k \le \bar{\alpha}$, $k \in K = \{1, \ldots, N\}$, such that $SIP_{\alpha BB}(E, \alpha)$ is consistent, as well as tolerances $\varepsilon_{\text{act}}, \varepsilon_{\text{stat}}, \varepsilon_{\cup} > 0$ with $\varepsilon_{\cup} \le 2\varepsilon_{\text{act}}/\bar{\alpha}$.

Step 2: By solving $P(E, \alpha)$, compute a stationary point x of $SIP_{\alpha BB}(E, \alpha)$ with ε_{act}−active indices y^k, $1 \le k \le n+1$, and multipliers (κ, λ).

Step 3: Terminate if x is $\varepsilon_{\text{stat}}$−stationary for *SIP* with $(2\varepsilon_{\text{act}})$-active indices y^k, $1 \le k \le n+1$, from Step 2 and multipliers (κ, λ) from Step 2. Otherwise construct a new set $\tilde{E}$ of subdivision points as the $\varepsilon_{\cup}$-union of E and $\{y^k | 1 \le k \le n+1\}$, and perform a refinement step for the elements in $\tilde{E} \setminus E$ to construct a new feasible set $M_{\alpha BB}(\tilde{E}, \tilde{\alpha})$.

Step 4: Put $E = \tilde{E}$, $\alpha = \tilde{\alpha}$, and go to Step 2.

Adaptive Convexification in Semi-Infinite Optimization, Algorithm 1

until a feasible point $(\bar{x}, \bar{z})$ with $\bar{z} \le 0$ of $SIP^{ph.I}_{\alpha BB}(E, \alpha)$ is found with some subdivision E and convexification parameters α. The point $\bar{x}$ is then obviously also feasible for $SIP_{\alpha BB}(E, \alpha)$ and can be used as an initial point to solve the latter problem. Due to the possible nonconvexity of the upper level problem of *SIP*, this phase I approach is not necessarily successful, but possible remedies for this situation are given in [6].

To initialize Algorithm 1 for phase I, select some point $\bar{x}$ in the box X and put $E^1 = \{0, 1\}$, that is, $Y^1 = Y = [0, 1]$. Compute α_1 according to (4) and solve the convex optimization problem $Q^1(\bar{x})$ with standard software. With its optimal value $\bar{z}$, the point $(\bar{x}, \bar{z})$ is feasible for $SIP^{ph.I}_{\alpha BB}(E^1, \alpha_1)$.

A Certificate for Global Optimality

After termination of Algorithm 1 one can exploit that the set $E \subset [0, 1]$ contains indices that should also yield a good outer approximation of M. The optimal value of the problem

$$P_{\text{outer}}: \min_{x \in X} f(x) \quad \text{subject to} \quad g(x, \eta) \le 0, \ \eta \in E,$$

yields a rigorous *lower* bound for the optimal value of *SIP*. If P_{outer} can actually be solved to global optimality (e. g., if a standard NLP solver is used, due to convexity with respect to x), then a comparison of this lower bound for the optimal value of *SIP* with the upper bound from Algorithm 1 can yield a certificate of global optimality for *SIP* up to some tolerance.

Conclusions

The adaptive convexification algorithm provides an easily implementable way to solve semi-infinite optimization problems with feasible iterates. To explain its basic ideas, in [6] the algorithm is presented in its simplest form. It can be improved in a number of ways, for example in the magnitude of the convexification parameters and in their adaptive refinement, or by using other convexification techniques. Although the numerical results from [6] are very promising, further work is needed on error estimates on the numerical solution of the auxiliary problem $P(E, \alpha)$, which is assumed to be solved to exact local optimality by the present adaptive convexification algorithm.

See also

- ► αBB Algorithm
- ► Bilevel Optimization: Feasibility Test and Flexibility Index
- ► Convex Discrete Optimization
- ► Generalized Semi-infinite Programming: Optimality Conditions

References

1. Adjiman CS, Androulakis IP, Floudas CA (1998) A global optimization method, αBB, for general twice-differentiable constrained NLPs – I: theoretical advances. Comput Chem Eng 22:1137–1158
2. Adjiman CS, Androulakis IP, Floudas CA (1998) A global optimization method, αBB, for general twice-differentiable constrained NLPs – II: implementation and computational results. Comput Chem Eng 22:1159–1179
3. Bhattacharjee B, Green WH Jr, Barton PI (2005) Interval methods for semi-infinite programs. Comput Optim Appl 30:63–93
4. Bhattacharjee B, Lemonidis P, Green WH Jr, Barton PI (2005) Global solution of semi-infinite programs. Math Program 103:283–307
5. Floudas CA (2000) Deterministic global optimization, theory, methods and applications. Kluwer, Dordrecht
6. Floudas CA, Stein O (2007) The adaptive convexification algorithm: a feasible point method for semi-infinite programming. SIAM J Optim 18:1187–1208
7. Hansen E (1992) Global optimization using interval analysis. Dekker, New York
8. Hettich R, Kortanek KO (1993) Semi-infinite programming: theory, methods, and applications. SIAM Rev 35:380–429
9. Hettich R, Zencke P (1982) Numerische Methoden der Approximation und semi-infiniten Optimierung. Teubner, Stuttgart
10. Kočvara M, Outrata J, Zowe J (1998) Nonsmooth approach to optimization problems with equilibrium constraints: theory, applications and numerical results. Kluwer, Dordrecht
11. Luo Z, Pang J, Ralph D (1996) Mathematical programs with equilibrium constraints. Cambridge University Press, Cambridge
12. Neumaier A (1990) Interval methods for systems of equations. Cambridge University Press, Cambridge
13. Polak E (1987) On the mathematical foundation of nondifferentiable optimization in engineering design. SIAM Rev 29:21–89
14. Polak E (1997) Optimization, algorithms and consistent approximations. Springer, Berlin
15. Reemtsen R, Görner S (1998) Numerical methods for semi-infinite programming: a survey. In: Reemtsen R, Rückmann J-J (eds) Semi-infinite programming. Kluwer, Boston, pp 195–275
16. Reemtsen R, Rückmann J-J (eds) (1998) Semi-infinite programming. Kluwer, Boston
17. Scholtes S, Stöhr M (1999) Exact penalization of mathematical programs with equilibrium constraints. SIAM J Control Optim 37:617–652
18. Stein O (2003) Bi-level strategies in semi-infinite programming. Kluwer, Boston
19. Stein O, Still G (2003) Solving semi-infinite optimization problems with interior point techniques. SIAM J Control Optim 42:769–788

Adaptive Global Search

J. M. Calvin
Department Computer and Information Sci.,
New Jersey Institute Techn., Newark, USA

MSC2000: 60J65, 68Q25

Article Outline

Keywords

Average case complexity; Adaptive algorithm; Wiener process; Randomized algorithms

This article contains a survey of some well known facts about the complexity of *global optimization*, and also describes some results concerning the *average-case complexity*.

Consider the following optimization problem. Given a class F of objective functions f defined on a compact subset of d-dimensional Euclidean space, the goal is to approximate the global minimum of f based on evaluation of the function at sequentially selected points. The focus will be on the error after n observations

$$\Delta_n = \Delta_n(f) = f_n - f^*,$$

where f_n is the smallest of the first n observed function values (other approximations besides f_n are often considered).

Complexity of optimization is usually studied in the worst- or average-case setting. In order for a *worst-case analysis* to be useful the class of objective functions F must be quite restricted. Consider the case where F is a subset of the continuous functions on a compact set. It is convenient to consider the class $F = C^r([0, 1]^d)$ of real-valued functions on $[0, 1]^d$ with continuous derivatives up to order $r \geq 0$. Suppose that $r > 0$ and f^r is bounded. In this case $\Theta(\epsilon^{-d/r})$ function evaluations are needed to ensure that the error is at most ϵ for any $f \in F$; see [8].

An *adaptive algorithm* is one for which the $(n + 1)$st observation point is determined as a function of the previous observations, while a nonadaptive algorithm chooses each point independently of the function values. In the worst-case setting, adaptation does not help much under quite general assumptions. If F is convex and symmetric (in the sense that $-F = F$), then the maximum error under an adaptive algorithm with n observations is not smaller than the maximum error of a nonadaptive method with $n + 1$ observations; see [4].

Virtually all global optimization methods in practical use are adaptive. For a survey of such methods see [6,9]. The fact that the worst-case performance can not be significantly improved with adaptation leads to consideration of alternative settings that may be more appropriate. One such setting is the average-case setting, in which a probability measure P on F is chosen. The object of study is then the sequence of random variables $\Delta_n(f)$, and the questions include under what conditions (for what algorithms) the error converges to zero and for convergent algorithms the speed of convergence. While the average-case error is often defined as the mathematical expectation of the error, it is useful to take a broader view, and consider for example convergence in probability of $a_n\Delta_n$ for some normalizing sequence $\{a_n\}$.

With the average-case setting one can consider less restricted classes F than in the worst-case setting. As F gets larger, the worst-case deviates more and more from the average case, but may occur on only a small portion of the set F. Even for continuous functions the worst-case is arbitrarily bad.

Most of what is known about the average-case complexity of optimization is in the one-dimensional setting under the *Wiener probability measure* on $C([0, 1])$. Under the Wiener measure, the increments $f(t)-f(s)$ have a normal distribution with mean zero and variance $t-s$, and are independent for disjoint intervals. Almost every f is nowhere differentiable, and the set of local minima is dense in the unit interval. One can thus think of the Wiener measure as corresponding to assuming 'only' continuity; i. e., a worst-case probabilistic assumption.

K. Ritter proved [5] that the best nonadaptive algorithms have error of order $n^{-1/2}$ after n function evaluations; the optimal order is achieved by observing at equally spaced points. Since the choice of each new observation point does not depend on any of the previous observations, the computation can be carried out in parallel. Thus under the Wiener measure, the optimal nonadaptive order of convergence can be accomplished with an algorithm that has computational cost that grows linearly with the number of observations and uses constant storage. This gives the base on which to compare adaptive algorithms.

Recent studies (as of 2000) have formally established the improved power of adaptive methods in the average-case setting by analyzing the convergence rates of certain adaptive algorithms. A *randomized algorithm* is described in [1] with the property that for any $0 < \delta < 1$, a version can be constructed so that under the Wiener measure, the error converges to zero at rate $n^{-1+\delta}$. This

algorithm maintains a memory of two past observation values, and the computational cost grows linearly with the number of iterations. Therefore, the convergence rate of this adaptive algorithm improves from the non-adaptive $n^{-1/2}$ rate to $n^{-1+\delta}$ with only a constant increase in storage.

Algorithms based on a random model for the objective function are well-suited to average-case analysis. H. Kushner proposed [3] a global optimization method based on modeling the objective function as a Wiener process. Let $\{z_n\}$ be a sequence of positive numbers, and let the $(n+1)$st point be chosen to maximize the probability that the new function value is less than the previously observed minimum minus z_n. This class of algorithms, often called *P-algorithms*, was given a formal justification by A. Žilinskas [7].

By allowing the $\{z_n\}$ to depend on the past observations instead of being a fixed deterministic sequence, it is possible to establish a much better convergence rate than that of the randomized algorithm described above. In [2] an algorithm was constructed with the property that the error converges to zero for any continuous function and furthermore, the error is of order e^{-nc_n}, where $\{c_n\}$ (a parameter of the algorithm) is a deterministic sequence that can be chosen to approach zero at an arbitrarily slow rate. Notice that the convergence rate is now almost exponential in the number of observations n. The computational cost of the algorithm grows quadratically, and the storage increases linearly, since all past observations must be stored.

See also

- ► Adaptive Simulated Annealing and its Application to Protein Folding
- ► Global Optimization Based on Statistical Models

References

1. Calvin J (1997) Average performance of a class of adaptive algorithms for global optimization. Ann Appl Probab 7:711–730
2. Calvin J (2001) A one-dimensional optimization algorithm and its convergence rate under the Wiener measure. J Complexity
3. Kushner H (1962) A versatile stochastic model of a function of unknown and time varying form. J Math Anal Appl 5: 150–167
4. Novak E (1988) Deterministic and stochastic error bounds in numerical analysis. Lecture Notes in Mathematics, vol 1349. Springer, Berlin
5. Ritter K (1990) Approximation and optimization on the Wiener space. J Complexity 6:337–364
6. Törn A, Žilinskas A (1989) Global optimization. Springer, Berlin
7. Žilinskas A (1985) Axiomatic characterization of global optimization algorithm and investigation of its search strategy. OR Lett 4:35–39
8. Wasilkowski G (1992) On average complexity of global optimization problems. Math Program 57:313–324
9. Zhigljavsky A (1991) Theory of global random search. Kluwer, Dordrecht

Adaptive Simulated Annealing and its Application to Protein Folding

ASA

Ruth Pachter, Zhiqiang Wang
Air Force Research Laboratory Materials & Manufacturing Directorate, Wright–Patterson AFB, Wright–Patterson AFB, USA

MSC2000: 92C05

Article Outline

Keywords

Optimization; Adaptive simulated annealing; Protein folding; Met-Enkephalin; Poly(L-Alanine)

The adaptive simulated annealing (ASA) algorithm [3] has been shown to be faster and more efficient than

simulated annealing and genetic algorithms [4]. In this article we first outline some of the aspects of the method and specific computational details, and then review the application of the ASA method to biomolecular structure determination [15], specifically for Met-Enkephalin and a model of the poly(L-Alanine) system.

The ASA Method

For a system described by a cost function $E(\{p^i\})$, where all p^i $(i = 1, \ldots, D)$ are parameters (variables) having ranges $[A_i, B_i]$, the ASA procedure to find the global optimum of 'E' contains the following elements.

Monte-Carlo Configurations

As the kth point is saved in a D-dimensional configuration space, the new point p^i_{k+1} is generated by:

$$p^i_{k+1} = p^i_k + y^i(B_i - A_i), \tag{1}$$

where the random variables y^i in $[-1, 1]$ (non-uniform) are generated from a random number u^i uniformly distributed in $[0, 1]$, and the temperature T_i associated with parameter p^i, as follows:

$$y^i = \mathrm{sgn}(u^i - 0.5)T_i\left[\left(1 + \frac{1}{T_i}\right)^{|2u^i - 1|} - 1\right]. \tag{2}$$

Note that if p^i_{k+1} is outside the range of $[A_i, B_i]$ it will be disregarded, with the process being repeated until it falls within the range. The choice of y^i is made so that the probability density distribution of the D parameters will satisfy the distribution of each parameter:

$$g^i(y^i; T_i) = \frac{1}{2(|y^i| + T_i)(1 + \frac{1}{T_i})}, \tag{3}$$

which is chosen to ensure that any point in configuration space can be sampled infinitely often in annealing time with a cooling schedule outlined below. Thus, at any annealing time k_0, the probability of not generating a global optimum, given infinite time, is zero:

$$\prod_{k=k_0}^{\infty}(1 - g_k) = 0, \tag{4}$$

where g_k is the distribution function at time step k. Note that all atoms move at each Monte-Carlo step in ASA. A Boltzmann acceptance criterion is then applied to the difference in the cost function.

Annealing Schedule

The annealing schedule for each parameter temperature from a starting temperature T_{0i}, and similarly for the cost temperature, is given by:

$$T_i(k_i) = T_{0i}\exp\left(-c_i k_i^{\frac{1}{D}}\right), \tag{5}$$

where c_i and k_i are the annealing scale and ASA step of parameter p^i. The index for re-annealing the cost function is determined by the number of accepted points instead of the number of generated points as is being used for the parameters. This choice was made since the Boltzmann acceptance criterion uses an exponential distribution which is not as 'fat-tailed' as the ASA distribution used for the parameters.

Re-Annealing

The temperatures may be periodically re-annealed or re-scaled according to the sensitivity of the cost function. At any given annealing time, the temperature range is 'stretched out' over the relatively insensitive parameters, thus guiding the search 'fairly' among the parameters. The sensitivity of the energy to each parameter is calculated by:

$$S_i = \frac{\partial E}{\partial p^i}, \tag{6}$$

while the re-annealing temperature is determined by:

$$T_i(k') = T_i(k)\frac{S_i}{S_{\max}}. \tag{7}$$

In this way, less sensitive parameters anneal faster. This is done approximately every 100 accepted events.

For comparison, within conventional simulated annealing [6] the cooling schedule is given by:

$$ST_k = T_0 e^{-(1-c)k} \quad (0 < c < 1), \tag{8}$$

where trial and error are applied to determine the annealing rate c—1 as well as the starting temperature T_0. A Monte-Carlo simulation is carried out at each temperature step k with temperature T_k. This cooling schedule is equivalent to $T_{k+1} = T_k\, c$.

The ASA algorithm is mostly suited to problems for which less is known about the system, and has proven to be more robust than other simulated annealing techniques for complex problems with multiple local minima, e. g., as compared to Cauchy annealing where T_i

$= T_0/k$, and Boltzmann annealing where $T_i = T_0/\ln k$. The annealing schedule in (8), faster than ASA for a large dimension of D, does not pass the infinitely often annealing-time test in (4), and is therefore referred to as *simulated quench* in the terminology of ASA.

Application to Protein Folding

Computational Details

A protein can be defined as a biopolymer of hundreds of amino acids bonded by peptide bonds, while the test models in this article contain less amino acids, namely oligopeptides. The Met-Enkephalin model was constructed as (H-Tyr-Gly-Gly-Phe-Met-OH). For 14(L-Alanine), the neutral —NH_2 and —COOH end groups were substituted at the termini. The conformation of a protein is described by the dihedral angles of the backbone (ϕ_i, ψ_i), side-chains (χ_i^j), and peptide bond (ω_i, often very close to 180°). Therefore, the conformation determination of the most stable protein is to find the set of $\{\phi, \psi, \chi, \omega\}$ which give the global minimal potential energy $E(\phi, \psi, \chi, \omega)$. Within the ASA nomenclature, the 'cost function' is the potential energy, while a 'parameter' is a dihedral angle variable.

Conformational analyses using conventional simulated annealing were carried out previously [9,11]. The modifications in these works include moving a number of dihedral angles in a Monte-Carlo step; adjusting the maximum deviation of the variables as the temperature decreases to insure that the acceptance ratio is more than 25%; and treating the variables differently according to their importance in the folding process, e. g., by increasing sampling for the backbone dihedral angles as compared to those of the side-chains. It is interesting to point out that within ASA these modifications are implicitly included.

Each ASA run in our work was started from a random initial configuration $\{\phi, \psi, \chi\}$. The dihedral angle ω was fixed to 180° in all of the ASA runs. The initial temperature was determined by the average energy of 5 or 10 random samplings, and a full search range of the dihedral angles $(-\pi, \pi)$ was set. The typical maximum number of calls to the energy function was 30000. An ASA run was terminated if it repeated the best energy value for 3 or 5 re-annealing cycles (each cycle generates 100 configurations). Further refinement of the final ASA optimized configuration was carried out by using the local minimizer SUMSL [1], or the conjugate gradient method. The combination of the ASA application and a local minimizer improved the efficiency of the search.

The ASA calculation is governed by various control parameters [3], for which the most important setting is the annealing rate for the temperatures of 'cost' and 'parameters', determined by the so-called 'temperature-ratio-scale' (the ratio of the final to the initial temperature after certain annealing steps) and the 'cost-parameter-scale'. The control parameters were varied to improve the search efficiency. Adequate control parameters used for obtaining the results reported in this study were: 'temperature-ratio-scale' = 10^{-4}; 'cost-parameter-scale' = 0.5. These parameter settings correspond to an annealing rate for energy of $c_{\text{cost}} = 3.6$, and for all dihedral angles of $c_{\text{parameter}} = 7.2$. Note that the annealing rate for all dihedral angles was chosen to be the same.

Met-Enkephalin

Met-Enkephalin has a complicated energy surface [11,16]. The lowest energy for Met-Enkephalin was found to be −12.9 kcal/mol with the force field being ECEPP/2 (Empirical Conformation Energy Program for Peptides) [8]. With all ω fixed, the lowest energy was found to be −10.7 kcal/mol by MCM [14]. Using different initial conformations and control parameter settings of the cooling schedule as described above, 55 independent ASA runs were carried out. Table 1 summarizes the energy distribution of these calculations. Most of the ASA calculations result in energies in the range of −8 to −3 kcal/mol, with 7 of the results determining conformations having energies that are only 3 kcal/mol above the known lowest energy, thus exhibiting the effectiveness of the approach. Moreover, as the range of search was somewhat narrowed, almost all of the ASA runs reach the global energy minimum.

Adaptive Simulated Annealing and its Application to Protein Folding, Table 1
The energy (in kcal/mol) distribution of ASA runs for Met-Enkephalin using a full search range

Energy	< -8	$(-8, -5)$	$(-5, -3)$	> -3
No. of runs	7	19	19	10

Adaptive Simulated Annealing and its Application to Protein Folding, Table 2
Energy and dihedral angles of the lowest energy conformations of Met-Enkephalin calculated by ASA. RMSD1 is the root-mean-square deviation (in Å) for backbone atoms, while RMSD2 is for all atoms

	A0	A	1	2	3	4
E	−12.9	−10.7	−10.6	−10.4	−10.1	−8.5
ϕ_1	−86	−87	−87	−87	−87	−87
ψ_1	156	154	153	153	156	153
ϕ_2	−155	−162	−161	−162	−166	−166
ψ_2	84	71	72	75	87	72
ϕ_3	84	64	64	63	68	63
ψ_3	−74	−93	−94	−95	−91	−97
ϕ_4	−137	−82	−83	−81	−103	−74
ψ_4	19	−29	−26	−30	−13	−30
ϕ_5	−164	−81	−79	−76	−76	−82
ψ_5	160	144	133	132	137	143
χ_1^1	−173	−180	180	179	−166	−180
χ_1^2	79	−111	−110	71	88	73
χ_1^3	−166	145	145	−35	−148	−179
χ_4^1	59	180	72	−179	71	179
χ_4^2	−86	−100	84	−100	−93	−100
χ_5^1	53	−65	−171	−173	−65	−65
χ_5^2	175	−179	176	176	−178	−179
χ_5^3	−180	−179	180	179	−178	−179
χ_5^4	−58	−180	−60	60	−178	−179
RMSD1		0	0.04	0.07	0.51	0.26
RMSD2		0	2.52	1.92	2.08	1.29

For the full range search, we identified three conformations with energies of −10.6, −10.4, and −10.1 kcal/mol, that exhibit the configuration of the known lowest geometry of −10.7 kcal/mol. Table 2 lists the conformations of these lowest energy configurations, as well as an additional low energy structure. Conformations A0 and A are the lowest-energy conformations with ω nonfixed and fixed, respectively, taken from [11,14]. The first two conformations, #1 and #2, have almost the same backbone configuration as that of A (−10.7 kcal/mol), with a backbone root-mean-square deviation (RMSD) of only 0.04 and 0.07Å, respectively. The all-atom RMSD of the listed conformations with energies ranging from −8.5 to −10.6 kcal/mol are about 2Å. For conformations #1 and #2, the noted differences are in the side-chains, corresponding to a 0.1 and 0.3 kcal/mol difference in energy, respectively.

Adaptive Simulated Annealing and its Application to Protein Folding, Table 3
The conformation of a model 14(L-Alanine) peptide as calculated by ASA

	2	3	4	5	6
ϕ	−99.4	−68.2	−68.0	−69.3	−66.9
ψ	158.1	−34.3	−38.8	−38.5	−38.6
	7	8	9	10	11
ϕ	−68.3	−66.7	−68.8	−67.1	−69.4
ψ	−39.2	−38.0	−38.7	−37.7	−39.6
	12	13	14	15	
ϕ	−65.0	−67.2	−87.7	−75.9	
ψ	−40.0	−44.6	65.8	−40.1	

Poly(L-Alanine)

The ASA algorithm was applied to a model of (L-Alanine) that is known to assume a dominant right-handed α-helical structure [13]. For a search range of dihedral angles that include both the right-handed (RH) α-helix and the β-sheet region in the Ramachandran's diagram, ψ: (−115°, − 180°) and ϕ: (− 115°, 0°), it was significant to find RH α-helices with $\phi \approx -68°$ and $\psi \approx -38°$ in all backbones except those near the end-groups, as shown in Table 3. The energy of such a geometry is typically −10.2 kcal/mol after a local minimization. The energy surfaces of the RH α-helical regions were found to be less complex than those of Met-Enkephalin. These results are consistent with a previous study [16].

Conclusion

The adaptive simulated annealing as a global optimization method intrinsically includes some of the modifications of conventional simulated annealing used for biomolecular structure determination. As applied to Met-Enkephalin, the performance of ASA is comparable to the simulated annealing study reported in [12], while better than the one reported in [11], although some differences other than the algorithms are noted. Utilizing a partial search range improves the efficiency significantly, showing that ASA may be useful for refinement of a molecular structure predicted or measured by other methods. A dominant right-handed α-helical conformation was found for the 14 residue (L-Alanine) model, with deviations observed only near the end groups.

Recent Studies and Future Directions

Recent studies have shown improved efficiency in the conformational search of Met-Enkephalin, *e.g.*, the so-called conformation space annealing (CSA), which combines the ideas of genetic algorithms, simulated annealing, a build up procedure, and local minimization [7]. The use of the multicanonical ensemble algorithm (ME) (one of the generalized-ensemble algorithms [2]), allows free random walks in energy space, escaping from any energy barrier. Both the ME and CSA algorithms outperform genetic algorithms (GA), simulated annealing (SA), GA with minimization (GAM) and Monte-Carlo with minimization (MCM). Our own work (unpublished) and the work in ref. [5] both show that simple GA alone underperforms simulated annealing for the Met-Enkephalin conformational search problem. Table 4 compares these algorithms for efficiency (the number of evaluations of energy and energy gradient, or the number of local minimizations) and effectiveness (the number of runs reaching the ground state conformation (hits) versus the number of total independent runs). Caution should be exercised since some differences exist between these studies, such as the version of the ECEPP potential used, the treatment of the peptide dihedral angle ω, etc. Ground state conformations are those having energy within approximately 1eV from the known global minimum energy. Note that the generalized-ensemble method can be carried out with both Monte-Carlo and molecular dynamics.

Adaptive Simulated Annealing and its Application to Protein Folding, Table 4
Comparison of the conformation search efficiency and effectiveness of Met-Enkephalin using different algorithms. N_E, $N_{\nabla E}$, and N_{minz} are the number of the evaluations of energy, energy gradient, and number of local minimizations of each run, in the unit of 10^3

	hits/total	N_E	$N_{\nabla E}$	N_{minz}
ME [2]	10/10	< 1900	0	0
MCM [11]	24/24	$*$	$*$	15
GAM [10]	5/5	$*$	$*$	50
ME [2]	18/20	950	0	0
CSA [7]	99/100	300	250	5
ME [2]	21/50	400	0	0
CSA [7]	50/100	170	130	2.6
SA [2]	8/20	1000	0	0
GA [5]	$< 1/27$	100	0	0.001

$*$: The total number of $E, \nabla E$ evaluations are not given, but can be estimated based on roughly 100 evaluations for each minimization.

In comparison to the studies summarized in Table 4, ASA seems to be using too small a number of function evaluations. Optimizing control parameters such as the annealing schedule and increasing the number of energy evaluations may improve the effectiveness. Search efficiency could also be improved by adopting parallellization to achieve scalable simulation for various algorithms. Extensive research on the protein conformational search using various hybrids of genetic algorithms and parallelization is in progress (as of 1999).

See also

- ▶ Adaptive Global Search
- ▶ Bayesian Global Optimization
- ▶ Genetic Algorithms
- ▶ Genetic Algorithms for Protein Structure Prediction
- ▶ Global Optimization Based on Statistical Models
- ▶ Global Optimization in Lennard–Jones and Morse Clusters
- ▶ Global Optimization in Protein Folding
- ▶ Molecular Structure Determination: Convex Global Underestimation
- ▶ Monte-Carlo Simulated Annealing in Protein Folding
- ▶ Multiple Minima Problem in Protein Folding: αBB Global Optimization Approach
- ▶ Packet Annealing
- ▶ Phase Problem in X-ray Crystallography: Shake and Bake Approach
- ▶ Protein Folding: Generalized-ensemble Algorithms
- ▶ Random Search Methods
- ▶ Simulated Annealing
- ▶ Simulated Annealing Methods in Protein Folding
- ▶ Stochastic Global Optimization: Stopping Rules
- ▶ Stochastic Global Optimization: Two-phase Methods

References

1. Gay DM (1983) Subroutines for unconstrained minimization using a model/trust-region approach. ACM Trans Math Softw 9:503

2. Hansmann UHE (1998) Generalized ensembles: A new way of simulating proteins. Phys A 254:15
3. Ingber L (1989) Very fast simulated re-annealing. Math Comput Modelling 12:967 ASA code is available from: ftp.alumni.caltech.edu:pub/ingber
4. Ingber L, Rosen B (1992) Genetic algorithm and very fast simulated re-annealing: A comparison. Math Comput Modelling 16:87
5. Jin AY, Leung FY, Weaver DF (1997) Development of a novel genetic algorithm search method (GAP1.0) for exploring peptide conformational space. J Comput Chem 18:1971
6. Kirkpatrick S, Gelatt CD, Vecchi MP Jr (1983) Optimization by simulated annealing. Science 220:671
7. Lee J, Scheraga HA, Rackovsky S (1997) New optimization method for conformational energy calculations on polypeptides: conformational space annealing. J Comput Chem 18:222
8. Li Z, Scheraga HA (1987) Monte Carlo-minimization approach to the multiminima problem in protein folding. Proc Nat Aca Sci USA 84:6611
9. Li Z, Scheraga HA (1988) Structure and free energy of complex thermodynamic systems. J Mol Struct (Theochem) 179:333
10. Merkle LD, Lamont GB, Gates GH, Pachter R (May, 1996) Hybrid genetic algorithms for minimization of polypeptide specific energy model. Proc. IEEE Int. Conf. Evolutionary Computation, p 192
11. Nayeem A, Vila J, Scheraga HA (1991) A comparative study of the simulated-annealing and Monte Carlo-with minimization approaches to the minimum-energy structures of polypeptides: Met-Enkephalin. J Comput Chem 12:594
12. Okamoto Y, Kikuchi T, Kawai H (1992) Prediction of low-energy structure of Met-Enkephalin by Monte Carlo simulated annealing. Chem Lett (Chem Soc Japan):1275
13. Piela L, Scheraga HA (1987) On the multiple-minima problem in the conformational analysis of polypeptides: I. backbone degrees of freedom for a perturbed α-helix. Biopolymers 26:S33
14. Vasquez M (1999) Private communication
15. Wang Z, Pachter R (1997) Prediction of polypeptide conformation by the adaptive simulated annealing approach. J Comput Chem 18:323
16. Wilson SR, Cui W (1990) Applications of simulated annealing to peptides. Biopolymers 29:225

Affine Sets and Functions

LEONIDAS PITSOULIS
Princeton University, Princeton, USA

MSC2000: 51E15, 32B15, 51N20

Article Outline

Keywords

Linear algebra; Convex analysis

A subset S of $\mathbf{R}^n$ is an *affine set* if

$$(1-\lambda)x + \lambda y \in S,$$

for any $x, y \in S$ and $\lambda \in \mathbf{R}$. A function f: $\mathbf{R}^n \to \mathbf{R}$ is an *affine function* if f is finite, convex and concave (cf. ► Convex max-functions).

See also

► Linear Programming
► Linear Space

References

1. Rockafellar RT (1970) Convex analysis. Princeton Univ. Press, Princeton

Airline Optimization

GANG YU[1], BENJAMIN THENGVALL[2]
[1] Department Management Sci. and Information Systems Red McCombs School of Business, University Texas at Austin, Austin, USA
[2] CALEB Technologies Corp., Austin, USA

MSC2000: 90B06, 90C06, 90C08, 90C35, 90C90

Article Outline

Keywords

Network design and schedule construction; Fleet assignment; Aircraft routing; Crew scheduling; Revenue management; Irregular operations; Air traffic control and ground delay programs

The airline industry was one of the first to apply operations research methodology and techniques on a large scale. As early as the late 1950s, operations researchers were beginning to study how the developing fields of mathematical programming could be used to address a number of very difficult problems faced by the airline industry. Since that time many airline related problems have been the topics of active research [26]. Most optimization-related research in the airline industry can be placed in one of the following areas:

- network design and schedule construction;
- fleet assignment;
- aircraft routing;
- crew scheduling;
- revenue management;
- irregular operations;
- air traffic control and ground delay programs.

In the following, each of these problem areas will be defined along with a brief discussion of some of the operations research techniques that have been applied to solve them. The majority of applications utilize network-based models. Solution of these models range from traditional mathematical programming approaches to a variety of novel heuristic approaches. A very brief selection of references is also provided.

Construction of flight schedules is the starting point for all other airline optimization problems and is a critical operational planning task faced by an airline. The *flight schedule* defines a set of flight segments that an airline will service along with corresponding origin and destination points and departure and arrival times for each flight segment. An airline's decision to offer certain flights will depend in large part on market demand forecasts, available aircraft operating characteristics, available manpower, and the behavior of competing airlines [11,12].

Of course, prior to the construction of flight schedules, an airline must decide which markets it will serve. Before the 1978 'Airline Deregulation Act', airlines had to fly routes as assigned by the Civil Aeronautics Board regardless of the demand for service. During this period, most airlines emphasized long point-to-point routes. Since deregulation, airlines have gained the freedom to choose which markets to serve and how often to serve them. This change led to a fundamental shift in most airlines routing strategies from point-to-point flight networks to hub-and-spoke oriented flight networks. This, in turn, led to new research activities for finding optimal hub [3,18] and maintenance base [13] locations.

Following network design and schedule construction, an aircraft type must be assigned to each flight segment in the schedule. This is called the *fleet assignment problem*. Airlines generally operate a number of different fleet types, each having different characteristics and costs such as seating capacity, landing weights, and crew and fuel costs. The majority of fleet assignment methods represent the flight schedule via some variant of a time-space network with flight arcs between stations and inventory arcs at each station. A *multicommodity network flow problem* can then be formulated with arcs and nodes duplicated as appropriate for all fleets that can take a particular flight. Side constraints must be implemented to ensure each flight segment is assigned to only one fleet. In domestic fleet assignment problems, a common simplifying assumption is that every flight is flown every day of the week. Under this assumption, the network model need only account for one day's flights and a looping arc connects the end of the day with the beginning. The resulting models are mixed integer programs [1,16,27,30].

Aircraft routing is a fleet by fleet process of assigning individual aircraft to fly each flight segment assigned to a particular fleet. A primary consideration at this stage is *maintenance* requirements mandated by the Federal Aviation Administration. There are different types of maintenance activities that must be performed after a given number of flight hours. The majority of these maintenance activities can be performed overnight; however, not all stations are equipped with proper maintenance facilities for all fleets. During the aircraft routing process, individual aircraft from each fleet must be assigned to fly all flight segments assigned to that fleet in a manner that provides maintenance opportunities for all aircraft at appropriate stations within the required time intervals. This problem has been formulated and solved in a number of ways including as a general integer programming problem solved by Lagrangian relaxation [9] and as a set partitioning problem solved with a branch and bound algorithm [10].

As described above, the problems of fleet assignment and aircraft routing have been historically solved in a sequential manner. Recently, work has been done to solve these problems simultaneously using a string-

based model and a branch and price solution approach [5].

Crew scheduling, like aircraft routing, is done following fleet assignment. The first of two sequentially solved crew scheduling problems is the crew pairing problem. A crew pairing is a sequence of flight legs beginning and ending at a crew base that satisfies all governmental and contractual restrictions (some times called legalities). These crew pairings generally cover a period of 2–5 days. The problem is to find a minimum cost set of such crew pairings such that all flight segments are covered. This problem has generally been modeled as a set partitioning problem in which pairings are enumerated or generated dynamically [15,17]. Other attempts to solve this problem have employed a decomposition approach based on graph partitioning [4] and a linear programming relaxation of a set covering problem [21]. Often a practice called *deadheading* is used to reposition flight crews in which a crew will fly a flight segment as passengers. Therefore, in solving the crew-pairing problem, all flight segments must be covered, but they may be covered by more than one crew.

The second problem to be solved relating to crew scheduling is the monthly crew rostering problem. This is the problem of assigning individual crew members to crew pairings to create their monthly schedules. These schedules must incorporate time off, training periods, and other contractual obligations. Generally, a *preferential bidding system* is used to make the assignments in which each personalized schedule takes into account an employee's pre-assigned activities and weighted bids representing their preferences. While the crew pairing problem has been widely studied, a limited number of publications have dealt with the monthly crew rostering problem. Approaches include an integer programming scheme [14] and a network model [24].

Revenue management is the problem of determining fare classes for each flight in the flight schedule as well as the allocation of available seats to each fare class. Not only are seats on an airplane partitioned physically into sections such as first class and coach, but also seats in the same section are generally priced at many different levels. The goal is to maximize the expected revenue from a particular flight segment by finding the proper balance between gaining additional revenue by selling more inexpensive seats and losing revenue by turning away higher fare customers. A standard assumption is that fare classes are filled sequential from the lowest to the highest. This is often the case where discounted fares are offered in advance, while last minute tickets are sold at a premium. Recent research includes a probabilistic decision model [6], a dynamic programming formulation [31] and some calculus-based booking policies [8].

When faced with a lack of resources, airlines often are not able to fly their published flight schedule. This is frequently the result of aircraft mechanical difficulties, inclement weather, or crew shortages. As situations like these arise, decisions must be made to deal with the shortage of resources in a manner that returns the airline to the originally planned flight schedule in a timely fashion while attempting to reduce operational cost and keep passengers satisfied. This general situation is called the *airline irregular operations problem* and it involves aircraft, crew, gates, and passenger recovery.

The aircraft schedule recovery problem deals with re-routing aircraft during irregular operations. This problem has received significant attention among irregular operations topics; papers dealing with crew scheduling during irregular operations have only recently started to appear [28,35]. Most approaches for dealing with aircraft schedule recovery have been based on network models. Some early models were pure networks [19]. Recently, more comprehensive models have been developed that better represent the problem, but are more difficult to solve as side constraints have been added to the otherwise network structure of these problems [2,33,36]. In practice, many airlines use heuristic methods to solve these problems as their real-time nature does not allow for lengthy optimization run times.

Closely related to the irregular operations problem is the *ground delay problem* in air traffic control. Ground delay is a program implemented by the Federal Aviation Administration in cases of station congestion. During ground delay, aircraft departing for a congested station are held on the ground before departure. The rational for this behavior is that ground delays are less expensive and safer than airborne delays. Several optimization models have been formulated to decrease the total minutes of delay experienced throughout the system during a ground delay program. These problems have generally been modeled as integer programs ([22,23]), but the problem has also been solved using

stochastic linear programming [25] and by heuristic methods [34].

Optimization based methods have also been applied to a myriad of other airline related topics such as gate assignment [7], fuel management [29], short term fleet assignment swapping [32], demand modeling [20], and others. Airline industry is an exciting arena for the interplay between optimization theory and practice. Many more optimization applications in the airline industry will evolve in the future.

See also

- ► Integer Programming
- ► Vehicle Scheduling

References

1. Abara J (1989) Applying integer linear programming to the fleet assignment problem. Interfaces 19(4):20–28
2. Argüello MF, Bard JF, Yu G (1997) Models and methods for managing airline irregular operations aircraft routing. In: Yu G (ed) Operations Research in Airline Industry. Kluwer, Dordrecht
3. Aykin T (1994) Lagrangian relaxation based approaches to capacitated hub-and-spoke network design problem. Europ J Oper Res 79(3):501–523
4. Ball M, Roberts A (1985) A graph partitioning approach to airline crew scheduling. Transport Sci 19(2):107–126
5. Barnhart C, Boland NL, Clarke LW, Johnson EL, Nemhauser G, Shenoi RG (1998) Flight string models for aircraft fleeting and routing. Transport Sci 32(3):208–220
6. Belobaba PP (1989) Application of a probabilistic decision model to airline seat inventory control. Oper Res 37:183–197
7. Brazile RP, Swigger KM, Wyatt DL (1994) Selecting a modelling technique for the gate assignment problems: Integer programming, simulation, or expert system. Internat J Modelling and Simulation 14(1):1–5
8. Brumelle SL, McGill JI (1993) Airline seat allocation with multiple nested fare classes. Oper Res 41(1):127–137
9. Daskin MS, Panagiotopoulos ND (1989) A Lagrangian relaxation approach to assigning aircraft to routes in hub and spoke networks. Transport Sci 23(2):91–99
10. Desaulniers G, Desrosiers J, Dumas Y, Solomon MM, Soumis F (1997) Daily aircraft routing and scheduling. Managem Sci 43(6):841–855
11. Dobson G, Lederer PJ (1993) Airline scheduling and routing in a hub-and-spoke system. Transport Sci 27(3):281–297
12. Etschamaier MM, Mathaisel DFX (1985) Airline scheduling: An overview. Transport Sci 9(2):127–138
13. Feo TA, Bard JF (1989) Flight scheduling and maintenance base planning. Managem Sci 35(12):1415–1432
14. Gamache M, Soumis F, Villeneuve D, Desrosiers J (1998) The preferential bidding system at Air Canada. Transport Sci 32(3):246–255
15. Graves GW, McBride RD, Gershkoff I, Anderson D, Mahidhara D (1993) Flight crew scheduling. Managem Sci 39(6):736–745
16. Hane CA, Barnhart C, Johnson EL, Marsten RE, Nemhauser GL, Sigismondi G (1995) The fleet assignment problem: Solving a large-scale integer program. Math Program 70(2):211–232
17. Hoffman KL, Padberg M (1993) Solving airline crew scheduling problems by branch-and-cut. Managem Sci 39(6):657–680
18. Jaillet P, Song G, Yu G (1997) Airline network design and hub location problems. Location Sci 4(3):195–212
19. Jarrah AIZ, Yu G, Krishnamurthy N, Rakshit A (1993) A decision support framework for airline flight cancellations and delays. Transport Sci 27(3):266–280
20. Jorge-Calderon JD (1997) A demand model for scheduled airline services on international European routes. J Air Transport Management 3(1):23–35
21. Lavoie S, Minoux M, Odier E (1988) A new approach for crew pairing problems by column generation with an application to air transportation. Europ J Oper Res 35:45–58
22. Luo S, Yu G (1997) On the airline schedule perturbation problem caused by the ground delay program. Transport Sci 31(4):298–311
23. Navazio L, Romanin-Jacur G (1998) The multiple connections multi-airport ground holding problem: Models and algorithms. Transport Sci 32(3):268–276
24. Nicoletti B (1975) Automatic crew rostering. Transport Sci 9(1):33–48
25. Richetta O, Odoni AR (1993) Solving optimally the static ground-holding policy problem in air traffic control. Transport Sci 27(3):228–238
26. Richter H (1989) Thirty years of airline operations research. Interfaces 19(4):3–9
27. Rushmeier RA, Kontogiorgis SA (1997) Advances in the optimization of airline fleet assignment. Transport Sci 31(2):159–169
28. Stojkovic M, Soumis F, Desrosiers J (1998) The operational airline crew scheduling problem. Transport Sci 32(3):232–245
29. Stroup JS, Wollmer RD (1992) A fuel management model for the airline industry. Oper Res 40(2):229–237
30. Subramanian R, Scheff RP Jr, Quillinan JD, Wiper DS, Marsten RE (1994) Coldstart: Fleet assignment at delta air lines. Interfaces 24(1):104–120
31. Tak TC, Hersh M (1993) A model for dynamic airline seat inventory control with multiple seat bookings. Transport Sci 27(3):252–265
32. Talluri KT (1993) Swapping applications in a daily airline fleet assignment. Transport Sci 30(3):237–248

33. Thengvall BT, Bard JF, Yu G (2000) Balancing user preferences for aircraft schedule recovery during airline irregular operations. IEE Trans Oper Eng 32(3):181–193
34. Vranas PB, Bertsemas DJ, Odoni AR (1994) The multi-airport ground-holding problem in air traffic control. Oper Res 42(2):249–261
35. Wei G, Song G, Yu G (1997) Model and algorithm for crew management during airline irregular operations. J Combin Optim 1(3):80–97
36. Yan S, Tu Y (1997) Multifleet routing and multistop flight scheduling for schedule perturbation. Europ J Oper Res 103(1):155–169

Algorithmic Improvements Using a Heuristic Parameter, Reject Index for Interval Optimization

TIBOR CSENDES
University of Szeged, Szeged, Hungary

MSC2000: 65K05, 90C30

Article Outline

Keywords and Phrases

Branch-and-bound; Interval arithmetic; Optimization; Heuristic parameter

Introduction

Interval optimization methods (► interval analysis: unconstrained and constrained optimization) have the guarantee not to loose global optimizer points. To achieve this, a deterministic branch-and-bound framework is applied. Still, heuristic algorithmic improvements may increase the convergence speed while keeping the guaranteed reliability.

The indicator parameter called RejectIndex

$$pf^*(X) = \frac{f^* - \underline{F}(X)}{\overline{F}(X) - \underline{F}(X)}$$

was suggested by L.G. Casado as a measure of the closeness of the interval X to a global minimizer point [1]. It was first applied to improve the work load balance of global optimization algorithms.

A subinterval X of the search space with the minimal value of the inclusion function $\underline{F}(X)$ is usually considered as the best candidate to contain a global minimum. However, the larger the interval X, the larger the overestimation of the range $f(X)$ on X compared to $F(X)$. Therefore a box could be considered as a good candidate to contain a global minimum just because it is larger than the others. To compare subintervals of different sizes we normalize the distance between the global minimum value f^* and $\underline{F}(X)$.

The idea behind pf^* is that in general we expect the overestimation to be symmetric, i. e., the overestimation above $f(X)$ is closely equal to the overestimation below $f(X)$ for small subintervals containing a global minimizer point. Hence, for such intervals X the relative place of the global optimum value inside the $F(X)$ interval should be high, while for intervals far from global minimizer points pf^* must be small. Obviously, there are exceptions, and there exists no theoretical proof that pf^* would be a reliable indicator of nearby global minimizer points.

The value of the global minimum is not available in most cases. A generalized expression for a wider class of indicators is

$$p(\hat{f}, X) = \frac{\hat{f} - \underline{F}(X)}{\overline{F}(X) - \underline{F}(X)},$$

where the $\hat{f}$ value is a kind of approximation of the global minimum. We assume that $\hat{f} \in F(X)$, i. e., this estimation is realistic in the sense that $\hat{f}$ is within the known bounds of the objective function on the search region. According to the numerical experience collected, we need a good approximation of the f^* value to improve the efficiency of the algorithm.

Subinterval Selection

I. Among the possible applications of these indicators the most promising and straightforward is in the *subinterval selection*. The theoretical and computational properties of the interval branch-and-bound optimization has been investigated extensively [6,7,8,9]. The most important statements proved are the follow-

ing for algorithms with balanced subdivision direction selection:

1. Assume that the inclusion function of the objective function is isotone, it has the zero convergence property, and the $p(f_k,Y)$ parameters are calculated with the f_k parameters converging to $\hat{f} > f^*$, for which there exists a point $\hat{x} \in X$ with $f(\hat{x}) = \hat{f}$. Then the branch-and-bound algorithm that selects that interval Y from the working list which has the maximal $p(f_i,Z)$ value can converge to a point $\hat{x} \in X$ for which $f(\hat{x}) > f^*$, i. e., to a point which is not a global minimizer point of the given problem.
2. Assume that the inclusion function of the objective function has the zero convergence property and f_k converges to $\hat{f} < f^*$. Then the optimization branch-and-bound algorithm will produce an everywhere dense sequence of subintervals converging to each point of the search region X regardless of the objective function value.
3. Assume that the inclusion function of the objective function is isotone and has the zero convergence property. Consider the interval branch-and-bound optimization algorithm that uses the cutoff test, the monotonicity test, the ► interval Newton step, and the concavity test as accelerating devices, and that selects as the next leading interval that interval Y from the working list which has the maximal $p(f_i,Z)$ value. A necessary and sufficient condition for the convergence of this algorithm to a set of global minimizer points is that the sequence $\{f_i\}$ converges to the global minimum value f^*, and there exist at most a finite number of f_i values below f^*.
4. If our algorithm applies the interval selection rule of maximizing the $p(f^*, X) = pf^*(X)$ values for the members of the list L (i. e., if we can use the known exact global minimum value), then the algorithm converges exclusively to global minimizer points.
5. If our algorithm applies the interval selection rule of maximizing the $p(\tilde{f}, X)$ values for the members of the list L, where $\tilde{f}$ is the best available upper bound for the global minimum, *and its convergence to f^* can be ensured*, then the algorithm converges exclusively to global minimizer points.
6. Assume that for an optimization problem $\min_{x\in X} f(x)$ the inclusion function $F(X)$ of $f(x)$ is isotone and α-convergent with given positive constants α and C. Assume further that the pf^* parameter is less than 1 for all the subintervals of X. Then an arbitrary large number $N(> 0)$ of consecutive leading intervals of the basic B&B algorithm that selects the subinterval with the smallest lower bound as the next leading interval may have the following properties:
   i. None of these processed intervals contains a stationary point.
   ii. During this phase of the search the pf^* values are maximal for these intervals.
7. Assume that the inclusion function of the objective function is isotone and it has the zero convergence property. Consider the interval branch-and-bound optimization algorithm that uses the cutoff test, the monotonicity test, the interval Newton step, and the concavity test as accelerating devices and that selects as the next leading interval that interval Y from the working list which has the maximal $pf(f_k,Z)$ value.
   i. The algorithm converges exclusively to global minimizer points if

$$\underline{f}_k \le f_k < \delta(\overline{f}_k - \underline{f}_k) + \underline{f}_k$$

   holds for each iteration number k, where $0 < \delta < 1$.
   ii. The above condition is sharp in the sense that $\delta = 1$ allows convergence to not optimal points.

Here $\underline{f}_k = \min\{\underline{F}(Y^l), l = 1, \ldots, |L_k|\} \le f_k < \tilde{f}_k = \overline{f}_k$, where $|L|$ stands for the cardinality of the elements of the list L.

II. These theoretical results are in part promising (e. g., 7), in part disappointing (5 and 6). The conclusions of the detailed numerical comparisons were that if the global minimum value is known, then the use of the pf^* parameter in the described way can accelerate the interval optimization method by orders of magnitude, and this improvement is especially strong for hard problems.

In case the global minimum value is not available, then its estimation, f_k, which fulfills the conditions of 7, can be utilized with similar efficacy, and again the best results were achieved on difficult problems.

Multisection

I. The multisection technique is a way to accelerate branch-and-bound methods by subdividing the actual interval into several subintervals in a single algorithm

step. In the extreme case half of the function evaluations can be saved [5,10]. On the basis of the RejectIndex value of a given interval it is decided whether simple bisection or two higher-degree multisections are to be applied [2,11]. Two threshold values, $0 < P_1 < P_2 < 1$, are used for selecting the proper multisection type.

This algorithm improvement can also be cheated in the sense that there exist global optimization problems for which the new method will follow for an arbitrary long number of iterations an embedded interval sequence that contains no global minimizer point, or that intervals in which there is a global minimizer have misleading indicator values.

According to the numerical tests, the new multisection strategies result in a substantial decrease both in the number of function evaluations and in the memory complexity.

II. The multisection strategy can also be applied to constrained global optimization problems [11]. The feasibility degree index for constraint $g_j(x) \leq 0$ can be formulated as

$$pu_{G_j}(X) = \min\left\{\frac{-\underline{G}_j(X)}{w(G_j(X))}, 1\right\} .$$

Notice that if $pu_{G_j}(X) < 0$, then the box is certainly infeasible, and if $pu_{G_j}(X) = 1$ then X certainly satisfies the constraint. Otherwise, the box is undetermined for that constraint. For boxes that are not certainly infeasible, i. e., for which $pu_{G_j}(X) \geq 0$ for all $j = 1, \ldots, r$ holds, the total infeasibility index is given by

$$pu(X) = \prod_{j=1}^{r} pu_{G_j}(X) .$$

We must only define the index for such boxes since certainly infeasible boxes are immediately removed by the algorithm from further consideration. With this definition,

- $pu(X) = 1 \Leftrightarrow X$ is certainly feasible and
- $pu(X) \in [0, 1) \Leftrightarrow X$ is undetermined.

Using the $pu(X)$ index, we now propose the following modification of the RejectIndex for constrained problems:

$$pup(\hat{f}, X) = pu(X) \cdot p(\hat{f}, X) ,$$

where $\hat{f}$ is a parameter of this indicator, which is usually an approximation of f^*. This new index works like $p(\hat{f}, X)$ if X is certainly feasible, but if the box is undetermined, then it takes the feasibility degree of the box into account: the less feasible the box is, the lower the value of $pu(X)$ is.

A careful theoretical analysis proved that the new interval selection and multisection rules enable the branch-and-bound interval optimization algorithm to converge to a set of global optimizer points assuming we have a proper sequence of $\{f_k\}$ parameter values. The convergence properties obtained were very similar to those proven for the unconstrained case, and they give a firm basis for computational implementation.

A comprehensive numerical study on standard global optimization test problems and on facility location problems indicated [11] that the constrained version interval selection rules and, to a lesser extent, also the new adaptive multisection rules have several advantageous features that can contribute to the efficiency of the interval optimization techniques.

Heuristic Rejection

RejectIndex can also be used to improve the efficiency of interval global optimization algorithms on very hard to solve problems by applying a rejection strategy to get rid of subintervals not containing global minimizer points. This heuristic rejection technique selects those subintervals on the basis of a typical pattern of changes in the pf^* values [3,4].

The RejectIndex is not always reliable: assume that the inclusion function $F(X)$ of $f(x)$ is isotone and α-convergent. Assume further that the RejectIndex parameter pf^* is less than 1 for all the subintervals of X. Then an arbitrary large number $N(> 0)$ of consecutive leading intervals may have the following properties:

i. Neither of these processed intervals contains a stationary point, and
ii. During this phase of the search the pf^* values are maximal for these intervals as compared with the subintervals of the current working list.

Also, when a global optimization problem has a unique global minimizer point x^*, there always exists an isotone and α-convergent inclusion function $F(X)$ of $f(x)$ such that the new algorithm does not converge to x^*.

In spite of the possibility of losing the global minimum, obviously there exist such implementations that allow a safe way to use heuristic rejection. For example, the selected subintervals can be saved on a hard disk for further possible processing if necessary.

Although the above theoretical results were not encouraging, the computational tests on very hard global optimization problems were convincing: when the whole list of subintervals produced by the B&B algorithm is too large for the given computer memory, then the use of the suggested heuristic rejection technique decreases the number of working list elements without missing the global minimum. The new rejection test may also make it possible to solve hard-to-solve problems that are otherwise unsolvable with the usual techniques.

References

1. Casado LG, García I (1998) New Load Balancing Criterion for Parallel Interval Global Optimization Algorithms. In: Proceedings of the 16th IASTED International Conference on Applied Informatics, Garmisch-Partenkirchen, pp 321–323
2. Casado LG, García I, Csendes T (2000) A new multisection technique in interval methods for global optimization. Computing 65:263–269
3. Casado LG, García I, Csendes T (2001) A heuristic rejection criterion in interval global optimization algorithms. BIT 41:683–692
4. Casado LG, García I, Csendes T, Ruiz VG (2003) Heuristic Rejection in Interval Global Optimization. JOTA 118:27–43
5. Csallner AE, Csendes T, Markót MC (2000) Multisection in Interval Branch-and-Bound Methods for Global Optimization I. Theoretical Results. J Global Optim 16:371–392
6. Csendes T (2001) New subinterval selection criteria for interval global optimization. J Global Optim 19:307–327
7. Csendes T (2003) Numerical experiences with a new generalized subinterval selection criterion for interval global optimization. Reliab Comput 9:109–125
8. Csendes T (2004) Generalized subinterval selection criteria for interval global optimization. Numer Algorithms 37:93–100
9. Kreinovich V, Csendes T (2001) Theoretical Justification of a Heuristic Subbox Selection Criterion for Interval Global Optimization. CEJOR 9:255–265
10. Markót MC, Csendes T, Csallner AE (2000) Multisection in Interval Branch-and-Bound Methods for Global Optimization II. Numerical Tests. J Global Optim 16:219–228
11. Markót MC, Fernandez J, Casado LG, Csendes T (2006) New interval methods for constrained global optimization. Math Programm 106:287–318

Algorithms for Genomic Analysis

EVA K. LEE, KAPIL GUPTA
Center for Operations Research in Medicine and HealthCare,
School of Industrial and Systems Engineering,
Georgia Institute of Technology, Atlanta, USA

MSC2000: 90C27, 90C35, 90C11, 65K05, 90-08, 90-00

Article Outline

Abstract

The genome of an organism not only serves as its blueprint that holds the key for diagnosing and curing diseases, but also plays a pivotal role in obtaining a holistic view of its ancestry. Recent years have witnessed a large number of innovations in this field, as exemplified by the Human Genome Project. This chapter provides an overview of popular algorithms used in genome analysis and in particular explores two important and deeply interconnected problems: phylogenetic analysis and multiple sequence alignment. We also describe our novel graph-theoretical approach that en-

compasses a wide variety of genome sequence analysis problems within a single model.

Introduction

Genomics encompasses the study of the genome in human and other organisms. The rate of innovation in this field has been breathtaking over the last decade, especially with the completion of Human Genome Project. The purpose of this chapter is to review some well-known algorithms that facilitate genome analysis. The material is presented in a way that is interesting to both the specialists working in this area and others. Thus, this review includes a brief sketch of the algorithms to facilitate a deeper understanding of the concepts involved. The list of problems related to genomics is very extensive; hence, the scope of this chapter is restricted to the following two related important problems: (1) phylogenetic analysis and (2) multiple sequence alignment. Readers interested in algorithms used in other fields of computational biology are recommended to refer to reviews by Abbas and Holmes [1] and Blazewicz et al. [7].

Genome refers to the complete DNA sequence contained in the cell. The DNA sequence consists of the four nucleotides adenine (A), thymine (T), cytosine (C), and guanine (G). Associated with each DNA strand (sequence) is a complementary DNA strand of the same length. The strands are complementary in that each nucleotide in one strand uniquely defines an associated nucleotide in the other: A and T are always paired, and C and G are always paired. Each pairing is referred to as a base pair; and bound complementary strands make up a DNA molecule. Typically, the number of base pairs in a DNA molecule is between thousands and billions, depending on the complexity of a given organism. For example, a bacterium contains about 600,000 base pairs, while human and mouse have some three billion base pairs. Among humans, 99.9% of base pairs are the same between any two unrelated persons. But that leaves millions of single-letter differences, which provide genetic variation between people.

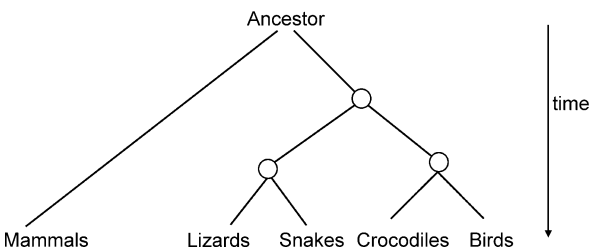
Understanding the DNA sequence is extremely important. It is considered as the blueprint for an organism's structure and function. The sequence order underlies all of life's diversity, even dictating whether an organism is human or another species such as yeast or a fruit fly. It helps in understanding the evolution of mankind, identifying genetic diseases, and creating new approaches for treating and controlling those diseases. In order to achieve these goals, research in genome analysis has progressed rapidly over the last decade.

The rest of this chapter is organized as follows. Section "Phylogenetic Analysis" discusses techniques used to infer the evolutionary history of species and Sect. "Multiple Sequence Alignment" presents the multiple sequence alignment problem and recent advances. In Sect. "Novel Graph-Theoretical Genomic Models", we describe our research effort for advancing genomic analysis through the design of a novel graph-theoretical approach for representing a wide variety of genomic sequence analysis problems within a single model. We summarize our theoretical findings, and present computational models based on two integer programming formulations. Finally, Sect. "Summary" summarizes the interdependence and the pivotal role played by the abovementioned two problems in computational biology.

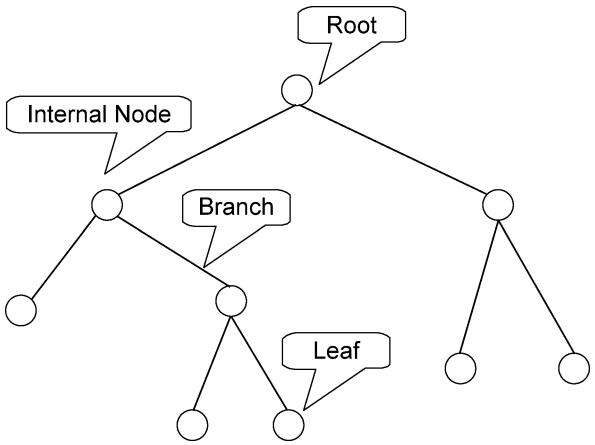
Phylogenetic Analysis

Phylogenetic analysis is a major aspect of genome research. It refers to the study of evolutionary relationships of a group of organisms. These hierarchical relationships among organisms arising through evolution are usually represented by a phylogenetic tree (Fig. 1). The idea of using trees to represent evolution dates back to Darwin. Both rooted and unrooted tree representations have been used in practice [17]. The branches of a tree represent the time of divergence and the root represents the ancestral sequence (Fig. 2).

The study of phylogenies and processes of evolution by the analysis of DNA or amino acid sequence data is



Algorithms for Genomic Analysis, Figure 1
An example of an evolutionary tree



Algorithms for Genomic Analysis, Figure 2
Tree terminology

called molecular phylogenetics. In this study, we will focus on methods that use DNA sequence data. There are two processes involved in inferring both rooted and unrooted trees. The first is estimating the branching structure or topology of the tree. The second is estimating the branch lengths for a given tree. Currently, there are wide varieties of methods available to conduct this analysis [16,19,55,79]. These available approaches can be classified into three broad groups: (1) distance methods; (2) parsimony methods; and (3) maximum likelihood methods. Below, we will discuss each of them in detail.

Methods Based on Pairwise Distance

In distance methods, an evolutionary distance d_{ij} is computed between each pair i, j of sequences, and a phylogenetic tree is constructed from these pairwise distances. There are many different ways of defining pairwise evolutionary distance used for this purpose. Most of the approaches estimate the number of nucleotide substitutions per site, but other measures have also been used [70,71]. The most popular one is the Jukes–Cantor distance [37], which defines d_{ij} as $-\frac{3}{4}\log(1-\frac{4f}{3})$, where f is the fraction of sites where nucleotides differ in the pairwise alignment [37].

There are a large number of distance methods for constructing evolutionary trees [78]. In this article, we discuss methods based on *cluster analysis* and *neighbor joining*.

Cluster Analysis: Unweighted Pair Group Method Using Arithmetic Averages The conceptually simplest and most known distance method is the unweighted pair group method using arithmetic averages (UPGMA) developed by Sokal and Michener [66]. Given a matrix of pairwise distances between each pair of sequences, it starts with assigning each sequence to its own cluster. The distances between the clusters are defined as $d_{ij} = \frac{1}{|C_i||C_j|}\sum_{p\in C_i, q\in C_j} d(p,q)$, where C_i and C_j denote sequences in clusters i and j, respectively. At each stage in the process, the least distant pair of clusters are merged to create a new cluster. This process continues until only one cluster is left. Given n sequences, the general schema of UPGMA is shown in Algorithm 1.

Algorithm 1 (UPGMA)

1. Input: Distance matrix d_{ij}, $1 \le i, j \le n$
2. **For** $i = 1$ to n **do**
3. Define singleton cluster C_i comprising of sequence i
4. Place cluster C_i as a tree leaf at height zero
5. **End for**
6. **Repeat**
7. Determine two clusters i, j such that d_{ij} is minimal.
8. Merge these two clusters to form a new cluster k having a distance from other clusters defined as the weighted average of the comprising two clusters. If C_k is the union of two clusters C_i and C_j, and if C_l is any other cluster, then $d_{kl} = \frac{d_{il}|C_i|+d_{jl}|C_j|}{|C_i|+|C_j|}$.
9. Define a node k at height $\frac{d_{ij}}{2}$ with daughter nodes i and j.
10. Until just a single cluster remains

The time and space complexity of UPGMA is $O(n^2)$, since there are $n-1$ iterations of complexity $O(n)$. A number of approaches have been developed which are motivated by UPGMA. Li [52] developed a similar approach which also makes corrections for unequal rates of evolution among lineages. Klotz and Blanken [43] presented a method where a present-day sequence serves as an ancestor in order to determine the tree regardless of the rates of evolution of the sequences involved.

Neighbor Joining Neighbor joining is another very popular algorithm based on pairwise distances [63]. This approach yields an unrooted tree and overcomes the assumption of the UPGMA method that the same rate of evolution applies to each branch.

Given a matrix of pairwise distances between each pair of sequences d_{ij}, it first defines the modified distance matrix $\bar{d}_{ij}$. This matrix is calculated by subtracting average distances to all other sequences from the d_{ij}, thus compensating for long edges. In each stage, the two nearest nodes (minimal $\bar{d}_{ij}$) of the tree are chosen and defined as neighbors in the tree. This is done recursively until all of the nodes are paired together.

Given n sequences, the general schema of neighbor joining is shown in Algorithm 2.

Algorithm 2 (Neighbor joining)

1. Input: Distance matrix d_{ij}, $\quad 1 \leq i, j \leq n$
2. **For** $i = 1$ to n
3. Assign sequence i to the set of leaf nodes of the tree (T)
4. **End for**
5. Set list of active nodes (L) = T
6. **Repeat**
7. Calculate the modified distance matrix $\bar{d}_{ij} = d_{ij} - (r_i + r_j)$, where $r_i = \frac{1}{|L|-2} \sum_{k \in L} d_{ik}$
8. Find the pair i, j in L having the minimal value of $\bar{d}_{ij}$
9. Define a new node u and set $d_{uk} = \frac{1}{2}(d_{ik} + d_{jk} - d_{ij})$, for all k in L
10. Add u to T joining nodes i, j with edges of length given by: $d_{iu} = \frac{1}{2}(d_{ij} + r_i - r_j)$, $d_{ju} = d_{ij} - d_{iu}$
11. Remove i and j from L and add u
12. **Until** only two nodes remain in L
13. Connect remaining two nodes i and j by a branch of length d_{ij}

Neighbor joining has a execution time of $O(n^2)$, like UPGMA. It has given extremely good results in practice and is computationally efficient [63,72]. Many practitioners have developed algorithms based on this approach. Gascuel [24] improved the neighbor-joining approach by using a simple first-order model of the variances and covariances of evolutionary distance estimates. Bruno et al. [10] developed a weighted neighbor joining using a likelihood-based approach. Goeffon et al. [25] investigated a local search algorithm under the maximum parsimony criterion by introducing a new subtree swapping neighborhood with an effective array-based tree representation.

Parsimony Methods

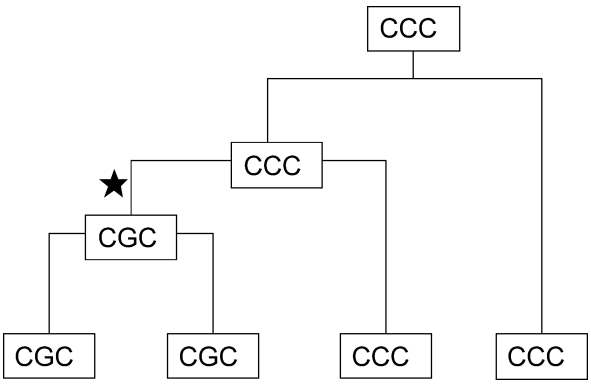
In science, notion of parsimony refers to the preference of simpler hypotheses over complicated ones. In the parsimony approach for tree building, the goal is to identify the phylogeny that requires the fewest necessary changes to explain the differences among the observed sequences. Of the existing numerical approaches for reconstructing ancestral relationships directly from sequence data, this approach is the most popular one. Unlike distance-based methods which build trees, it evaluates all possible trees and gives each a score based on the number of evolutionary changes that are needed to explain the observed sequences. The most parsimonious tree is the one that requires the fewest evolutionary changes for all sequences to derive from a common ancestor [69]. As an example, consider the trees in Fig. 3 and Fig. 4. The tree in Fig. 3 requires only one evolutionary change (marked by the star) compared with the tree in Fig. 4, which requires two changes. Thus, Fig. 3 shows the more parsimonious tree.

There are two distinct components in parsimony methods: given a labeled tree, determine the score; determine global minimum score by evaluating all possible trees, as discussed below.

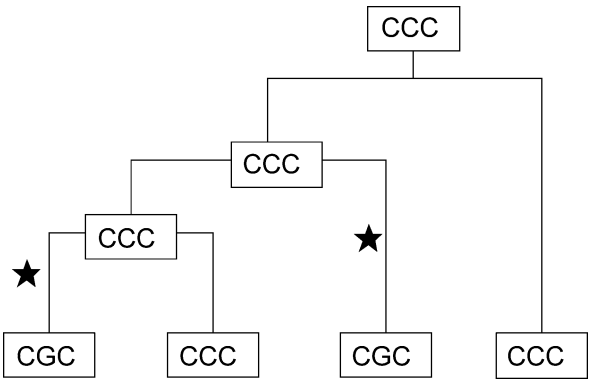
Score Computation Given a set of nucleotide sequences, parsimony methods treat each site (position) independently. The algorithm evaluates the score at each position and then sums them up over all the positions. As an example, suppose we have the following three aligned nucleotide sequences:

CCC
GGC
CGC

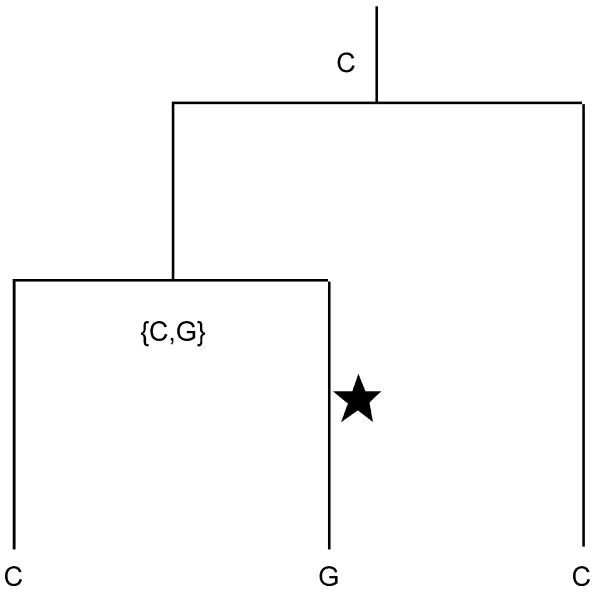
Then, for a given tree topology, we would calculate the minimal number of changes required at each of the three sites and then sum them up. Here, we investigate a traditional parsimony algorithm developed by Fitch [21], where the number of substitutions required is taken as a score. For a particular topology, this ap-



Algorithms for Genomic Analysis, Figure 3
Parsimony tree 1



Algorithms for Genomic Analysis, Figure 4
Parsimony tree 2



Algorithms for Genomic Analysis, Figure 5
The sets R_k for the first site of given three sequences

proach starts by placing nucleotides at the leaves and traverses toward the root of the tree. At each node, the nucleotides common to all of the descendant nodes are placed. If this set is empty then the union set is placed at this node. This continues until the root of the tree is reached. The number of union sets { equals} the number of substitutions required.

The general scheme for every position is shown in Algorithm 3.

Algorithm 3 (Parsimony: score computation)

1. Each leaf l is labeled with set R_l having observed nucleotide at that position.
2. Score $S = 0$
3. **For all** internal nodes k with children i and j having labels R_i and R_j **do**
4. $R_k = R_i \bigcap R_j$
5. **if** $R_k = \emptyset$ **then**
6. $R_k = R_i \bigcup R_j$
7. $S = S + 1$
8. **end if**
9. **End for**
10. Minimal score $= S$

Figure 5 shows the set R_k obtained by Algorithm 3. The computation is done for the first site of the three sequences shown above. The minimal score given by the algorithm is 1.

A wide variety of approaches have been developed by modifying Fitch's algorithm [68]. Sankoff and Cedergren [64] presented a generalized parsimony method which does not just count the number of substitutions, but also assigns a weighted cost for each substitution.

Ronquist [62] improved the computational time by including strategies for rapid evaluation of tree lengths and increasing the exhaustiveness of branch swapping while searching topologies.

Search of Possible Tree Topologies The number of possible tree topologies dramatically increases with the number of sequences. Consequently, in practice usu-

ally only a subset of them are examined using efficient search strategies. The most commonly used strategy is branch and bound methods to select branching patterns [60]. For large-scale problems, heuristic methods are typically used [69]. These exact and heuristic tree search strategies are implemented in various programs like PHYLIP (phylogeny inference package) and MEGA (molecular evolutionary genetic analysis) [20,47].

Maximum Likelihood Methods

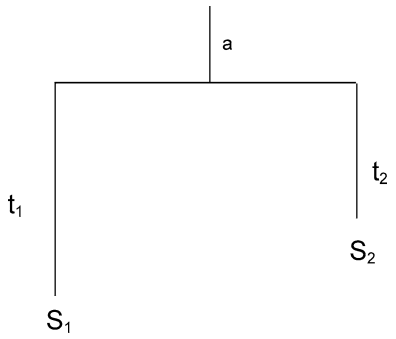
The method of maximum likelihood is one of the most popular statistical tools used in practice. In molecular phylogenetics, maximum likelihood methods find the tree that has the highest probability of generating observed sequences, given an explicit model of evolution. The method was first introduced by Felenstein [18]. We discuss herein both the evolution models and the calculation of tree likelihood.

Model of Evolution A model of evolution refers to various events like mutation, which changes one sequence to another over a period of time. It is required to determine the probability of a sequence S_2 arising from an ancestral sequence S_1 over a period of time t. Various sophisticated models of evolution have been suggested, but simple models like the Jukes–Cantor model are preferred in maximum likelihood methods.

The Jukes–Cantor [37] model assumes that all nucleotides (A, C, T, G) undergo mutation with equal probability, and change to all of the other three possible nucleotides with the same probability. If the mutation rate is 3α per unit time per site, the mutation matrix P_{ij} (probability that nucleotide i changes to nucleotide j in unit time) takes the form

$$\begin{pmatrix} 1-3\alpha & \alpha & \alpha & \alpha \\ \alpha & 1-3\alpha & \alpha & \alpha \\ \alpha & \alpha & 1-3\alpha & \alpha \\ \alpha & \alpha & \alpha & 1-3\alpha \end{pmatrix}.$$

The above matrix is integrated to evaluate mutation rates over time t and is then used to calculate $P(nt_2|nt_1, t)$, defined as the probability of nucleotide nt_1 being substituted by nucleotide nt_2 over time t.



Algorithms for Genomic Analysis, Figure 6
A simple tree

Various other evolution models like the Kimura model have also been mentioned in the literature [9,42].

Likelihood of a Tree The likelihood of a tree is calculated as the probability of observing a set of sequences given the tree.

$$L(\text{tree}) = \text{probability}[\text{sequences}|\text{tree}]$$

We begin with the simple case of two sequences S^1 and S^2 of length n having a common ancestor **a** as shown in Fig. 6. It is assumed that all different sites (positions) evolve independently, and thus the total likelihood is calculated as the product of the likelihood of all sites [15]. Here, the likelihood of each site is obtained using substitution probabilities based on an evolution model.

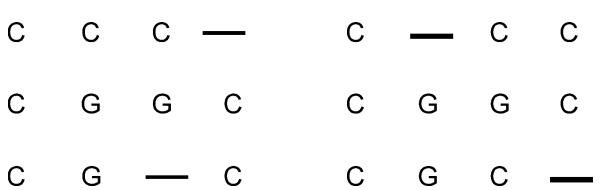
Given q_a is the equilibrium distribution of nucleotide a, the likelihood for the simple tree in Fig. 6 is calculated as $L(\text{tree}) = P(S^1, S^2) = \prod_{i=1}^{n} P(S_i^1, S_i^2)$, where $P(S_i^1, S_i^2) = \sum_a q_a P(S_i^1|a) P(S_i^2|a)$. To generalize this approach for m sequences, it is assumed that diverged sequences evolve independently after diverging. Hence, the likelihood for every node in a tree depends only on its immediate ancestral node and a recursive procedure is used to evaluate the likelihood of the tree. The conditional likelihood $L_{k,a}$ is defined as the likelihood of the subtree rooted at node k, given that the nucleotide at node k is a. The general schema for every site is shown in Algorithm 4. The likelihood is then maximized over all possible tree topologies and branch lengths.

Algorithm 4 (Likelihood: computation at given site)

1. **For all** leaf l **do**
2. **if** leaf has nucleotide a at that site **then**
3. $L_{l,a} = 1$
4. **else**
5. $L_{l,a} = 0$
6. **end if**
7. **End for**
8. **For all** internal nodes k with children i and j
9. define the conditional likelihood
$L_{k,a} = \sum_{b,c}[P(b|a)L_{i,b}][P(c|a)L_{j,c}]$
10. **End for**
11. Likelihood at given site = $\sum_a q_a\, L_{root,a}$

Recent Improvements The maximum likelihood approach has received great attention owing to the existence of powerful statistical tools. It has been made more sophisticated using advance tree search algorithms, sequence evolution models, and statistical approaches. Yang [80] extended it to the case where the rate of nucleotide substitutions differ over sites. Huelsenbeck and Crandall [34] incorporated the improvements in substitution models. Piontkivska [59] evaluated the use of various substitution models in the maximum likelihood approach and inferred that simple models are comparable in terms of both efficiency and reliability with complex models.

The enormously large number of possible tree topologies, especially while working with a large number of sequences, makes this approach computationally intensive [72]. It has been proved that reconstructing the maximum likelihood tree is nondeterministic polynomial time hard (NP) hard even for certain approximations [14]. In order to reduce computational time, Guindon and Gascuel [31] developed a simple hill-climbing algorithm based on the maximum-likelihood principle that adjusts tree topology and branch lengths simultaneously. Recently, parallel computation has been used to address huge computational requirement. Stamatakis et al. [67] have used OpenMP–parallelization for symmetric multiprocessing machines and Keane et al. [39] developed a distributed platform for phylogeny reconstruction by maximum likelihood.



Algorithms for Genomic Analysis, Figure 7
Two possible alignments for given three sequences

Multiple Sequence Alignment

Multiple sequence alignment is arguably among the most studied and difficult problems in computational biology. It is a vital tool because it compactly represents conserved or variable features among the family members. Alignment also allows character-based analysis compared to distance-based analysis and thus helps to elucidate evolutionary relationships better. Consequently, it plays a pivotal role in a wide range of sequence analysis problems like identifying conserved motifs among given sequences, predicting secondary and tertiary structures of protein sequences, and molecular phylogenetic analysis. It is also used for sequence comparison to find the similarity of a new sequence with pre-existing ones. This helps in gathering information about the function and structure of newly found sequences from existing ones in databases like GenBank in the USA and EMBL in Europe.

The multiple sequence alignment problem can be stated formally as follows. Let $\sum$ be the alphabet and let $\hat{\sum} = \sum \bigcup \{-\}$, where "–" is a symbol to represent "gaps" in sequences. For DNA sequences, alphabet $\hat{\sum} = \{\text{A,T,C,G,}-\}$.

An alignment for N sequences $S_1, \ldots, S_N$ is given by a set $\hat{S} = \{S_1, \ldots, S_N\}$ over the alphabet $\hat{\sum}$ which satisfy the following two properties: (1) the strings in $\hat{S}$ are of the same length; (2) S_i can be obtained from $\hat{S}_i$ by removing the gaps. Thus, an alignment in which each string $\hat{S}_i$ has length K can be interpreted as an alignment matrix of N rows and K columns, where row i corresponds to sequence S_i. Alphabets that are placed into the same column of the alignment matrix are said to be aligned with each other.

Figure 7 shows two possible alignments for given three sequences: S_1 = CCC, S_2 = CGGC, and S_3 = CGC.

For two sequences, the optimal multiple sequence alignment is easily obtained using dynamic program-

ming (Needleman–Wunsch algorithm). Unfortunately, the problem becomes much harder for more than two sequences, and the optimal solution can be found only for a limited number of sequences of moderate length (approximately 100) [8]. Researchers have tried to solve it by generalizing the dynamic programming approach to a multidimensional space. However, this approach has huge time and memory requirements and thus cannot be used in practice even for small problems of five sequences of length 100 each. This algorithm has been improved by identifying the portion of hyperspace which does not contribute to the solution and excluding it from the computation [11]. But even this approach of Carrillo and Lipman implemented in the multiple sequence alignment program can only align up to ten sequences [53]. Although, Gupta et al. [32] improved the space and time usage of this approach, it cannot align large data sets. To reduce the huge time and memory expenses, a wide variety of heuristic approaches for multiple sequence alignment have been developed [56].

There are two components for finding the multiple sequence alignment: (1) searching over all the possible multiple alignments; (2) scoring each of them to find the best one.

The problem becomes more complex for remotely related homologous sequences, i. e., sequences which are not derived from a common ancestor [28]. Numerous approaches have been proposed, but the quest for an approach which is accurate and fast is continuing. It must be remembered that even the choice of sequences and calculating the score of alignment is a nontrivial task and is an active research field in itself.

Scoring Alignment

There is no unanimous way of characterizing an alignment as the correct one and the strategy depends on the biological context. Different alignments are possible and we never know for sure which alignment is correct. Thus, one scores every alignment according to an appropriate objective function and alignments with higher scores are deemed to be better. A typical alignment scoring scheme consists of the following steps.

Independent Columns The score of alignment is calculated in terms of columns of alignments. The individual columns are assumed to be independent and thus the total score of an alignment is a simple summation over column scores. Thus, the score for an alignment $\text{score}(A) = \sum_j \text{score}(A_j)$, where A_j is column j of the multiple alignment A. Now, the score for every column j is calculated as the "sum-of-pairs" function using the scoring matrices described below. The sum-of-pairs score for column A_j is obtained as $\text{score}(A_j) = \sum_{k<l} \text{score}(A_j^k, A_j^l)$, where A_j^k and A_j^l are nucleotides in column j of the alignment corresponding to sequences k and l, respectively. If the gap costs are linear, score(nucleotide, –) and score(–, nucleotide) will be the insertion cost. But, this approach would not differentiate between opening a gap and extending it. So, affine gap penalties are often used where gap opening and extension penalty are treated as two different parameters. The correct value of both of these parameters is a major concern since their values can be set only empirically [75]. Also most schemes used in practice score columns as the weighted sum of pairwise substitutions instead of just addition as described before. The weights are decided in accordance with the amount of independent information each sequence possesses [4].

Both the assumption of treating every column independently and using the sum-of-pairs score for the column have limitations. The problem increases as the number of sequences increases.

Scoring Matrices Any alignment can be obtained by performing three evolution operations: insertion, deletion, and substitution. It is assumed that all the different operations occur independently and thus the complete score is evaluated as the sum of scores from every operation. Insertion and deletion scores are calculated as either linear or affine gap penalty. Substitutions scores are stored as a substitution score matrix, which contains the score for every pair of nucleotides. Thus, these scores S(A,B) can be treated as the score of aligning nucleotide A with nucleotide B.

These substitution score matrices can be obtained in various ways. One could adopt an ad hoc approach of setting up a score matrix which produces good alignments for a given set of sequences. The second approach would be more fundamental and look into the physical and chemical properties of nucleotides. If two nucleotides have similar properties, they would be more likely to be substituted by one another. The third and

the most prominent one is a statistical approach where the maximum likelihood principle is used in conjunction with probabilistic models of evolution [3].

Alignment Approaches

The number of different approaches for the multiple sequence alignment problem has steadily increased over the last decade and thus being exhaustive will not be possible. In this chapter, we will emphasize the most widely used class of algorithms and the new emerging and most promising approaches:

1. Progressive alignment algorithms: The most widely used type of algorithm based on using pairwise alignment information of input sequences. It assumes that input sequences are phylogenetically related, and uses these relationships to guide the alignment [13].
2. Graph-based algorithms: A new trend where graph-based models are used to approach this problem.
3. Iterative alignment algorithms: Typically an alignment is produced and is then refined through a series of iterations until no more improvement can be made.

Progressive Algorithms

Progressive alignment constitutes one of the simplest and most effective ways for multiple alignment. This strategy was introduced by various researchers, like Waterman and Perlwitz [77]. Among all the progressive algorithms, ClustalW is the most famous one. It is a noniterative, deterministic algorithm that attempts to optimize the weighted sums-of-pairs with affine gap penalties [73].

The typical progressive algorithm scheme is as follows:

- Compute the distance between all pairs of given sequences by aligning them. The distances represent the divergence of each pair of sequences. These distances could be calculated by fast approximation methods or by slower but more precise methods like complete dynamic programming. Since for given N sequences $\frac{N(N-1)}{2}$ pairwise scores have to be calculated and the scores are used just for construction of a guide tree and not the alignment itself, it is desirable to use approximation methods like k tuple matches.
- Find a guide tree from the distance matrix. This is typically achieved using the clustering algorithms discussed in the construction of an evolutionary tree. Once again, since the aim is to get the alignment and not the tree itself, approximation methods are used to construct the evolution trees.
- Align sequences progressively according to the branching order in the guide tree. The basic idea is to start from the leaves of the guide tree and move toward its root and to use a series of pairwise alignments to align larger and larger groups of sequences. Some algorithms have only a single growing alignment to which every remaining sequence is aligned, whereas other approaches align a subgroup of sequences and then merge the alignments.

There are three main shortcomings of the progressive algorithms.

1. There does not exist an undisputable "best" way of ordering the given sequences.
2. Once a sequence has been aligned, that alignment will not be modified even if it conflicts with sequences added later in the process. Hence, the order in which sequences are added becomes crucial, and since there is no undisputed best way to order the sequences, this approach returns suboptimal solutions.
3. For a given set of n sequences, $\binom{n}{2}$ pairwise alignments are generated; but while computing the final multiple alignment, most of these algorithms use fewer than n pairwise alignments. Thus, the resulting multiple alignment agrees with only a small amount of information available in the data.

Therefore, there is a growing need for an algorithm to align extremely divergent sequences whose pairwise alignments are likely to be incorrect. In order to address all these issues, some techniques have been developed; while they are innovative, it is understandable that they have their own assumptions and drawbacks.

Graph-Based Algorithms

Over the last few years, the field of genomics has undergone evolutionary changes with a rapid increase in new solution strategies. The use of graph-based models is easily seen as one of the most emerging and far-reaching trends. Just and Vedova [38] used a relation between the facility location problem and sequence

alignment to prove the NP-hardness of multiple sequence alignment. In this section, we review the most prominent integer programming approaches for finding multiple sequence alignment.

Maximum-Weight Trace Kececioglu et al. [40] used a solution of the maximum trace problem to construct alignment. The algorithm starts by calculating all pairwise alignments and using them to find a trace. To achieve this, given n sequences, an input alignment graph $G = (V, E)$ is constructed. It is an n-partite graph whose vertex set V represents the characters of the given sequences and whos edge set E represents the pairs of characters matched in the pairwise alignments. The subset of matching in E realized by an alignment is called a trace.

Alignment graph $G = (V, E)$ is extended to a mixed graph $G' = (V, E, A)$ by adding arc set A which connects the characters of every sequence to the next character in the same sequence. The objective of the algorithm is to find the maximum weight trace by finding cycles termed as "critical mixed cycles" in graph G' such that they satisfy sequence alignment properties [61].

The integer programming model for this problem is formulated as

$$\text{Maximize} \sum_{e \in E} w_e x_e \tag{1}$$

$$\text{subject to} \sum_{e \in P \cap E} x_e = |E \cap P| - 1 \; \forall \text{ critical mixed cycles } P \text{ in } G' \,, \; x_e \in \{0, 1\} \text{ for all } e \in E \,. \tag{2}$$

An implementation of a branch-and-cut algorithm is used to solve the above problem. Various valid inequalities for the polytope are added as cuts, some of which are facet-defining. The algorithm is capable of giving an exact solution under the sum-of-pairs objective function with linear gap costs. Kececioglu et al. [40] have made a significant contribution by introducing a polyhedral approach capable of obtaining exact solutions for a subclass of multiple sequence alignment. However, this method has its own drawbacks like not being able to capture the order of insertions and deletions between two matchings and affine gap costs. Recently, Althaus et al. [2] proposed a general model using this approach in which arbitrary gap costs are allowed.

Minimum-Spanning Tree and Traveling Salesman Problem Shyu et al. [65] explored the use of minimum spanning trees to determine the order of sequences. The idea of the approach is to preserve the most informative distances among the set of given sequences. The criterion used is meaningful and capable of working better than the traditional criteria like those in sum-of-pairs. The algorithm itself is very efficient for practical usage, and can be easily implemented. However, it fails to address the issue of using all the information in pairwise alignments, since it only uses the score and not the pairwise alignments themselves. Moreover, this approach has all the drawbacks of the progressive strategy.

A similar approach was also developed by Korostensky and Gonnet [44] using the traveling salesman problem. In this technique, a circular sum measure is used instead of a sum-of-pairs score. The cities in the traveling salesman problem correspond to the sequences and the scores of pairwise alignment are taken as the distances. The problem is to find the longest tour where each sequence is visited exactly once [45].

Eulerian Path Approach Zhang and Waterman [81] proposed a new approach motivated by the Eulerian method for fragment assembly in DNA sequencing. In their work, a consensus sequence is found and later pairwise alignments are obtained between each input sequence and consensus sequence. Finally, multiple sequence alignment is obtained according to these pairwise alignments. The most significant advantage of this method is the linear time and memory cost for finding the consensus sequence. And, if the consensus sequence is the one closest to all given sequences, good quality alignment can be obtained in a reasonable amount of time. Once again, this approach suffers from the prominent drawback of the progressive strategy and issues in graph formation while finding the consensus sequence.

Iterative Algorithms

The main shortcoming of the progressive strategy is the failure to remove errors in the alignment, which are introduced early. The iterative algorithms are developed precisely to overcome this flaw. They are based on the idea of reconsidering and realigning previously aligned

sequences with the goal of improving the overall alignment score. Each modification step is an iteration to improve the quality of the alignment.

These available approaches can be classified into two broad categories: probabilistic iterative algorithms, and deterministic iterative algorithms. We will briefly discuss them below.

Probabilistic Algorithms We will discuss both the traditional probabilistic optimization approaches like the genetic algorithm and relatively recent approaches based on a Bayesian idea.

- *Simulated annealing and genetic algorithm*. Simulated annealing and the genetic algorithm are very popular stochastic methods for solving complex optimization problems. While they are often viewed as separate and competing paradigms, both of them are iterative algorithms which search for new solutions "near" to already known good solutions. The fundamental difference between simulated annealing and the genetic algorithm is that simulated annealing performs a local move only on one solution to create a new solution, whereas the genetic algorithm also creates solutions by combining information from two different solutions. The performance of simulated annealing and the genetic algorithm varies with the problem and representation used.
  The algorithms starts with an initial alignment and the alignment score is taken to be the objective function [57]. Various operations like mutation, insertion, and substitution constitute the local move which is used to a get new solution from existing ones. Flexibility in the scoring systems and the ability to correct for errors introduced during the early phase makes these approaches desirable [41].
- *Hidden Markov model and Gibbs sampler*. The hidden Markov model and the Gibbs sampler are relatively recent approaches which view multiple sequence alignment in a statistical context. Both of them use the central Bayesian idea of simultaneously maximizing the data and the model. The Gibbs sampler find motifs using local alignment techniques [49]. It is essentially similar to the hidden Markov model with no insert and delete states.
  The hidden Markov model is a statistical model based on the Markov process, which has gained importance in various fields related to pattern recognition. It determines the hidden parameters of the system on the basis of the observable parameters of the model. For multiple sequence alignment, the hidden Markov model consists of three types of states: match states, insert states, and delete states [46]. Each state has its own emission probability of nucleotides and transition probability to other states. The standard expectation-maximization algorithm or gradient descent algorithms are used to train the model and evaluate the parameters.
  Although the hidden Markov model has been successfully used in other areas, it faces a lot of challenges. There need to be some minimum number of sequences (approximately 50) required to train the model and the hidden Markov model can be easily trapped in local optima like other hill-climbing approaches [35].

Deterministic Algorithms A deterministic iterative algorithm starts with an initial alignment and then attempts to improve it. This helps in overcoming the drawback of a progressive alignment strategy where partial alignments are "frozen" [6]. A typical scheme is as follows:

- Given N sequences $S_1, S_2, \ldots, S_N$, find alignment A.
- Remove sequence S_1 from alignment A and realign it to the profile of other aligned sequences $S_2, \ldots, S_N$ to get new alignment A'.
- Calculate the score of the new alignment A' and if it is better replace A by A'.
- Remove sequence S_2 from A' and realign it. Continue this procedure for $S_3, \ldots, S_N$.
- Repeat the realignment steps until the alignment score converges or the number of iterations reaches the user-specified limit.

Many iteration strategies which enable very accurate alignments have been developed [76]. The aim is to reduce the greedy nature of the algorithm and avoid getting trapped in a local optimum. One approach is to remove and realign every sequence to the rest in each iteration. Then, the alignment with the best score is taken to be the input for the next iteration. The other famous approach is to randomly split a set of sequences into two sets, which are then realigned.

Some researchers have incorporated the iterative strategy in the progressive alignment procedure itself. For instance, a double iteration loop has been

used to make the alignment, guide tree, and sequence weights mutually consistent [27]. Recently, Chakrabarti et al. [12] developed an approach which provides a fast and accurate method for refining existing block-based alignments.

Novel Graph-Theoretical Genomic Models

In this section, we present our research effort for a novel graph-theoretical approach for representing a wide variety of genomic sequence analysis problems within a single model [50]. The model allows incorporation of the operations "insertion," "deletion," and "substitution," and various parameters such as relative distances and weights. Conceptually, we refer the problem as the *minimum weight common mutated sequence* (MWCMS) problem. The MWCMS model has many applications, including the multiple sequence alignment problem, phylogenetic analysis, the DNA sequencing problem, and the sequence comparison problem, which encompass a core set of very difficult problems in computational biology. Thus, the model presented in this section lays out a mathematical modeling framework that allows one to investigate theoretical and computational issues, and to forge new advances for these distinct, but related problems.

DNA sequencing refers to determining the exact order of nucleotide sequences in a segment of DNA. This was the greatest technical challenge in the Human Genome Project. Achieving this goal has helped reveal the estimated 30,000 human genes that are the basic physical and functional units of heredity. The resulting DNA sequence maps are being used by scientists to explore human biology and other complex phenomena.

The structure of a DNA strand (sequence) is determined by experimentation. Typically, short sequences are determined to be in the strand, and the short sequences identified are then "connected" to form a long sequence. Recent advances attempting to identify DNA strand structure involve sequencing by hybridization [5,36]. Sequencing by hybridization is the process where every possible sequence of length n (4^n possibilities) is compared with a full DNA strand. Practical values for n are 8–12. Each short string either binds or does not bind to the full strand. Biologists can thus determine exactly which short strings are contained in the DNA strand and which are not.

However, the experiment does not identify the exact location of each short string in the full strand. Hence, an important issue involves how these short strings are connected together to form the complete strand. This problem can be viewed as a shortest common superstring problem and has been studied extensively [22,23,54]. Unfortunately, errors may arise during sequencing experiments. Three types of errors are deletions (a letter appears in an input string that should not be in the final sequence), insertions (a letter is missing from an input string), and substitutions (a letter in an input string should be substituted with another letter). *The MWCMS problem can be used to model and solve this shortest common superstring problem while addressing the issue of possible errors.*

Sequence comparison is one of the most crucial problems faced by researchers in the area of bioinformatics. The sequence patterns are conserved during evolution. Given a new sequence, it will be of interest to understand how much similarity it has with pre-existing sequences. Significant similarity between two sequences implies similarities in their structures and/or functions. There are lots of DNA databases containing DNA sequences and their functions. The major ones are GenBank in the USA and the EMBL data library in Europe. If one finds a new sequence similar to existing ones in these databases, one can transfer information about the function and structure [78]. Hence, an algorithm for sequence comparison which is efficient for a large number of sequences will play a pivotal role in rapid sequence analysis. The MWCMS problem can be used to address this issue.

Definitions

Our motivation for first defining the problem arose from the desire to help quantify the concept of the "best" representative sequence in the evolutionary distance problem. The evolutionary distance problem involves finding the DNA sequence of the most likely ancestor associated with a given set of DNA sequences from distinct but similar organisms. In other words, find the DNA strand that best represents a possible ancestor, if each of the organisms evolved from the same ancestor. Changes that contribute to differences between the given sequences and the ancestor are referred to as insertions, deletions, and substitutions.

These operations account for both evolutionary mutations and experimental errors in sequencing. Mathematically, given two sequences S and B, let ord(S, B) be an ordered collection of insertions, deletions, and substitutions to convert sequence S to sequence B. (For any two sequences S and B, there are an infinite number of collections ord(S, B).) Let $w(\text{ord}(S, B))$ be the weight of the conversion from S to B, where the weight is the sum of an expression involving values η, δ, and $\psi \in \Re^+$ which represent the weights associated with a single insertion, deletion, and substitution, respectively. Let $\text{ord}^*(S, B)$ be such that $w(\text{ord}^*(S, B)) \leq w(\text{ord}(S, B))$ for all ord(S, B). Define $d(S, B) = w(\text{ord}^*(S, B))$. Formally, the MWCMS problem can be stated as follows: Given positive weights η, δ, and ψ corresponding to a single insertion, deletion, and substitution respectively, a positive threshold κ, and finite sequences $S_1, \ldots, S_m$ from a finite alphabet, does there exist a sequence B such that $\sum_{i=1}^{m} d(S_i, B) \leq \kappa$?

We have defined the MWCMS problem—which incorporates the notions of insertion, deletion, and substitution—to help quantify the concept of the "best" representative sequence in the evolutionary distance problem. We now define precisely the operations of *insertion*, *deletion*, and *substitution*. Let $S = \{s_1, \ldots, s_n\}$ be a finite sequence of letters from a finite alphabet:

1. An *insertion* of an element x in position i of the sequence S is characterized by the addition of x between elements s_i and s_{i+1}. An insertion carries an associated penalty cost of η.
2. A *deletion* of an element in position i of S amounts to deleting s_i from the sequence S. The penalty for deletion is represented by δ.
3. A *substitution* of an element in position i of S amounts to replacing s_i with another letter from the alphabet. The penalty for substitution is represented by ψ.

We remark that a penalty cost for an operation could, more generally, depend on the position where the operation is performed and/or the element to be inserted/deleted/substituted.

Let $S_1 = \{s_{11}, \ldots, s_{1m}\}$ and $S_2 = \{s_{21}, \ldots, s_{2n}\}$ be two finite sequences of letters from a finite alphabet $\sum$. We say that the *relative distance* between elements s_{1i} and s_{2j} is k if $|i - j| = k$. We define a *k-restrictive* bipartite graph as a graph $G_k = (V_1, V_2, E_k)$ such that the nodes in V_1 and V_2 correspond, respectively, to each of the elements from the first and the second sequences. We assume the nodes in V_i are ordered in the same order as they appear in the sequence S_i. There is an edge between nodes $u \in V_1$ and $v \in V_2$ if u and v are identical (i. e., the same letter of the alphabet $\sum$) and if the relative distance between these two elements is less than or equal to k. The problem of identifying the "greatest similarity" between these two sequences can then be approached as the problem of finding a maximum cardinality matching between the associated node sets, subject to restrictions on which matchings are allowed. In particular, one must take into consideration the ordering of nodes so as to preserve the relative occurrence of the elements in the matching. In addition, matchings that have edge crossings must be prevented. When $k = \max\{|S_1|, |S_2|\} - 1$, we denote the graph by $G = (V_1, V_2, E)$, and the problem is equivalent to the well-studied longest common subsequence problem for two sequences, which is polynomial time solvable [23].

Construction of a Conflict Graph from Paths of Multiple Sequences

Let S_i, $i = 1, \ldots, m$, be a collection of finite sequences, each of length n, over a common alphabet $\sum$. Let $G_k = (V_1, \ldots V_m, E_1, E_2, \ldots, E_{m-1})$ be the *k-restrictive multilayer* graph in which each element in S_i forms a distinct node in V_i. Assume the nodes in V_i are ordered in the same order as they appear in the sequence S_i. E_i denotes the set of edges between nodes in V_i and V_{i+1}. There is an edge between nodes $u \in V_i$ and $v \in V_{i+1}$ if and only if u and v are the same letter in the alphabet $\sum$, and the relative distance between them is less than or equal to k. The multiple sequence comparison problem involves finding the longest common subsequence within the sequences S_i, $i = 1, \ldots, m$. We call a path $P = p_1, p_2, \ldots, p_m$ a *complete* path in G_k if $p_i \in V_i$ and $p_i p_{i+1} \in E_i$. Two complete paths are said to be *parallel* if their node sets are disjoint and the edges do not cross. Hence, a set of parallel complete paths in G_k corresponds to a feasible solution to longest common subsequence problem on the collection of sequences $S_i, i = 1, \ldots, m$. We say that two complete paths P_1 and P_2 *cross* if they are not parallel. We remark that the longest common subsequence problem with the number of sequences bounded,is polynomial time

solvable using dynamic programming [23]. In general, the problem remains NP-complete.

We can incorporate insertions by generating new paths which include inserted nodes on various layers. The weight for such a new path will be affected by the total number of insertions in the path. In particular, if L is a common subsequence for S_i and $|S_i| = n$ for all $i = 1, \ldots, m$, then the total number of unmatched elements remaining will be $m(n - |L|)$. These elements can be deleted completely, or for a given unmatched element, one can increase the size of L by 1 by appropriately inserting this element into various sequences. By doing so, one decreases the number of unmatched elements. Let l be the number of insertions needed to generate a new complete path. Then the number of unmatched elements will decrease by $m - l$. If we assume that at the end of the sequencing process all unmatched elements will be deleted, then the penalty for generating this new complete path will be given by $l\eta - (m - l)\delta$.

We next define the concept of a conflict graph relative to the complete paths in G_k.

Definition 1 Let $\mathcal{P} = \{P_1, \ldots, P_s\}$ be a finite collection of complete paths in G_k. The *conflict graph* $C_{\mathcal{P}} = (V_{\mathcal{P}}, E_{\mathcal{P}})$ associated with $\mathcal{P}$ is constructed as follows:

- $V_{\mathcal{P}} = \{P_1, \ldots, P_s\}$;
- there is an edge between two nodes P_i and P_j in $V_{\mathcal{P}}$ if and only if P_i and P_j cross each other.

This definition applies to any multilayer graph in general. Note that any stable set of nodes in $C_{\mathcal{P}}$ corresponds to a set of parallel complete paths for G_k, and thereby to a feasible solution to the longest common subsequence problem on the collection of sequences S_i, $i = 1, \ldots, m$.

We remark that when $m = 2$, the resulting conflict graph is weakly triangulated, and thus is perfect. For $m > 2$, the conflict graph can contain an antihole of size 6. However, these complete paths can be viewed as continuous functions on the interval from 0 to 1; thus, by construction, $C_{\mathcal{P}}$ is perfect [26].

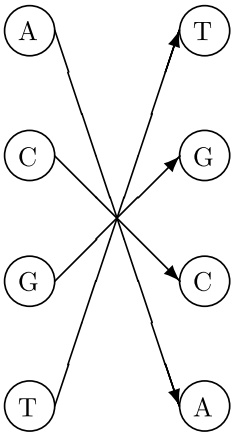
Complexity Theory

Recall that the notation $\text{ord}(S, B)$, $w(\text{ord}(S, B))$, $\text{ord}^*(S, B)$, and the formal definition of the MWCMS problem were given in Sect. "Definitions". As an optimization problem, the MWCMS problem can be stated as follows. Given a set of input sequences, the MWCMS problem seeks to mutate every input sequence to the same a priori unknown sequence using the operations of insertion, deletion, and substitution; weights are assigned for each operation, and the total weight associated with all mutations is to be minimized. Levenshtein [51] first considered a special case of this problem by changing a single input sequence to another sequence using insertions, deletions, and substitutions. Our study involves changing multiple input sequences to arrive at an a priori unknown common sequence.

Given positive weights η, δ, and ψ corresponding, respectively, to insertions, deletions, and substitutions and any two sequences S and B, clearly any $\text{ord}^*(S, B)$ will never contain more than $|B|$ insertions or substitutions. Proving that the MWCMS is in NP is not obvious. While one can transform the MWCMS to special applications (as described at beginning of Sect. "Novel Graph-Theoretical Genomic Models") to conclude that it is in NP, here we prove it directly for the general case. One needs to be able to evaluate $d(S, B)$ in polynomial time for any two sequences S and B. We next construct a graph that can be used to establish the existence of a polynomial-time algorithm for obtaining $d(S, B)$. The constructs and arguments used here typify those used to establish many of the results presented in this chapter. It is noteworthy that the notions of both conflict graph and perfect graph come into play.

Let $\sum$ be a finite alphabet, and define $\sum$-cross to be a directed bipartite graph consisting of $|\sum|$ vertices in each bipartition such that each vertex in the bipartition represents a distinct element in $\sum$. There is an arc between two vertices if the vertices correspond to the same element in $\sum$, and the geometric layout is rigidly constructed so that every arc crosses every other arc. This graph will be used as a "supernode" for insertion and substitution operations in our model. Figure 8 shows an example for $\sum$-cross when $\sum = \{\text{A,C,G,T}\}$.

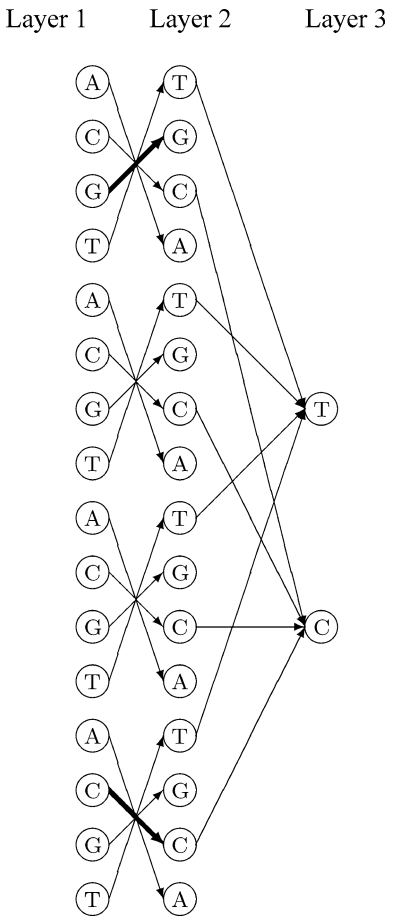
We now construct a *three-layer supergraph*, G_L, using the sequences S and B along with the $\sum$-cross graphs. Layers 1 and 2 consist of exactly $|B|(|S| + 1) + |S|$ $\sum$-crosses. The first $|B|$ $\sum$-crosses represent potential insertions before the first letter in S. The next $\sum$-cross represents either the first letter of S or a substitution of this letter. The next $|B|$ $\sum$-crosses represent potential insertions between the first and second letters of S. And this is followed by a $\sum$-cross rep-



Algorithms for Genomic Analysis, Figure 8
An example of $\sum$-cross when $\sum$ = {A, C, G, T}

resenting either the second letter of S or a substitution of this letter. This continues for each letter in S with the final $|B|$ $\sum$-crosses representing up to $|B|$ insertions after the last letter in S. Each $\sum$-cross is called either an *insertion supernode* or a *substitution supernode*, according to what it represents. The weight of all of the arcs in an insertion supernode is η. An arc in a substitution supernode has weight $-\delta$ if the arc represents the original letter in the sequences, or $\psi - \delta$ if the arc represents a substitution of the original letter. Layer 3 consists of the vertices represented by B. A vertex in layer 2 is connected to a vertex in layer 3 if they have the same letter. The weight of every arc between layers 2 and 3 is $M \leq -(\eta + \delta + \psi)$. A sample of a three-layer supergraph is given in Fig. 9. The bold arcs are used to denote the original letters in S (the weight of these arcs is $-\delta$). For simplicity, we omit the first two insertion supernodes before the first letter G. The first supernode thus represents the letter G from the original sequence, which allows for substitution. The second and third supernodes correspond to insertion supernodes, and the fourth supernode corresponds to the letter C and allows substitution as well. There are two more insertion supernodes which are omitted from the graph.

The main step in proving $d(S, B)$ to be polynomial time solvable for any sequences S and B involves the use of the conflict graph as defined in Definition 1. We state some preliminary theoretical results below. Detailed proofs can be found in Lee et al. [50].



Algorithms for Genomic Analysis, Figure 9
An example of the three-layer supergraph for converting the sequence S = GC to B = TC. *Bold arcs* are used to denote the original letters in S (the weight of these arcs is $-\delta$). For simplicity, we omit the first two insertion supernodes before the first letter G. The first supernode thus represents the letter G from the original sequence, which allows for substitution. The second and third supernodes correspond to insertions, and the fourth supernode corresponds to the letter C and allows substitution as well. There are two more insertion supernodes which are omitted from the graph

Lemma 1 *The following statements are equivalent:*

1. *There exists a conversion from S to B using no more than a total of $|B|$ insertions or substitutions.*
2. *There exist a set of noncrossing complete paths in the associated three-layer supergraph G_L of size $|B|$.*
3. *There exists a node packing of size $|B|$ in the associated conflict graph C.*

Lemma 2 *Calculating d(S, B) for any sequences S and B can be accomplished in polynomial time.*

The three-layer supergraph can be generalized to a multilayer supergraph when multiple sequences are considered. Clearly, such multilayer supergraphs are much too large for practical purposes, yet polynomiality is preserved in the construction, and it is therefore sufficient. We can now arrive at the result that the MWCMS is in NP.

Theorem 1 *The MWCMS is in NP.*

To prove that the MWCMS is polynomial time solvable when the number of input sequences is bounded by a positive constant, the following lemma is crucial, though trivial.

Lemma 3 *Given η, δ, $\psi \in \Re^+$, an optimal solution B to any MWCMS problem has the following properties. B has no substitutions from letters other than the original letters in S_i, and B will never have an element which is inserted in every sequence (in the same location). Therefore, there are at most $\sum_{i=1}^{m} |S_i|$ insertions in any sequence.*

In addition, we also require the construction of a (directed) $2m$-layer supergraph, G_L^m, similar to the three-layer supergraph, G_L.

Given sequences $S_1, \ldots, S_m$, generate a $2m$-layer (directed) graph $G_L^m = (V, E)$ as follows. Layers $2i-1$ and $2i$ consist of $(\sum_{j=1}^{m} |S_j|)(|S_i|+1) + |S_i|$ copies of $\sum$-crosses for $i = 1, \ldots, m$, constructed in exactly the same manner as layers 1 and 2 of the three-layer supergraph using the input sequence S_i. The first $\sum_{j=1}^{m} |S_j|$ $\sum$-crosses represent the possibility that $\sum_{j=1}^{m} |S_j|$ different letters can be inserted before the first element in S_i. The next $\sum$-cross corresponds to either the first letter in S_i or a substitution of this letter. This is repeated $|S_i|$ times (for each letter in S_i), and the final $\sum_{j=1}^{m} |S_j|$ $\sum$-crosses represent insertions after the final letter in S_i. Thus, the first $\sum_{j=1}^{m} |S_j|$ $\sum$-crosses represent the insertion supernodes, followed by one $\sum$-cross representing a letter in S_i or a substitution supernode, and so forth. An arc exists from a vertex in layer $2i$ to a vertex in layer $2i+1$ if the vertices correspond to the same letter. Observe that G_L^m is an acyclic directed graph which is polynomial in the size of the input sequences. Assign every arc between layers $2i$ and $2i+1$ a weight of 0. There are three different weights for arcs between layers $2i-1$ and $2i$ each corresponding to an insertion, deletion, or substitution. The assignment of weights on such arcs is analogous to the assignment in G_L: a weight of η is assigned to every arc contained in an insertion supernode; and an arc in a substitution supernode is assigned a weight of $-\delta$ if it corresponds to the original letter, or $\psi - \delta$, otherwise.

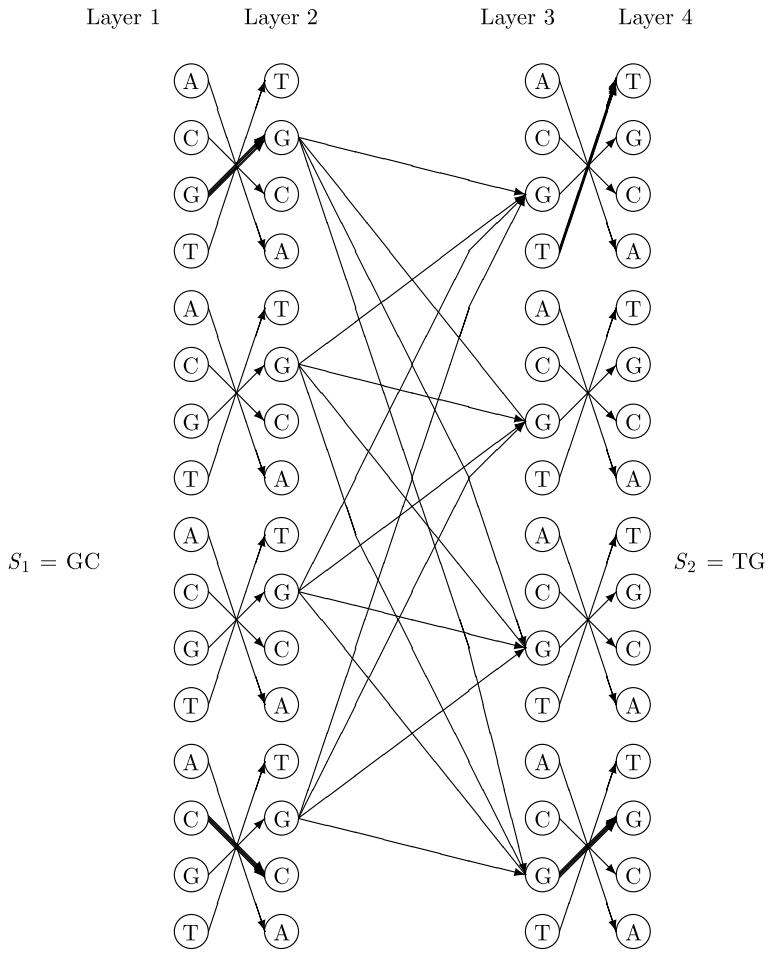
Figure 10 shows a sample graph for two sequences: $S_1 = \text{GC}$ and $S_2 = \text{TG}$. Observe that at most two insertions are needed in an optimal solution; thus, we can reduce the number of $\sum$-crosses as insertion supernodes from $\sum_{i=1}^{2} |S_i| = 4$ to 2. For simplicity, in the graph shown in Fig. 10, we have not included the two insertion supernodes before the first letter nor those after the last letter of each sequence. Thus, in the figure, the first $\sum$-cross represents the substitution supernode associated with the first letter in S_1. The second and third $\sum$-crosses represent two insertion supernodes. And the last $\sum$-cross represents the substitution supernode associated with the second letter in S_1. For simplicity, we include only arcs connecting vertices associated to the element G between layers 2 and 3. The arcs for other vertices follow similarly.

A conflict graph C associated with G_L^m can be generated by finding all complete paths (paths from layer 1 to layer $2m$) in G_L^m. These complete paths correspond to the set of vertices in C, as in Definition 1. If we assign a weight to each vertex equal to the weight of the associated complete path, then the following result can be established.

Theorem 2 *Every node packing in C represents a candidate solution to the MWCMS if and only if at most $\sum_{i=1}^{m} |S_i|$ letters can be inserted between any two original letters. Furthermore, the weight of the node packing is equal to the weight of the MWCMS $- \sum_{i=1}^{m} |S_i|\delta$.*

The supergraph G_L^m and its associated conflict graph are fundamental to our proof of the following theorem on the polynomial-time solvability of a restricted version of the MWCMS problem.

Theorem 3 *The MWCMS problem restricted to instances for which the number of sequences is bounded by a positive constant is polynomial time solvable.*



Algorithms for Genomic Analysis, Figure 10
A sample graph G_L^m of MWCMS with $S_1 = GC$ to $S_2 = TG$, where $\sum = \{A,C,G,T\}$

Special Cases of MWCMS

The MWCMS encompasses a very broad class of problems. In computational biology as discussed in this chapter, first and foremost, it represents a model for phylogenetic analysis. The MWCMS as defined is the "most likely ancestor problem," and the concept of the three-layer supergraph as described in Sect. "Complexity Theory" describes the evolutionary distance problem. An optimal solution to a multiple sequence alignment instance can be found using the solution of the MWCMS problem obtained on the $2m$-layer supergraph, G_L^m. The alignment is the character matrix obtained by placing together the given sequences incorporating the insertions into the solution of the MWCMS problem. Furthermore, DNA sequencing can be viewed as the shortest common superstring problem, while sequence comparison of a given sequence B to a collection of N sequences $S_1, \ldots, S_N$ is the MWCMS problem itself.

Broader than the computational biology applications, special cases of the MWCMS include shortest common supersequences, longest common subsequences, and shortest common superstring; these problems are of interest in their own right as combinatorial optimization problems and for their role in complexity theory.

Computational Models: Integer Programming Formulation

The construction of the multilayer supergraphs described in our theoretical study lays the foundation and provides direction for computational models and solution strategies that we will explore in future research. Although the theoretical results obtained are polynomial-time in nature, they present computational challenges. In many cases, calculating the worst-case scenario is not trivial. Furthermore, the polynomial-time result of a node-packing problem for a perfect graph by Grötschel et.al. [29,30] is existential in nature, and relies on the polynomial-time nature of the ellipsoid algorithm. The process itself involves solving an integer program relaxation multiple times. In our case, the variables of the integer program generated are the complete paths in the multilayer supergraph, G_L^m. Formally, the integer program corresponding to our conflict graph can be stated as follows.

Let x_p be the binary variable denoting the use or nonuse of the complete path p with weight w_p. Then the corresponding node-packing problem is

$$\begin{array}{lll} \text{Minimize} & \sum w_p x_p & \\ \text{subject to} & x_p + x_q \leq 1 & \text{if complete paths } p \text{ and } q \text{ cross} \\ & x_p \in \{0,1\} & \text{for all complete paths } p \text{ in } G_L^m . \end{array} \quad \text{(MIP1)}$$

We call the inequality $x_p + x_q \leq 1$ an adjacency constraint. A natural approach to improve the solution time for (MIP1) is to decrease the size of the graph G_L^m and thus the number of variables. Reductions in the size of G_L^m can be accomplished for shortest common superstrings, longest common subsequences, and shortest common supersequences. Among these three problems, the graph G_L^m is smallest for longest common subsequences. In longest common subsequences, all insertion and substitution supernodes can be eliminated.

Our theoretical results thus far rely on the creation of *all* complete paths. Clearly, the typical number of complete paths will be on the order of n^m, where $n = \max |S_i|$. In this case, an instance with three sequences and 300 letters in each sequence generates more than one million variables; hence, an exact formulation with all complete paths is impractical in general. A simultaneous column and row generation approach within a parallel implementation may lead to computational advances related to this formulation.

An alternative formulation can be obtained by examining G_L^m from a network perspective using arcs (instead of complete paths) in G_L^m as variables. Namely, let $x_{i,j}$ denote the use or nonuse of arc (i,j) in the final sequence, with $c_{i,j}$ the cost of the arc in G_L^m. The network formulation can be stated as

$$\begin{array}{ll} \text{Minimize} & \displaystyle\sum_{(i,j)\in E} c_{i,j} x_{i,j} \\ \text{subject to} & \displaystyle\sum_{i:(i,j)\in E} x_{i,j} = \sum_{k:(j,k)\in E} x_{j,k} \\ & \text{for all } j \in V \text{ in layers } 2,\ldots,2m-1 \\ & x_{i,j} + x_{k,l} \leq 1 \\ & \text{for all crossing arcs } (i,j) \text{ and } (k,l) \in E \\ & x_{i,j} \in \{0,1\} \\ & \text{for all } (i,j) \in E . \end{array} \quad \text{(MIP2)}$$

The first set of constraints ensures flow in equals flow out in all vertices contained in sequences $2,\ldots,m-1$ (complete paths). The second set of constraints ensures that no two arcs cross. This model grows linearly in the number of sequences. This alternative integer programming formulation is still large, but is manageable for even fairly large instances.

Utilizing a collection of DNA sequences (each with 40,000 base pairs in length) from a bacterium, and a collection of short sequences associated with genes found in breast cancer patients, computational tests of our graph-theoretical models are under way. We are seeking to develop computational strategies to provide reasonable running times for evolutionary distance problem instances derived from these data. In an initial test, when three sequences each with 100 letters are used, the initial linear program requires more than 10,000 s to provide a solution when tight constraints are employed (in this case, each adjacency constraint is replaced by a maximal clique constraint). Our ongoing computational effort will focus on developing and investigating solution techniques for practical problem instances, in-

cluding those based on the abovementioned two integer programming formulations, as well as development of fast heuristic procedures.

In [50], we outline a simple yet practical heuristic based on (MIP2) that we developed for solving the multiple sequence alignment problem; and we report on preliminary tests of the algorithm using different sets of sequence data. Motivation for the heuristic is derived from the desire to reduce computational time through various strategies for reducing the number of variables in (MIP2).

Summary

Multiple sequence alignment and phylogenetic analysis are deeply interconnected problems in computational biology. A good multiple alignment is crucial for reliable reconstruction of the phylogenetic tree [58]. On the other hand, most of the multiple alignment methods require a phylogenetic tree as the guide tree for progressive iteration.

Thus, the evolutionary tree construction might be biased by the guide tree used for obtaining the alignment. In order to avoid this pitfall, various algorithms have been developed which simultaneously find alignment and phylogenetic relationship among given sequences. Sankoff and Cedergren [64] developed a parsimony-based algorithm using a character-substitution model of gaps. The algorithm is guaranteed to find the evolutionary tree and alignment which minimizes tree-based parsimony cost. Hein [33] also developed a parsimony-type algorithm but used an affine gap cost, which is more realistic than the character-substitution gap model. This algorithm is also faster than Sankoff and Cedergreen's approach but makes simplifying assumptions in choosing ancestral sequences.

Like parsimony methods for finding a phylogenetic tree, both of the abovementioned approaches require a search over all possible trees to find the global optimum. This makes these algorithms computationally very intensive. Hence, there has been a strong focus on developing an efficient algorithm that considers both alignment and the tree. Vingron and Haeseler [74] have developed an approach based on three-way alignment of prealigned groups of sequences. It also allows change in the alignment made early in the course of computation. Many programs, like MEGA, are trying to develop an efficient integrated computing environment that allows both sequence alignment and evolutionary analysis [48].

We addressed this issue of simultaneously finding alignment and phylogenetic relationships by presenting a novel graph-theoretical approach. Indeed, our model can be easily tailored to find theoretically provable optimum solutions to a wide range of crucial sequence analysis problems. These sequence analysis problems were proven to be NP-hard, and thus understandably present computational challenges. In order to strike a balance between the time and the quality of the solution, a variety of parameters are provided. Ongoing research efforts are exploring the development of efficient computational models and solution strategies in a massive parallel environment.

Acknowledgement

This research was partially supported by grants from the National Science Foundation.

References

1. Abbas A, Holmes S (2004) Bioinformatics and management science: Some common tools and techniques. Oper Res 52(2):165–190
2. Althaus E, Caprara A, Lenhof H, Reinert K (2006) A branch-and-cut algorithm for multiple sequence alignment. Math Program 105(2-3):387–425
3. Altschul S (1991) Amino acid substitution matrices from an information theoretic perspective. J Mol Biol 219(3):555–565
4. Altschul SF, Carroll RJ, Lipman DJ (1989) Weights for data related by a tree. J Mol Biol 207(4):647–653
5. Bains W, Smith G (1988) A novel nethod for DNA sequence determination. J Theor Biol 135:303–307
6. Barton GJ, Sternberg MJE (1987) A strategy for the rapid multiple alignment of protein sequences: confidence levels from tertiary structure comparisons. J Mol Biol 198:327–337
7. Blazewicz J, Formanowicz P, Kasprzak M (2005) Selected combinatorial problems of computational biology. Eur J Oper Res 161:585–597
8. Bonizzoni P, Vedova G (2001) The complexity of multiple sequence alignment with SP-score that is a metric. Theor Comput Sci 259:63–79
9. Bos D, Posada D (2005) Using models of nucleotide evolution to build phylogenetic trees. Dev Comp Immunol 29(3):211–227

10. Bruno WJ, Socci ND, Halpern AL (2000) Weighted neighbor joining: A likelihood-based approach to distance-based phylogeny reconstruction. Mol Biol Evol 17:189–197
11. Carrillo H, Lipman D (1988) The multiple sequence alignment problem in biology. SIAM J Appl Math 48(5):1073–1082
12. Chakrabarti S, Lanczycki CJ, Panchenko AR, Przytycka TM, Thiessen PA, Bryant SH (2006) Refining multiple sequence alignments with conserved core regions. Nucleic Acids Res 34(9):2598–2606
13. Chenna R, Sugawara H, Koike T, Lopez R, Gibson TJ, Higgins DG, Thompson JD (2003) Multiple sequence alignment with the clustal series of programs. Nucleic Acids Res 31(13):3497–3500
14. Chor B, Tuller T (2005) Maximum likelihood of evolutionary trees: hardness and approximation. Bioinf 21(Suppl. 1):I97–I106
15. Clote P, Backofen R (2000) Computational Molecular Biology: An Introduction. Wiley, NY, USA
16. Delsuc F, Brinkmann H, Philippe H (2005) Phylogenomics and the reconstruction of the tree of life. Nature reviews. Genet 6(5):361–375
17. Durbin R, Eddy S, Krogh A, Mitchison G (1998) Biological Sequence Analysis. Cambridge University Press, UK
18. Felsenstein J (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. J Mol Evol 17(6): 368–376
19. Felsenstein J (1988) Phylogenies from molecular sequences: Inference and reliability. Annu Rev Genet 22:521–565
20. Felsenstein J (1989) PHYLIP – phylogeny inference package (version 3.2). Cladistics 5:164–166
21. Fitch WM (1971) Toward defining the course of evolution: Minimum change for a specific tree topology. Syst Zool 20(4):406–416
22. Gallant J, Maider D, Storer J (1980) On finding minimal length superstrings. J Comput Syst Sci 20:50–58
23. Garey M, Johnson D (1979) Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, San Francisco, USA
24. Gascuel O (1997) BIONJ: An improved version of the NJ algorithm based on a simple model of sequence data. Mol Biol Evol 14(7):685–695
25. Goeffon A, Richer J, Hao J (2005) Local search for the maximum parsimony problem. Lect Notes Comput Sci 3612:678–683
26. Golumbic MC, Rotem D, Urrutia J (1983) Comparability graphs and intersection graphs. Discret Math 43:37–46
27. Gotoh O (1996) Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments. J Mol Biol 264(4):823–838
28. Gotoh O (1999) Multiple sequence alignment: algorithms and applications. Adv Biophys 36:159–206
29. Grötschel M, Lovász L, Schrijver A (1984) Polynomial algorithms for perfect graphs. Annals Discret Math 21:325–356
30. Grötschel M, Lovász L, Schrijver A (1988) Geometric algorithms and combinatorial optimization. Springer, New York
31. Guindon S, Gascuel O (2003) A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. Syst Biol 52(5):696–704
32. Gupta S, Kececioglu J, Schaeffer A (1995) Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment. J Comput Biol 2:459–472
33. Hein J (1989) A new method that simultaneously aligns and reconstructs ancestral sequences for any number of homologous sequences, when the phylogeny is given. Mol Biol Evol 6(6):649–668
34. Huelsenbeck J, Crandall K (1997) Phylogeny estimation and hypothesis testing using maximum likelihood. Annu Rev Ecol Syst 28:437–66
35. Hughey R, Krogh A (1996) Hidden markov models for sequence analysis: extension and analysis of the basic method. Comput Appl Biosci 12(2):95–107
36. Idury RM, Waterman MS (1995) A new algorithm for DNA sequence assembly. J Comput Biol 2(2):291–306
37. Jukes TH, Cantor CR (1969) Evolution of protein molecules. In: Munro HN (ed) Mammalian Protein Metabolism. Academic Press, New York, pp 21–123
38. Just W, Vedova G (2004) Multiple sequence alignment as a facility-location problem. INFORMS J Comput 16(4):430–440
39. Keane T, Naughton T, Travers S, McInerney J, McCormack G (2005) DPRml: distributed phylogeny reconstruction by maximum likelihood. Bioinf 21(7):969–974
40. Kececioglu J, Lenhof H, Mehlhorn K, Mutzel P, Reinert K, Vingron M (2000) A polyhedral approach to sequence alignment problems. Discret Appl Math 104:143–186
41. Kim J, Pramanik S, Chung MJ (1994) Multiple sequence alignment using simulated annealing. Bioinf 10(4):419–426
42. Kimura M (1980) A simple method for estimating evolutionary of base substitution through comparative studies of nucleotide sequences. J Mol Evol 16:111–120
43. Klotz L, Blanken R (1981) A practical method for calculating evolutionary trees from sequence data. J Theor Biol 91(2):261–272
44. Korostensky C, Gonnet GH (1999) Near optimal multiple sequence alignments using a traveling salesman problem approach. In: Proceedings of the String Processing and Information Retrieval Symposium. IEEE, Cancun, pp 105–114
45. Korostensky C, Gonnet GH (2000) Using traveling salesman problem algorithms for evolutionary tree construction. Bioinf 16(7):619–627
46. Krogh A, Brown M, Mian IS, Sjolander K, Haussler D (1994) Hidden markov models in computational biology: Applications to protein modeling. J Mol Biol 235:1501–1531

47. Kumar S, Tamura K, Nei M (1994) MEGA: Molecular evolutionary genetics analysis software for microcomputers. Comput Appl Biosci 10:189–191
48. Kumar S, Tamura K, Nei M (2004) MEGA3: integrated software for molecular evolutionary genetics analysis and sequence alignment. Brief Bioinform 5(2):150–163
49. Lawrence C, Altschul S, Boguski M, Liu J, Neuwald A, Wootton J (1993) Detecting subtle sequence signals: a gibbs sampling strategy for multiple alignment. Science 262:208–214
50. Lee EK, Easton T, Gupta K (2006) Novel evolutionary models and applications to sequence alignment problems. Annals Oper Res 148(1):167–187
51. Levenshtein VL (1966) Binary codes capable of correcting deletions, insertions, and reversals. Cybern Control Theor 10(9):707–710
52. Li W (1981) Simple method for constructing phylogenetic trees from distance matrices. Proc Natl Acad Sci USA 78(2):1085–1089
53. Lipman D, Altschul S, Kececioglu J (1989) A tool for multiple sequence alignment. Proc Natl Acad Sci USA 86(12):4412–4415
54. Maier D, Storer JA (1977) A note on the complexity of the superstring problem. Technical Report 233, Princeton University, USA
55. Nei M (1996) Phylogenetic analysis in molecular evolutionary genetics. Annu Rev Genet 30:371–403
56. Notredame C (2002) Recent progress in multiple sequence alignment: a survey. Pharmacogenomics 3(1):131–144
57. Notredame C, Higgins D (1996) SAGA: sequence alignment by genetic algorithm. Nucleic Acids Res 24(8):1515–1524
58. Phillips A, Janies D, Wheeler W (2000) Multiple sequence alignment in phylogenetic analysis. Mol Phylogenet Evol 16(3):317–330
59. Piontkivska H (2004) Efficiencies of maximum likelihood methods of phylogenetic inferences when different substitution models are used. Mol Phylogenet Evol 31(3):865–873
60. Purdom P, Bradford PG, Tamura K, Kumar S (2000) Single column discrepancy and dynamic max-mini optimizations for quickly finding the most parsimonious evolutionary trees. Bioinformamtics 16:140–151
61. Reinert K, Lenhof H, Mutzel P, Mehlhorn K, Kececioglu J (1997) A branch-and-cut algorithm for multiple sequence alignment. In: Proceedings of the First Annual International Conference on Computational Molecular Biology (RECOMB-97). ACM Press, Santa Fe, pp 241–249
62. Ronquist F (1998) Fast fitch-parsimony algorithms for large data sets. Cladistics 14:387–400
63. Saitou N, Nei M (1987) The neighbor-joining method: a new method for reconstructing phylogenetic trees. Mol Biol Evol 4:406–425
64. Sankoff D, Cedergren RJ (1983) Simultaneous comparison of three or more sequences related by a tree. In: Sankoff D, Kruskal JB (eds) Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison. Addison-Wesley, MA, USA, pp 253–264
65. Shyu SJ, Tsai YT, Lee R (2004) The minimal spanning tree preservation approaches for DNA multiple sequence alignment and evolutionary tree construction. J Comb Optim 8(4):453–468
66. Sokal R, Michener C (1958) A statistical method for evaluating systematic relationships. University of Kansas, Scientific Bull 38:1409–1438
67. Stamatakis A, Ott M, Ludwig T (2005) RAxML-OMP: An efficient program for phylogenetic inference on SMPs. Lect Notes Comput Sci 3606:288–302
68. Swofford DL, Maddison WP (1987) Reconstructing ancestral character states under wagner parsimony. Math Biosci 87:199–229
69. Swofford DL, Olsen GJ (1990) Phylogeny reconstruction. In: Hillis DM, Moritz G (eds) Molecular Systs. Sinauer Associates, MA, USA, pp 411–501
70. Tajima F, Nei M (1984) Estimation of evolutionary distance between nucleotide sequences. Mol Biol Evol 1(3):269–85
71. Tajima F, Takezaki N (1994) Estimation of evolutionary distance for reconstructing molecular phylogenetic trees. Mol Biol Evol 11:278–286
72. Takahashi K, Nei M (2000) Efficiencies of fast algorithms of phylogenetic inference under the criteria of maximum parsimony, minimum evolution, and maximum likelihood when a large number of sequences are used. Mol Biol Evol 17:1251–1258
73. Thompson JD, Higgins DG, Gibson TJ (1994) CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic Acids Res 22(22):4673–4680
74. Vingron M, Haeseler A (1997) Towards integration of multiple alignment and phylogenetic tree construction. J Comput Biol 4(1):23–34
75. Vingron M, Waterman M (1994) Sequence alignment and penalty choice. review of concepts, case studies and implications. J Mol Biol 235(1):1–12
76. Wallace IM, O'Sullivan O, Higgins DG (2005) Evaluation of iterative alignment algorithms for multiple alignment. Bioinformatics 21(8):1408–14
77. Waterman M, Perlwitz M (1984) Line geometries for sequence comparisons. Bull Math Biol 46(4):567–577
78. Waterman MS (1995) Introduction to Computational Biology: Maps, Sequences and Genomes. Chapman and Hall
79. Whelan S, Lio P, Goldman N (2001) Molecular phylogenetics: state-of-the-art methods for looking into the past. Trends Genet 17(5):262–272
80. Yang Z (1993) Maximum-likelihood estimation of phylogeny from DNA sequences when substitution rates differ over sites. Mol Biol Evol 10(6):1396–401
81. Zhang Y, Waterman M (2003) An eulerian path approach to global multiple alignment for DNA sequences. J Comput Biol 10(6):803–819

Alignment Problem

CLAUDE G. DIDERICH[1], MARC GENGLER[2]
[1] Computer Sci. Department,
Swiss Federal Institute Technology-Lausanne,
Lausanne, Switzerland
[2] Ecole Sup. d'Ingénieurs de Luminy,
University Méditerrannée, Marseille, France

MSC2000: 05-02, 05-04, 15A04, 15A06, 68U99

Article Outline

Keywords

Alignment problem; Automatic parallelization; Computation and data mapping; Nested loops; Scheduling functions

Since the mid-1990s the need for techniques to parallelize numerical applications has increased. When parallelizing nested loops for distributed memory parallel computers, two major problems have to be solved: the scheduling of the loop iterations and the mapping of the computations and data elements onto the processors. The scheduling functions must satisfy all the data dependences existing in the sequential loop nests. The mapping functions should maximize the degree of parallelism obtained. Furthermore they should minimize the amount of communication overhead due to non local data references.

This survey presents the *alignment problem*, that is, the problem of mapping computation and data onto the processors. The alignment problem has been studied extensively since the beginning of the nineties, that is, since the beginning of the introduction of massively parallel distributed memory computers. For different sub-problems of the alignment problem, the most interesting results are surveyed.

Alignment Problem

The alignment problem is the problem of finding an alignment of loop iterations with the array elements accessed. This means computing mapping functions of the loop iterations, called computations, and mapping functions of the array elements, called data, to a multidimensional grid of virtual processors. The name of the problem comes from the idea of aligning the processors computing with the ones owning the data. The alignment problem is tightly related to the mapping of the computation and data objects onto a grid of virtual processors.

As input, programs containing nested loops are considered. Each loop nest may contain one or more instructions. For the sake of simplicity, only assignment instructions are considered. The data access functions are described by the functions F_l: $\mathbf{I}_j \rightarrow \mathbf{D}_K$, where $\mathbf{I}_j$ represents the iteration space surrounding instruction S_j and $\mathbf{D}_K$ the domain of the array K.

To solve the alignment problem, computation and data mapping functions C_j and D_K have to be computed such as to minimize the overall execution time of the resulting parallel program.

$$C_j : \mathbf{I}_j \rightarrow \mathbf{P},$$
$$D_K : \mathbf{D}_K \rightarrow \mathbf{P},$$

where $\mathbf{P}$ represents a multidimensional grid of virtual processors.

To minimize the overall execution time a solution to the alignment problem has to address the following needs:

i) maximize the degree of parallelism, that is, use as many dimensions of the virtual grid of processors as possible,
ii) minimize the need for non local data accesses, that is, distribute the array elements such that a minimal amount of communication overhead is required to

access data elements stored on different processors than the ones accessing them,

iii) guarantee the existence of scheduling functions compatible with the computation mapping functions.

Clearly the needs i)–iii) depend on each other. In this survey we only focus on the first two needs.

Need i) can be expressed by maximizing the dimension of the virtual processor grid **P** onto which the computations and data elements are mapped.

The need for a given data access F_l to be local is expressed by the equation (1) being satisfied:

$$C_j(\vec{\imath}) = D_K(F_l(\vec{\imath})). \tag{1}$$

Equation (1) is called *alignment constraint* or *locality constraint*. Depending on how the needs i) and ii) are satisfied, various subproblems of the alignment problem can be defined.

Communication-Free Alignment Problem

The *communication-free alignment problem* (CFAP) is the problem of finding computation and data mapping functions for each instruction and for each data array such that no communication is needed and the degree of parallelism obtained is maximal. The CFAP can be formulated as an optimization problem:

$$\begin{cases} \max_{C_j, D_K} & \text{dimension of } \mathbf{P} \\ \text{s.t.} & \forall j, l, K\colon\ C_j(\vec{\imath}) = D_K(F_l(\vec{\imath})). \end{cases}$$

Constant-Degree Parallelism Alignment Problem

Let $\mathcal{F}$ be the set of data access functions from a set of loop nests forming an alignment problem and d a positive constant. Let $c(\mathcal{F}', \mathcal{F})$ be a cost function on a subset $\mathcal{F}' \subseteq \mathcal{F}$ of data access functions. The *constant degree parallelism alignment problem* (CDPAP), denoted by $(\mathcal{F}, d)$, is the problem of finding a subset $\mathcal{F}' \subseteq \mathcal{F}$ of data access functions such that:

1) There exists a solution to the CFAP consisting of all data accesses in the set $\mathcal{F}'$ admitting a degree of parallelism of at least d.
2) The cost function $c(\mathcal{F}', \mathcal{F})$ on the subset $\mathcal{F}'$ is minimized.

As for the CFAP, the CDPAP can be formulated as follows as an optimization problem:

$$\begin{cases} \max_{C_j, D_K} & \sum_{j,l,K}[[C_j(\vec{\imath}) = D_K(F_l(\vec{\imath}))]] \\ \text{s.t.} & \text{dimension of } \mathbf{P} \geq d. \end{cases}$$

Example 1 The data accesses in this example are encoded by the three functions $F_1(i, j) = (i\, j + 1)$, $F_2(i, j) = (i{-}1\, j + 1)$ and $F_3(i, j) = (i + 1\, j + 1)$. A possible solution requiring no communication and admitting one degree of parallelism is given by $C(i, j) = j$ and $D_a(i, j) = j - 1$, **P** being a one-dimensional processor set.

```
DO i = 2, n − 1
    DO j = 2, n − 1
        a(i, j + 1) = a(i − 1, j + 1) + a(i + 1, j + 1)
    END DO
END DO
```

Solving the Alignment Problem

Communication-Free Alignment Approaches

C.-H. Huang and P. Sadayappan [17], in 1991, were the first to formulate the alignment problem in a linear algebra framework. They focus on a communication-free solution. The data array elements as well as the loop iterations are partitioned in disjoint sets represented by hyperplanes. Each set is mapped onto a different processor. The partitions are sought such that they result in the elimination of communication. A characterization of a necessary and sufficient condition for communication-free hyperplane partitioning is provided. Various results are given characterizing the situation where the iteration and data space can be partitioned along hyperplanes so that no communication is necessary. More precisely, two data elements accessed during a single iteration in a single instruction must be located on a single processor and two iterations in the same instruction accessing a single data element must be executed on the same processor.

In [30], a matrix notation is presented to describe array accesses in fully parallel loop nests. A sufficient condition on the matrices for computing a communication-free mapping of the arrays onto the processors is given. The owner computes rule is assumed for the computation mapping. The presented

existence condition for communication-free partitions is based on the connectivity of the data access graph which models the data access patterns. To compute data mapping functions, a set of systems of linear equations is constructed, one system of linear equations per pair of read and write data accesses. If there exists a solution to the set of systems of linear equations, then there exists a communication-free partitioning of the array elements into parallel hyperplanes.

In [2] a linear algebra approach is proposed, based on [17]. The communication-free alignment problem is solved by computing a basis of the null space of the application representing the alignment constraints. The problem of data replication is addressed.

In [6], T.-S. Chen and J.-P. Sheu consider perfect loop nests. They compute iteration and data space partitioning functions requiring no communication. Their work focuses only on uniformly generated data references. Sufficient conditions are given for the existence of a communication-free partition. The method for partitioning the data onto the processors is based on the computation of independent blocks called iteration and data partitions respectively. If no communication-free partitioning exists, data replication is considered.

In [24], an algorithm is presented that extracts all the degrees of communication-free parallelism that can be obtained via loop fission, fusion, interchange, reversal, skewing, scaling, re-indexing and statement reordering. The algorithm first assigns the iterations of the instructions in the program to processors via affine processor mapping functions. Then it generates the correct code by assuring that the semantics of the sequential program are satisfied.

Alignment Approaches Based on Generating HPF like Data Distributions

J. Li and M. Chen [22,23] are interested in the indices of the arrays that have to be aligned with one another to minimize remote data references. The techniques were initially developed for compiling the functional language 'Crystal', but can be applied in the process of compiling imperative languages like 'Fortran'. The parallelism is assumed to be specified explicitly and the single assignment form is used. The goal of their approach is to find alignment functions such that the dimensions of each array are projected onto the same space of a virtual processor grid. They consider four basic alignments:

i) permutations of the indices,
ii) embeddings,
iii) translations by a constant, and
iv) reflections.

To find a set of data accesses for which valid alignment functions exist, a component affinity graph is constructed. It represents the affinities between cross reference patterns. The nodes of the graph represent the components of the index domains to be aligned. An edge represents an affinity between the two corresponding domain components. The alignment problem then consists in partitioning the set of nodes of the component affinity graph into disjoint subsets with the restriction that no two nodes belonging to the same array are allowed in the same subset. A fast and quite efficient heuristic algorithm is presented.

M. Gupta, in his thesis in 1992 [16], presents a data distribution algorithm that operates in four passes. The first pass serves to compute an alignment of the array dimensions. The algorithm developed is based on the notion of component affinity graph introduced by Li and Chen [22]. In the second phase the arrays are partitioned using either block or cyclic data distributions. In the third pass, the block sizes of the arrays distributed are computed whereas the last pass computes the number of processors on which each array dimension is distributed.

K. Kunchithapadam and B.P. Miller [20], in opposition to other approaches, assume that a user-defined data distribution is given. The data accesses of a program are modeled by a colored proximity graph. Each vertex of the graph represents a part of an array and the color of a vertex represents the current processor to which this array part is assigned. Edges of the graph represent assignments of values arising from part of one or more arrays to part of another array assuming the owner computes rule for the computation mapping. Edges between vertices of different colors are assigned a weight representing the associated communication costs. The problem of improving a given set of data mapping functions is to find a sequence of color exchanges, that is, data redistributions, that minimize the weight of the graph, that is, the communication costs. A possible algorithm for solving this problem is presented.

B. Sinharoy and B.K. Szymanski [32] study the problem of finding computation and data alignment functions for regular iterative algorithms. A loop nest can be represented by a regular iterative algorithm if and only if all the data access functions are constant offset functions and the loop nest's instructions are in single assignment form. The communication cost function used is based on the distance of the processors exchanging data on the virtual processor grid. The authors show that finding computation and data mapping functions is equivalent to minimizing a sum of absolute values composed of sums. An exact enumeration algorithm is presented and a polynomial time algorithm for finding an approximate solution is described.

Approaches Using a Graph Based Framework

K. Knobe, J.D. Lukas and G.L. Steele Jr. [19] study the problem of aligning the array elements accessed amongst each other. They target their approach towards SIMD machines. Two different kinds of preferences are distinguished:

i) identity preferences representing alignment preferences due to different data accesses to the same array, and
ii) conformance preferences relating two different arrays.

To compute what preferences can be satisfied without loosing parallelism, a cyclic preference graph is constructed. Each data access is represented by a vertex and two vertices are related by an undirected weighted edge if there exists a preference between the two data accesses. The weight of each edge is defined by the loop depth at which the data accesses occur. Conflicts between preferences are represented by cycles in the cyclic preference graph. A heuristic, using a greedy approach, is presented to remove annoying cycles or to reduce the parallelism.

In [5] an intermediate representation of a program called the alignment-distribution graph is described. The *alignment-distribution graph* is a directed graph in which nodes represent communication and edges represent the data flow. It exposes the communication requirements of the program. The framework restricts the alignments computed to alignments in which each axis of an array maps to a different axis of an HPF like template and data elements are evenly spaced along the template axis. The alignments computed have three components:

i) the axis,
ii) the stride, and
iii) the offset.

The papers present two separate algorithms called the compact dynamic programming algorithm and the constraint graph method for minimizing a communication cost function.

A. Darte and Y. Robert [8] study the problem of mapping perfectly nested affine loops onto distributed memory parallel computers. The problem is formulated by introducing the communication graph that captures all the required information to align data and computations. Each instruction and each array is represented by a vertex, the directed edges representing read and write data accesses. The problem of message vectorization and the use of global communication operations, like broadcasting, is addressed.

In [11] an algorithm is presented for computing HPF like data distribution functions. A distribution graph is constructed representing the relation between the data access functions and the array accessed. Based on the distribution graph a decision tree, modeling all possible combinations of data distribution functions, is traversed using a branch and bound algorithm. The cost function minimized by the algorithm is based on a communication analysis tool. The computation mapping is done in accordance with the owner computes rule.

M. Wolfe and M. Ikey [33] propose in 1994 an adaption of the techniques introduced by Li and Chen [22,23] for the language 'Crystal' to the imperative language 'Tiny'. The alignment phase is decomposed into four operations:

i) finding reference patterns,
ii) adding implicit dimensions to the arrays when required,
iii) building a component affinity graph, and
iv) partitioning the component affinity graph.

As the partitioning problem is *NP*-hard, a heuristic is used. The authors furthermore describe an algorithm to generate SPMD code based on the alignments computed.

J. Garcia, E. Ayguagé and J. Labarta [15] proposed for an algorithm to compute data distribution functions that can be expressed using HPF distribute statements. This algorithm is based on the construction and traver-

sal of a single data structure, called the computation-parallelism graph. The computation-parallelism graph represents all possible data distributions along the dimensions of the arrays. Parallelism constraints are modeled as hyper-edges. Weights are associated to the edges to represent the associated communication costs. Negative costs are associated with the hyperedges to represent the associated parallelism. It is shown that distributing the data according to one dimension is equivalent to finding a path through the computation-parallelism graph fulfilling some additional constraints. The problem is formulated as a 0–1 integer programming problem. In contrast to other graph based approaches, the computation-parallelism graph models both the possible data distribution, that is, the locality constraints within a single data structure, and the possible parallelism.

W. Kelly and W. Pugh [18] describe a technique to minimize communication while preserving parallelism. The approach is not sensitive to the original program structure. For each array, the possible data mapping functions form a finite set of candidate space mappings. These sets consist of each dimension of the original iteration space being distributed. Next, for each candidate space, that is, for each possible data distribution function, all possible permutations of the surrounding loops are considered and the obtained parallelism measured. In a third step a weighted graph is constructed to model the parallelism as well as the communication cost associated with various data decompositions. One node in this weighted graph represents one candidate space mapping for each statement. The weight associated with a node is its degree of parallelism obtained. The edges represent the communication required and their weight models the communication costs. The alignment problem, as formulated in [18], is the problem of selecting one node per statement such that the sum of the weights of the selected nodes and edges is minimized. An algorithm to find such a set using various pruning strategies to reduce the size of the search space is presented.

Approaches Using a Linear Algebra Framework

Sheu and T.-H. Toi [31] introduced a method for the parallel execution of nested loops with constant loop-carried data dependences by reducing the communication overhead. First the nested loops are partitioned into large blocks which result in little inter-block communication. For a given linear transformation found by the hyperplane method [21], the iterations are partitioned into blocks such that the communication among the blocks is reduced while the execution order defined by the time transformation is not disturbed. The partitioning is based on projection techniques. In a second step these blocks are mapped onto message-passing multiprocessor systems according to specific properties of the target machine.

M. O'Boyle and G.A. Hedayat [26,27] express the alignment problem in a linear algebra framework. In this framework, aligned data can be viewed as forming a subspace in the iteration space. The problem solved is the computation of a transformation of the data access functions relative to one another such as to maximize the number of iteration points in the loop iteration space for which no communication is needed.

P. Feautrier [14] addresses the problem of finding an alignment function that maps the computations on a one-dimensional grid of virtual processors. The data mapping functions are defined by the owner computes rule which is imposed. The alignment constraints between computation and data accesses are derived from the data-flow graph of the program, procedure or loop nest considered. The data-flow graph is a directed graph. Vertices correspond to statements and the arcs to producers and consumers of data. For each statement, the alignment function is assumed to be an affine function of the iteration vectors with unknown parameters. The locality of data accesses is imposed by asking that the producer and the consumer of a data element be the same processor. Feautrier defines distance vectors between all pairs of producers and consumers. To any arc of the data-flow graph corresponds a distance vector that expresses the difference of the indices of the processor that computes the data and the one that uses it. Thus, a computation is local if and only if the corresponding distance vector is zero. The edges are hence transformed into affine equations and the problem consists in determining nontrivial parameters for the computation mappings that zero out as many distance vectors as possible. A heuristic is used to sort the equations in decreasing order of the communication traffic induced. The system of equations, which usually does not have a non trivial solution, is solved by successive

Gauss–Jordan eliminations as long as a feasible solution remains nontrivial. A solution is nontrivial if it has one degree of parallelism.

J.M. Anderson and M.S. Lam [1] describe necessary conditions for the data elements accessed by each processor to be local. They present a greedy algorithm to compute the computation and data mapping functions that can be satisfied. They incrementally add constraints as long as their conditions are satisfied, starting with the most frequently used array access functions. They only consider the linear part of the data access functions, taking care of the constant offsets in a second step. Their heuristic technique is close to the one defined in [9].

A. Platonoff [28,29] develops extensions to Feautrier's [14] automatic data distribution algorithm. A method is presented to extract global broadcast operations as well as translation operations to optimize the data mapping functions. In the data-flow graph, patterns representing broadcast and other global communication patterns are searched for. The data distribution is then chosen such as to maximize the number of global communication operations possible.

M. Dion and Robert [12,13] consider a problem in which all data access functions are of full rank and no smaller than d, the required degree of parallelism. This ensures that the parallelism obtained is indeed as large as wanted. By considering only the linear parts they compute the largest set of alignment constraints that can be satisfied while yielding the given degree of parallelism d. The constant offsets are considered subsequently, using techniques developed by Darte and Robert [8]. They consider a set of candidate solutions and search for an optimal one that verifies the largest number of constraints while effectively yielding the degree of parallelism desired. In their approach, Dion and Robert consider three basic cases depending on the structure of the data access function. Then, they build a directed graph defined as follows. Vertices correspond either to statements or arrays. There is an arc from vertex p to vertex q if and only if a mapping of rank d can be computed for q from a given mapping of rank d for p according to the basic cases enumerated previously. In this graph they search for a tree containing a maximal number of arcs. Obviously, choosing a mapping of rank d for the root of the computed tree implicitly determines mappings of rank d for all other vertices.

C. Mongenet [25] is interested in minimizing communication costs in the presence of systems of affine recurrence equations, that is, single assignment loop nests. The data dependences are subdivided into two classes:

i) auto dependences, and
ii) cross dependences.

Auto-dependences are data dependences between two data accesses to the same array. The domains of these arrays are projected onto hyperplanes such as to minimize the number of remote data accesses. Cross-dependences are dependences between data accesses to different arrays. Unimodular transformations are applied to the projected domains to align the different data array and so minimize the resulting communications. A heuristic based on these two steps is introduced.

C.G. Diderich [9] and Diderich and M. Gengler [10] present and extend the algorithm for solving this problem introduced in [2]. In a second step they introduce the constant degree parallelism alignment problem. It is the problem of finding computation and data mapping functions that minimize the number of remote data accesses for a given degree of parallelism. An exact implicit enumeration algorithm is presented. It proceeds by enumerating all interesting subsets of alignment constraints to be satisfied. To allow large alignment problems to be solved an efficient heuristic is presented and applied to various benchmarks.

Other Approaches

B.M. Chapman, T. Fahringer and H.P. Zima [4] for a software tool to provide automatic support for the mapping of the data onto the processors of the target machine. The computation is mapped by using the owner computes rule. The tool is integrated within the *Vienna Fortran Compilation System*, a compiler for Vienna Fortran, an HPF like Fortran dialect. The tool makes use of performance analysis methods and uses, via heuristics, empirical performance data. Once the performance data has been obtained for a given program, an inter-procedural alignment and pattern matching phase determines a suitable alignment of the arrays within each procedure. The alignments are then propagated through the call graph of the program.

Eventually more versions of a procedure are generated, corresponding to differently distributed actual arguments. Finally code is generated using the selected data distributions.

In [7], P. Crooks and R.H. Perrott present an algorithm for determining data mapping functions by generating HPF like directives. Their approach is based on identifying reference patterns. To each read/write pair is associated an ideal data distribution that minimized inter-processor communication. Once the preferences for the individual accesses are determined, a performance estimator is used to select the combination of preferences that gives the best performance estimate.

R. Bixby, K. Kennedy and U. Kremer [3] present an automatic data layout algorithm based on 0–1 integer programming techniques. The data mapping functions, following the HPF alignment structure, are optimized for a target distributed memory machine, a specific problem size and the number of available processors. The distribution analysis uses the alignment search space, that is, the space of all possible HPF like alignments, to build candidate data layout search spaces of reasonable data mapping functions for each loop nest. In a second step the inter-phase or inter-loop nests data layout problem is addressed. By using an integer programming formulation, a data mapping function is selected for each loop nest such that a single global cost function, modeling the communication costs, is minimized.

Conclusion

This article presents major advancements made in solving the alignment problem. Different subproblems are defined and described. One major open problem is how to incorporate scheduling information into the algorithms computing efficient alignment functions. See [9] for a first approach towards computing scheduling functions compatible with computation and data mapping functions. The question of which cost function to use when computing alignment functions has to be addressed with more details.

See also

► Integer Programming

References

1. Anderson JM, Lam MS (1993) Global optimizations for parallelism and locality on scalable parallel machines. In: ACM SIGPLAN Conf. Programming Language Design and Implementation (PLDI '93). ACM, New York, pp 112–125
2. Bau D, Kodukula I, Kotylar V, Pingali K, Stodghill P (1994) Solving alignment using elementary linear algebra. In: 7th Internat. Workshop Languages and Compilers for Parallel Computing (LCPC '94). In: Lecture Notes Computer Sci, vol 892. Springer, Berlin, pp 46–60
3. Bixby R, Kennedy K, Kremer U (1994) Automatic data layout using 0–1 integer programming. Internat. Conf. Parallel Architectures and Compilation Techniques (PACT '94). pp 111–122
4. Chapman BM, Fahringer T, Zima HP (1993) Automatic support for data distribution on distributed memory multiprocessor systems. In: 6th Internat. Workshop Languages and Compilers for Parallel Computing (LCPC '93). In: Lecture Notes Computer Sci, vol 768. Springer, Berlin, pp 184–199
5. Chatterjee S, Gilbert JR, Schreiber R, Sheffler TJ (1994) Array distribution in data-parallel programs. In: 7th Internat. Workshop Languages and Compilers for Parallel Computing (LCPC '94). In: Lecture Notes Computer Sci, vol 892. Springer, Berlin, pp 78–91
6. Chen T-S, Sheu J-P (1994) Communication-free data allocation techniques for parallelizing compilers on multicomputers. IEEE Trans Parallel and Distributed Systems 5(9):921–938
7. Crooks P, Perrott RH (1993) An automatic data distribution generator for distributed memory MIMD machines. In: 4th Internat. Workshop Compilers for Parallel Computers, pp 33–44
8. Darte A, Robert Y (1994) On the alignment problem. Parallel Proc Lett 4(3):259–270
9. Diderich CG (1998) Automatic data distribution for massively parallel distributed memory computers. PhD Thesis. Computer Sci. Dept. Swiss Federal Inst. Tech., Lausanne
10. Diderich CG, Gengler M (1997) The alignment problem in a linear algebra framework. In: Proc. Hawaii Internat. Conf. System Sci. (HICSS-30); Software Techn. Track. IEEE Computer Soc Press, New York, pp 586–595
11. Dierstein A, Hayer R, Rauber T (1994) The ADDAP system on the iPSC/860: Automatic data distribution and parallelization. J Parallel Distributed Comput 32(9):1–10
12. Dion M (1996) Alignement et distribution en parallélisation automatique. PhD Thesis. Ecole Normale Sup. Lyon (In French)
13. Dion M, Robert Y (1996) Mapping affine loop nests. Parallel Comput 22:1373–1397
14. Feautrier P (1992) Towards automatic distribution. Parallel Proc Lett 4(3):233–244
15. Garcia J, Ayguadé E, Labarta J (1995) A novel approach towards automatic data distribution. In: Supercomputing '95 Conf

16. Gupta M (1992) Automatic data partitioning on distributed memory multicomputers. PhD Thesis. Univ. Illinois at Urbana-Champaign, Urbana, IL
17. Huang C-H, Sadayappan P (1991) Communication-free hyperplane partitioning of nested loops. In: 4th Internat Workshop Languages and Compilers for Parallel Computing (LCPC '91), vol 589. In: Lecture Notes Computer Sci, vol 589. Springer, Berlin, pp 186–200
18. Kelly W, Pugh W (1996) Minimizing communication while preserving parallelism. In: 1996 ACM Internat. Conf. Supercomputing (ICS '96). ACM, New York, pp 52–60
19. Knobe K, Lukas JD, Steele GL Jr (1990) Data optimization: Allocation of arrays to reduce communication on SIMD machines. J Parallel Distributed Comput 8(2):102–118
20. Kunchithapadam K, Miller BP (1994) Optimizing array distributions in data-parallel programs. In: 7th Internat. Workshop Languages and Compilers for Parallel Computing (LCPC '94). In: Lecture Notes Computer Sci, vol 892. Springer, Berlin, pp 470–484
21. Lamport L (1974) The parallel execution of DO loops. Comm ACM 17(2):83–93
22. Li J, Chen M (1990) Index domain alignment: Minimizing cost of cross-referencing between distributed arrays. In: 3rd Symp. Frontiers of Massively Parallel Computation (Frontiers '90). IEEE Computer Soc Press, New York, pp 424–433
23. Li J, Chen M (1991) The data alignment phase in compiling programs for distributed-memory machines. J Parallel Distributed Comput 13:213–221
24. Lim AW, Lam MS (1994) Communication-free parallelization via affine transformations. In: 7th Internat. Workshop Languages and Compilers for Parallel Computing (LCPC '94). In: Lecture Notes Computer Sci, vol 892. Springer, Berlin, pp 92–106
25. Mongenet C (1995) Mappings for communications minimization using distribution and alignment. In: Internat. Conf. Parallel Architectures and Compilation Techniques (PACT '95). pp 185–193
26. O'Boyle M (1993) A data partitioning algorithm for distributed memory compilation. Techn Report Ser Univ Manchester, England UMCS-93-7-1
27. O'Boyle M, Hedayat GA (1992) Data alignment: Transformation to reduce communication on distributed memory architectures. In: Scalable High Performance Computing Conf. (SHPCC '92). IEEE Computer Soc Press, New York, pp 366–371
28. Platonoff A (1995) Automatic data distribution for massively parallel computers. In: Int. Workshop Compilers for Parallel Computers, pp 555–570
29. Platonoff A (1995) Contribution à la distribution automatique des données pour machines massivement parallèles. PhD Thesis. Ecole Normale Sup. Mines de Paris (In French)
30. Ramanujam J, Sadayappan P (1991) Compile-time techniques for data distribution in distributed memory machines. IEEE Trans Parallel and Distributed Systems 2(4): 472–482
31. Sheu J-P, Tai T-H (1991) Partitioning and mapping nested loops on multiprocessor systems. IEEE Trans Parallel and Distributed Systems 2(4):430–439
32. Sinharoy B, Szymanski BK (1994) Data and task alignment in distributed memory architectures. J Parallel Distributed Comput 21:61–74
33. Wolfe M, Ikei M (1994) Automatic array alignment for distributed memory multicomputers. 27th Annual Hawaii Internat. Conf. System Sci., vol II. IEEE Computer Soc. Press, New York, pp 23–32

αBB Algorithm

CLAIRE S. ADJIMAN,
CHRISTODOULOS A. FLOUDAS
Department Chemical Engineering,
Princeton University, Princeton, USA

MSC2000: 49M37, 65K10, 90C26, 90C30

Article Outline

Keywords

Global optimization; Interval arithmetic; Twice-differentiable NLPs; Branch and bound; αBB algorithm

Deterministic global optimization techniques for nonconvex NLPs have been the subject of growing interest because they can potentially provide a very complete characterization of the problem being considered. In addition to guaranteeing identification of the global solution within arbitrary accuracy, they enable the location of all local and global solutions of the problem. As a result, they can be used to determine the feasibility of a given problem with certainty [1,2,3,4], or to find all solutions of a nonlinear system of equations [13]. They are especially valuable in the study of systems in which the global optimum solution is the only physically meaningful solution, as is the case of the phase equilibrium of non ideal mixtures [16,17,18,19,20]. Traditionally, a major theoretical limitation of these approaches has been their inability to tackle problems with arbitrary nonconvexities. However, the recent development of rigorous convex relaxation techniques for general twice continuously differentiable functions [2,3,4] has greatly expanded the class of problems that can be addressed through deterministic global optimization. These approaches have been incorporated within a branch and bound framework to create the *αBB global optimization algorithm* for twice continuously differentiable problems [3,6,12]. The theoretical basis of the algorithm as well as the efficient search strategies it uses are discussed in this article.

General Framework

The αBB algorithm guarantees finite ϵ-convergence to the global solution of nonlinear programming problems (NLPs) belonging to the general class

$$\begin{cases} \min_{\mathbf{x}} & f(\mathbf{x}) \\ \text{s.t.} & \mathbf{g}(\mathbf{x}) \leq 0 \\ & \mathbf{h}(\mathbf{x}) = 0 \\ & \mathbf{x} \in [\mathbf{x}^L, \mathbf{x}^U], \end{cases} \tag{1}$$

where $f(\mathbf{x})$, $\mathbf{g}(\mathbf{x})$ and $\mathbf{h}(\mathbf{x})$ are continuous twice-differentiable functions.

The solution scheme is based on the generation of a nonincreasing sequence of upper bounds and a nondecreasing sequence of lower bounds on the global solution. The monotonicity of these sequences is ensured through successive partitioning of the search space which enables the construction of increasingly tight relaxations of the problem. The validity of the bounds obtained is of crucial importance in a rigorous global optimization approach. The *upper bounding* step does not present any theoretical difficulties and consists of a local optimization of the nonconvex problem. The *lower bounding* step is a more challenging operation in which the nonconvex problem must be convexified and underestimated in the current subdomain. The strategy adopted dictates the applicability of the algorithm and plays a pivotal role in its performance as it determines the tightness of the lower bounds obtained. The procedure followed in the αBB algorithm is discussed in the next section. Finally, the *branching* step involves the partition of the solution domain with the smallest lower bound on the global optimum solution into a covering set of subdomains. Although this is a simple task, the choice of partition has implications for the rate of convergence of the algorithm and efficient branching rules must be used.

Convexification and Underestimation Strategy

A convex relaxation of problem (1) is obtained by constructing convex underestimators for the nonconvex objective function and inequality constraints and by relaxing the nonlinear equality constraints, replacing them with less stringent linear equality constraints or a set of two convex inequalities. The general convexification/relaxation procedure used is first discussed for the objective function and nonconvex inequalities.

Function Decomposition

A convex underestimator for a twice continuously differentiable function is constructed by following a two-stage procedure. In the first stage, the function is decomposed into a summation of terms of special structure, such as linear, convex, bilinear, trilinear, fractional, fractional trilinear, concave in one variable and

general nonconvex terms. Then, based on the fact that the summation of convex functions results in a convex function, a tailored convex underestimator is used for each different term type. Thus, a twice-differentiable function $F(\mathbf{x})$ defined over the domain $[\mathbf{x}^L, \mathbf{x}^U]$ is written as

$$\begin{aligned} F(\mathbf{x}) = {} & c^\top \mathbf{x} + F_C(\mathbf{x}) + \sum_{i=1}^{bt} b_i x_{B_i,1} x_{B_i,2} \\ & + \sum_{i=1}^{tt} t_i x_{T_i,1} x_{T_i,2} x_{T_i,3} + \sum_{i=1}^{ft} f_i \frac{x_{F_i,1}}{x_{F_i,2}} \\ & + \sum_{i=1}^{ftt} ft_i \frac{x_{FT_i,1} x_{FT_i,2}}{x_{FT_i,3}} + \sum_{i=1}^{uct} F_{UC_i}(x_{UC_i}) \\ & + \sum_{i=1}^{nct} F_{NC_i}(\mathbf{x}), \end{aligned} \tag{2}$$

where c is a scalar vector; $F_C(\mathbf{x})$ is a convex function; bt is the number of bilinear terms, b_i is the coefficient of the ith bilinear term and $x_{B_i,1}$ and $x_{B_i,2}$ are the two variables participating in the bilinear term; tt is the number of trilinear terms, t_i is the coefficient of the ith trilinear term and $x_{T_i,1}$ $x_{T_i,2}$ and $x_{T_i,3}$ are the three variables participating in the trilinear term; ft is the number of fractional terms, f_i is the coefficient of the ith fractional term and $x_{F_i,1}$ and $x_{F_i,2}$ are the two variables participating in the fractional term; ftt is the number of fractional trilinear terms, ft_i is the coefficient of the ith fractional trilinear term and $x_{FT_i,1}$, $x_{FT_i,2}$ and $x_{FT_i,3}$ are the three variables participating in the fractional trilinear term; uct is the number of univariate concave terms, F_{UC_i} is the ith univariate concave term and x_{UC_i} is the variable participating in the univariate concave term; nct is the number of general nonconvex terms and $F_{NC_i}(\mathbf{x})$ is the ith general nonconvex term.

The decomposition phase serves two purposes: it can lead to the construction of a tight underestimator by taking advantage of the special structure of the function and it may reduce the complexity of the underestimation strategy by permitting the treatment of terms which involve a smaller number of variables than the overall nonconvex function. As will become apparent, this is especially important for general nonconvex terms.

Linear and Convex Terms

Any term that has been identified as linear or convex does not need to be modified during the convexification/underestimation procedure.

Bilinear Terms

The bilinear terms can be replaced by their convex envelope [5,15]. A new variable w_B substitutes a bilinear term x_1 x_2 and is bounded by a set of four inequality constraints which depend on the variable bounds.

$$\begin{cases} w_B \geq x_1^L x_2 + x_2^L x_1 - x_1^L x_2^L, \\ w_B \geq x_1^U x_2 + x_2^U x_1 - x_1^U x_2^U, \\ w_B \leq x_1^U x_2 + x_2^L x_1 - x_1^U x_2^L, \\ w_B \leq x_1^L x_2 + x_2^U x_1 - x_1^L x_2^U. \end{cases} \tag{3}$$

Trilinear, Fractional and Fractional Trilinear Terms

For trilinear, fractional and fractional trilinear terms, the convex underestimators proposed in [13] can be used. They are constructed in a fashion similar to the bilinear term underestimators: a new variable replaces the term and a set of inequality constraints provides bounds on this variable. For a trilinear term $x_1 x_2 x_3$, for instance, the substitution variable w_T is subject to

$$\begin{cases} w_T \geq & x_1 x_2^L x_3^L + x_1^L x_2 x_3^L \\ & + x_1^L x_2^L x_3 - 2 x_1^L x_2^L x_3^L, \\ w_T \geq & x_1 x_2^U x_3^U + x_1^U x_2 x_3^L \\ & + x_1^U x_2^L x_3 - x_1^U x_2^L x_3^L - x_1^U x_2^U x_3^U, \\ w_T \geq & x_1 x_2^L x_3^L + x_1^L x_2 x_3^U \\ & + x_1^L x_2^U x_3 - x_1^L x_2^U x_3^U - x_1^L x_2^L x_3^L, \\ w_T \geq & x_1 x_2^U x_3^L + x_1^U x_2 x_3^U \\ & + x_1^L x_2^U x_3 - x_1^L x_2^U x_3^L - x_1^U x_2^U x_3^U, \\ w_T \geq & x_1 x_2^L x_3^U + x_1^L x_2 x_3^L \\ & + x_1^U x_2^L x_3 - x_1^U x_2^L x_3^U - x_1^L x_2^L x_3^L, \\ w_T \geq & x_1 x_2^L x_3^U + x_1^L x_2 x_3^U \\ & + x_1^U x_2^U x_3 - x_1^L x_2^L x_3^U - x_1^U x_2^U x_3^U, \\ w_T \geq & x_1 x_2^U x_3^L + x_1^U x_2 x_3^L \\ & + x_1^L x_2^L x_3 - x_1^U x_2^U x_3^L - x_1^L x_2^L x_3^L, \\ w_T \geq & x_1 x_2^U x_3^U + x_1^U x_2 x_3^U \\ & + x_1^U x_2^U x_3 - 2 x_1^U x_2^U x_3^U. \end{cases} \tag{4}$$

For a fractional term x_1/x_2 with $x_2^L > 0$, the new variable w_F is bounded by

$$w_F \geq \begin{cases} \frac{x_1^L}{x_2} + \frac{x_1}{x_2^U} - \frac{x_1^L}{x_2^U} & \text{if } x_1^L \geq 0, \\ \frac{x_1}{x_2^U} - \frac{x_1^L x_2}{x_2^L x_2^U} + \frac{x_1^L}{x_2^L} & \text{if } x_1^L < 0, \end{cases}$$
$$w_F \geq \begin{cases} \frac{x_1^U}{x_2} + \frac{x_1}{x_2^L} - \frac{x_1^U}{x_2^L} & \text{if } x_1^U \geq 0, \\ \frac{x_1}{x_2^L} - \frac{x_1^U x_2}{x_2^L x_2^U} + \frac{x_1^U}{x_2^U} & \text{if } x_1^U < 0. \end{cases} \quad (5)$$

Finally, for a fractional trilinear term x_1x_2/x_3 with $x_1^L, x_2^L \geq 0$ and $x_3^L > 0$, the substitution variable w_{FT} is subject to

$$\begin{cases} w_{FT} \geq \frac{x_1 x_2^L}{x_3^U} + \frac{x_1^L x_2}{x_3^U} + \frac{x_1^L x_2^L}{x_3} - \frac{2x_1^L x_2^L}{x_3^U}, \\ w_{FT} \geq \frac{x_1 x_2^L}{x_3^U} + \frac{x_1^L x_2}{x_3^L} + \frac{x_1^L x_2^U}{x_3} - \frac{x_1^L x_2^U}{x_3^L} - \frac{x_1^L x_2^L}{x_3^U}, \\ w_{FT} \geq \frac{x_1 x_2^U}{x_3^L} + \frac{x_1^U x_2}{x_3^U} + \frac{x_1^U x_2^L}{x_3} - \frac{x_1^U x_2^L}{x_3^U} - \frac{x_1^U x_2^U}{x_3^L}, \\ w_{FT} \geq \frac{x_1 x_2^U}{x_3^U} + \frac{x_1^U x_2}{x_3^L} + \frac{x_1^L x_2^U}{x_3} - \frac{x_1^L x_2^U}{x_3^U} - \frac{x_1^U x_2^U}{x_3^L}, \\ w_{FT} \geq \frac{x_1 x_2^L}{x_3^U} + \frac{x_1^L x_2}{x_3^L} + \frac{x_1^U x_2^L}{x_3} - \frac{x_1^U x_2^L}{x_3^L} - \frac{x_1^L x_2^L}{x_3^U}, \\ w_{FT} \geq \frac{x_1 x_2^U}{x_3^U} + \frac{x_1^U x_2}{x_3^L} + \frac{x_1^L x_2}{x_3} - \frac{x_1^L x_2^U}{x_3^U} - \frac{x_1^U x_2^U}{x_3^L}, \\ w_{FT} \geq \frac{x_1 x_2^L}{x_3^U} + \frac{x_1^L x_2}{x_3^L} + \frac{x_1^U x_2^L}{x_3} - \frac{x_1^U x_2^L}{x_3^L} - \frac{x_1^L x_2^L}{x_3^U}, \\ w_{FT} \geq \frac{x_1 x_2^U}{x_3^L} + \frac{x_1^U x_2}{x_3^L} + \frac{x_1^U x_2^U}{x_3} - \frac{2x_1^U x_2^U}{x_3^L}. \end{cases} \quad (6)$$

Univariate Concave Terms

For univariate concave terms, the convexification/underestimation procedure does not require the introduction of new variables or constraints: a simple linearization of the term suffices. Thus, a univariate concave term $F_{UC}(x)$ is replaced by the linear term

$$F_{UC}(x^L) + \frac{F_{UC}(x^U) - F_{UC}(x^L)}{x^U - x^L}(x - x^L). \quad (7)$$

General Nonconvex Terms

For a general nonconvex term $F_{NC}(\mathbf{x})$, a convex underestimator $\check{F}_{NC}(\mathbf{x})$ over $[\mathbf{x}^L, \mathbf{x}^U]$ is constructed by subtracting a positive separable quadratic term from $F_{NC}(\mathbf{x})$ [12]:

$$\check{F}_{NC}(\mathbf{x}) = F_{NC}(\mathbf{x}) - \sum_{j=1}^{n} \alpha_j (x_j - x_j^L)(x_j^U - x_j), \quad (8)$$

where n is the number of variables and the α parameters are positive scalars.

The magnitude of the α parameters determines both the quality of the convex underestimator, that is, its tightness, and its convexity. It was shown in [12] that the maximum separation distance, $d_{\max}$, between the nonconvex term $F_{NC}(\mathbf{x})$ and its convex underestimator $\check{F}_{NC}(\mathbf{x})$ is given by

$$d_{\max} = \max_{\mathbf{x}} \left(F_{NC}(\mathbf{x}) - \check{F}_{NC}(\mathbf{x}) \right) = \frac{1}{4} \sum_{j=1}^{n} \alpha_j (x_j^U - x_j^L)^2 . \quad (9)$$

Thus, small α values are needed to construct a tight underestimator. The dependence of the maximum separation distance on the square of the variable ranges is especially important for the convergence proof of the algorithm [12]. Provided that the α values do not increase from a parent node to a child node, relation (9) guarantees that the convex relaxations become increasingly tight as the branch and bound iterations progress and smaller subdomains are generated. In the limit, the convex underestimators match the original functions. As a result, the monotonicity of the lower bound sequence can be ensured.

To meet the convexity requirement of $\check{F}_{NC}(\mathbf{x})$, the positive quadratic term needs to be sufficiently large to overcome the nonconvexity of $F_{NC}(\mathbf{x})$. This is achieved by manipulating the value of the α parameters. Based on the properties of convex functions, a necessary and sufficient condition for the convexity of $\check{F}_{NC}(\mathbf{x})$ is the positive semidefiniteness of the matrix $H_{F_{NC}}(\mathbf{x}) + 2\,\mathrm{diag}(\alpha_j)$ for all $\mathbf{x} \in [\mathbf{x}^L, \mathbf{x}^U]$, where $H_{F_{NC}}(\mathbf{x})$ is the Hessian matrix of the nonconvex term $F_{NC}(\mathbf{x})$. The diagonal matrix $\Delta = \mathrm{diag}(\alpha_j)$ results in a shift in the diagonal elements of the matrix $H_{F_{NC}}(\mathbf{x})$ and is therefore referred to as the *diagonal shift matrix*. The rigorous derivation

of a matrix Δ that satisfies the convexity condition is a difficult matter in the general case, primarily because of the nonlinear dependence of the Hessian matrix on the $\mathbf{x}$ variables. This problem can be alleviated by using *interval arithmetic* to generate an *interval Hessian matrix* $[H_{F_{NC}}]$ such that $H_{F_{NC}}(\mathbf{x}) \in [H_{F_{NC}}]$ for all $\mathbf{x} \in]\mathbf{x}^L, \mathbf{x}^U]$ [1,3,4]. This process allows the formulation of a sufficient convexity condition for the underestimator: if all real symmetric matrices in $[H_{F_{NC}}] + 2\,\mathrm{diag}(\alpha_j)$ are positive semidefinite, then $\breve{F}_{NC}(\mathbf{x})$ is convex over $[\mathbf{x}^L, \mathbf{x}^U]$.

Based on the interval Hessian matrix, a number of methods may be used to automatically and rigorously compute a diagonal shift matrix Δ that guarantees the convexity of $\breve{F}_{NC}(\mathbf{x})$. The first class of techniques generates a *uniform* diagonal shift matrix by equating all the diagonal elements of Δ with a single α value. In the second class of techniques, different α values are used and a *nonuniform* diagonal shift matrix is obtained [1,3].

In the first class of methods, the convexity condition is equivalent to the positive semidefiniteness of all real symmetric matrices in $[H_{F_{NC}}] + 2\,\mathrm{diag}(\alpha)$ and is satisfied by any α parameter such that

$$\alpha \geq \max\left\{0, -\frac{1}{2}\lambda_{\min}\left([H_{F_{NC}}]\right)\right\}, \tag{10}$$

where $\lambda_{\min}\left([H_{F_{NC}}]\right)$ is the minimum eigenvalue of $[H_{F_{NC}}]$ [3,12].

Consider a square symmetric interval Hessian matrix family $[H]$ whose element (ij) is the interval $[\underline{h}_{ij}, \overline{h}_{ij}]$ and whose radius matrix ΔH is defined as $(\Delta H)_{ij} = \frac{(\overline{h}_{ij} - \underline{h}_{ij})}{2}$. A lower bound on the minimum eigenvalue of $[H]$ can be obtained using one of the following methods [1,3,4]:

- Method I.1 — the Gershgorin theorem approach;
- Method I.2a — the E-matrix approach with $E = 0$;
- Method I.2b — the E-matrix approach with $E = \mathrm{diag}(\Delta H)$;
- Method I.3 — Mori–Kokame's approach;
- Method I.4 — the lower bounding Hessian approach;
- Method I.5 — an approach based on the Kharitonov theorem;
- Method I.6 — the Hertz approach.

Method I.1 is an extension of the *Gershgorin theorem* for real matrices to interval matrices. The minimum eigenvalue of $[H]$ is such that

$$\lambda_{\min}([H]) \geq \min_i \left[\underline{h}_{ii} - \sum_{j \neq i} \max\left(\left\|\underline{h}_{ij}\right\|, \left\|\overline{h}_{ij}\right\|\right)\right].$$

Methods I.2a and I.2b are a generalization of the results presented in [8,23]. It requires the computation of the modified midpoint matrix $\widetilde{H}_M$ such that $(\widetilde{H}_M)_{ij} = \frac{(\overline{h}_{ij} + \underline{h}_{ij})}{2}$ for $i \neq j$ and $(\widetilde{H}_M)_{ii} = 0$, as well as the computation of the modified radius matrix $\widetilde{\Delta H}$ such that $(\widetilde{\Delta H})_{ij} = \frac{(\overline{h}_{ij} - \underline{h}_{ij})}{2}$ for $i \neq j$ and $(\widetilde{\Delta H})_{ii} = \underline{h}_{ij}$. Given an arbitrary real symmetric matrix E, the minimum eigenvalue of the interval Hessian matrix $[H]$ is such that

$$\lambda_{\min}([H]) \geq \lambda_{\min}\left(\widetilde{H}_M + E\right) - \rho\left(\widetilde{\Delta H} + \|E\|\right),$$

where $\rho(M)$ denotes the spectral radius of the real matrix M. In practice, two E-matrices have been used: $E = 0$ (Method I.2a) and $E = \Delta H$ (Method I.2b).

Method I.3 is based on a result presented in [21], which uses the lower vertex matrix H, such that $(H)_{ij} = h_{ij}$, and the upper vertex matrix $\overline{H}$, such $(\overline{H})_{ij} = \overline{h}_{ij}$. The minimum eigenvalue of $[H]$ is such that

$$\lambda_{\min}([H]) \geq \lambda_{\min}(\underline{H}) - \rho(\overline{H} - \underline{H}).$$

Method I.4 uses a *lower bounding Hessian* of the interval Hessian matrix. Such a matrix is defined in [24] as a real symmetric matrix whose minimum eigenvalue is smaller than the minimum eigenvalue of any real symmetric matrix in the interval Hessian family. It therefore suffices to compute the minimum eigenvalue of this real matrix to obtain the desired lower bound. A lower bounding Hessian $L = (l_{ij})$ can be constructed from the following rule:

$$l_{ij} = \begin{cases} \underline{h}_{ii} + \sum_{k \neq i} \frac{\underline{h}_{ik} - \overline{h}_{ik}}{2}, & i = j, \\ \frac{\underline{h}_{ij} + \overline{h}_{ij}}{2}, & i \neq j. \end{cases}$$

Method I.5 is based on the *Kharitonov theorem* [11] which, by extension, gives a lower bound on the minimum eigenvalue of an interval Hessian matrix family [2]. First, the corresponding characteristic polynomial family must be derived

$$[K] = [\underline{c}_0, \overline{c}_0] + [\underline{c}_1, \overline{c}_1]\lambda + [\underline{c}_2, \overline{c}_2]\lambda^2 + [\underline{c}_3, \overline{c}_3]\lambda^3 + [\underline{c}_4, \overline{c}_4]\lambda^4 + [\underline{c}_5, \overline{c}_5]\lambda^5 + \cdots,$$

where the coefficients of λ depend on the elements of the interval Hessian matrix $[H]$. A lower bound on the roots of this polynomial can then obtained by calculating the minimum roots of only four real polynomials. The appropriate bounding polynomials are the Kharitonov polynomials

$$\begin{cases} K_1 = & \underline{c}_0 + \underline{c}_1\lambda + \overline{c}_2\lambda^2 + \overline{c}_3\lambda^3 \\ & + \underline{c}_4\lambda^4 + \underline{c}_5\lambda^5 + \cdots, \\ K_2 = & \overline{c}_0 + \overline{c}_1\lambda + \underline{c}_2\lambda^2 + \underline{c}_3\lambda^3 \\ & + \overline{c}_4\lambda^4 + \overline{c}_5\lambda^5 + \cdots, \\ K_3 = & \overline{c}_0 + \underline{c}_1\lambda + \underline{c}_2\lambda^2 + \overline{c}_3\lambda^3 \\ & + \overline{c}_4\lambda^4 + \underline{c}_5\lambda^5 + \cdots, \\ K_4 = & \underline{c}_0 + \overline{c}_1\lambda + \overline{c}_2\lambda^2 + \underline{c}_3\lambda^3 \\ & + \underline{c}_4\lambda^4 + \overline{c}_5\lambda^5 + \cdots. \end{cases}$$

Method I.6 allows the computation of the exact minimum eigenvalue of the family of symmetric matrices represented by the interval Hessian matrix. It requires the construction of 2^{n-1} vertex matrices H^k of the interval matrix $[H]$ as defined by

$$(H^k)_{ij} = \begin{cases} \underline{h}_{ii} & \text{if } i = j, \\ \underline{h}_{ij} & \text{if } u_i u_j \geq 0,\ i \neq j, \\ \overline{h}_{ij} & \text{if } u_i u_j < 0,\ i \neq j, \end{cases}$$

where all possible combinations of the signs of the arbitrary scalars u_i and u_j are enumerated. It was shown in [4,10] that the lowest minimum eigenvalue from this set of real matrices is the minimum eigenvalue of the interval matrix.

Three rigorous techniques for the generation of a non uniform shift matrix Δ can be used [1,3]:

- Method II.1a — the scaled Gershgorin theorem approach with scaling vector $\mathbf{d} = \underline{1}$;
- Method II.1b — the scaled Gershgorin theorem approach with scaling vector $\mathbf{d} = \mathbf{x}^U - \mathbf{x}^L$;
- Method II.2 — the H-matrix approach;
- Method II.3 — an approach based on the minimization of the maximum separation distance.

The main advantage of these techniques is that resorting to a different value of the α parameter for each variable may lead to tighter underestimators by taking into account the individual contribution of each variable to the overall nonconvexity of the term being considered. In the case of a uniform diagonal shift, the worst contribution is uniformly assigned to all variables.

Methods II.1a and II.1b bear resemblance with the Gershgorin theorem used for Method I.1. In the present case, however, each row is considered independently and the ith element of the diagonal shift matrix, α_i, is the maximum of zero and

$$-\frac{1}{2}\left(\underline{h}_{ii} - \sum_{j \neq i} \max\left\{\left\|\underline{h}_{ij}\right\|, \left\|\overline{h}_{ij}\right\|\right\} \frac{d_j}{d_i}\right),$$

where $\mathbf{d}$ is an arbitrary positive vector. In practice, $\mathbf{d} = \underline{1}$ (Method II.1a) and $\mathbf{d} = \mathbf{x}^U - \mathbf{x}^L$ (Method II.1b) have been used. The latter choice of scaling often helps to reduce the maximum separation distance between the nonconvex term and its underestimator by assigning smaller α values to variables with a larger range.

Method II.2 is an iterative method based on the properties of H-matrices: a square interval matrix that has the H-matrix property is regular and does not have 0 as an eigenvalue [22]. In order to determine whether a square interval matrix $[H]$ is an H-matrix, its comparison matrix $\langle H \rangle$ must first be defined. For $i \neq j$, the off-diagonal element $(\langle H \rangle)_{ij}$ of the comparison matrix is given by $-\max\{\left\|\underline{h}_{ij}\right\|, \left\|\overline{h}_{ij}\right\|\}$. A diagonal element $(\langle H \rangle)_{ii}$ of the comparison matrix is given by

$$\begin{cases} 0, & 0 \in [\underline{h}_{ii}, \overline{h}_{ii}], \\ \min\left\{\left\|\underline{h}_{ii}\right\|, \left\|\overline{h}_{ii}\right\|\right\}, & 0 \notin [\underline{h}_{ii}, \overline{h}_{ii}]. \end{cases}$$

A real matrix such as $\langle H \rangle$ is an M-matrix if all its off-diagonal elements are nonpositive – this is always true for $\langle H \rangle$ – *and* if there exists a real positive vector $\mathbf{u}$ such that $\langle H \rangle \mathbf{u} > 0$. The interval matrix $[H]$ is an H-matrix if its comparison matrix $\langle H \rangle$ is an M-matrix. Method II.2 follows an iterative procedure to construct a nonuniform diagonal shift matrix Δ such that $[H] + 2\,\Delta$ is an H-matrix whose modified midpoint matrix is positive definite. If these conditions are met, the diagonal elements of the shift matrix are guaranteed to lead to the construction of a *convex* underestimator for the nonconvex term. The initial guess chosen for Δ is the uniform diagonal shift matrix given by Method I.2.

Method II.3 aims to generate a non uniform diagonal shift matrix which minimizes the maximum separation distance between the nonconvex term and its underestimator. For this purpose, the following semidefinite programming problem is solved using an interior

point method [25]:

$$\begin{cases} \min_{\alpha_i} & (\mathbf{x}^U - \mathbf{x}^L)^\top \Delta (\mathbf{x}^U - \mathbf{x}^L) \\ \text{s.t.} & L + 2\,\mathrm{diag}(\alpha_i) \geq 0 \\ & \alpha_i \geq 0, \quad \forall i, \end{cases}$$

where L is the lower bounding Hessian matrix defined in Method I.4. Because this approach is based on the lower bounding Hessian matrix rather than the exact $\mathbf{x}$-dependent Hessian matrix, the solution found does not correspond to the smallest achievable maximum separation distance, but can be expected to be smaller than when Method I.4 is used.

A comparative study [1,3] of all the methods available for the generation of a diagonal shift matrix found that Methods II.1a, II.1b and II.3 usually give the tightest underestimators. However, Method II.3 is computationally intensive and therefore results in poorer convergence rates than Methods II.1a and II.1b. Since the least computationally expensive techniques for the generation of the diagonal shift matrix, Methods I.1, II.1a and II.1b, are of order $O(n^2)$, the decomposition of the nonconvex terms into a summation of terms involving a smaller number of variables may have a significant impact on the performance of the algorithm.

Overall Convexification/Relaxation Strategy

Based on the rigorous convexification/underestimation schemes for bilinear, trilinear, fractional, fractional trilinear, univariate concave and general nonconvex terms, the overall convex underestimator $\check{F}(\mathbf{x}, \mathbf{w})$ for a twice continuously differentiable function $F(\mathbf{x})$ decomposed according to (2) is

$$\begin{aligned} \check{F}(\mathbf{x}, \mathbf{w}) &= c^\top \mathbf{x} + F_C(\mathbf{x}) + \sum_{i=1}^{bt} b_i w_{B_i} \\ &+ \sum_{i=1}^{tt} t_i w_{T_i} + \sum_{i=1}^{ft} f_i w_{F_i} + \sum_{i=1}^{ftt} ft_i w_{FT_i} \\ &+ \sum_{i=1}^{uct} \Bigg(F_{UC_i}(x^L_{UC_i}) \\ &+ \frac{F_{UC_i}(x^U_{UC_i}) - F_{UC_i}(x^L_{UC_i})}{x^U_{UC_i} - x^L_{UC_i}} (x_{UC_i} - x^L_{UC_i}) \Bigg) \\ &+ \sum_{i=1}^{nct} \left(F_{NC_i}(\mathbf{x}) - \sum_{j=1}^{n} \alpha_{ij} (x_j - x^L_j)(x^U_j - x_j) \right), \end{aligned} \tag{11}$$

where the notation is as defined for (2). The introduction of the new variables w_{B_i}, w_{T_i}, w_{F_i} and w_{FT_i} is accompanied by the addition of convex inequalities of the type given in (3), (4), (5) and (6). For the trilinear, fractional and fractional trilinear terms, the specific form of these equations depends on the sign of the term coefficients and variable bounds.

The form given by (11) can be used to construct convex underestimators for the objective function and inequality constraints.

Equality Constraints

For nonlinear equality constraints, two different convexification/relaxation schemes are used, depending on the mathematical structure of the function. If the equality $h(\mathbf{x} = 0$ involves only linear, bilinear, trilinear, fractional and fractional trilinear terms, it is first decomposed into the equivalent equality constraint

$$\begin{aligned} c^\top \mathbf{x} + \sum_{i=1}^{bt} b_i x_{B_i,1} x_{B_i,2} + \sum_{i=1}^{tt} t_i x_{T_i,1} x_{T_i,2} x_{T_i,3} \\ + \sum_{i=1}^{ft} f_i \frac{x_{F_i,1}}{x_{F_i,2}} + \sum_{i=1}^{ftt} ft_i \frac{x_{FT_i,1} x_{FT_i,2}}{x_{FT_i,3}} = 0, \end{aligned} \tag{12}$$

where the notation is as previously defined. (12) is then replaced by

$$\begin{aligned} c^\top \mathbf{x} + \sum_{i=1}^{bt} b_i w_{B_i} + \sum_{i=1}^{tt} t_i w_{T_i} \\ + \sum_{i=1}^{ft} f_i w_{F_i} + \sum_{i=1}^{ftt} ft_i w_{FT_i} = 0, \end{aligned} \tag{13}$$

with the addition of convex inequalities of the type given by (3), (4), (5) and (6). If the nonlinear equality contains at least one convex, univariate concave or general nonconvex term, the convexification/relaxation strategy must first transform the equality constraint $h(\mathbf{x})$ into a set of two equivalent inequality constraints

$$\begin{cases} & h(\mathbf{x}) \leq 0 \\ - & h(\mathbf{x}) \leq 0, \end{cases} \tag{14}$$

which can then be convexified and underestimated independently using (11).

The transformation of a nonconvex twice-differentiable problem into a convex lower bounding problem

described in this section allows the generation of valid and increasingly tight lower bounds on the global optimum solution.

Branching Variable Selection

Once upper and lower bounds have been obtained for all the existing nodes of the branch and bound tree, the region with the smallest lower bound is selected for branching. The partitioning of the solution space can have a significant effect on the quality of the lower bounds obtained because of the strong dependence of the convex underestimators described by (3)–(8) on the variable bounds. It is therefore important to identify the variables which most contribute to the separation between the original problem and the convex lower bounding problem at the current node. Several branching variable selection criteria have been designed for this purpose [1].

Least Reduced Axis Rule

The first strategy leads to the selection of the variable that has least been branched on to arrive at the current node. It is characterized by the largest ratio

$$\frac{x_i^U - x_i^L}{x_{i,0}^U - x_{i,0}^L},$$

where $x_{i,0}^L$ and $x_{i,0}^U$ are the lower and upper bounds on variable x_i at the first node of the branch and bound tree and x_i^L and x_i^U are the current lower and upper bounds on variable x_i.

The main disadvantage of this simple rule is that it does not account for the specificities of the participation of each variable in the problem and therefore cannot accurately identify the critical variables that determine the quality of the underestimators.

Term Measure

A more sophisticated rule is based on the computation of a term measure μ_j^t for term t_j defined as

$$\mu_j^t = t_j(\mathbf{x}^*) - \check{t}_j(\mathbf{x}^*, \mathbf{w}^*), \tag{15}$$

where $t_j(\mathbf{x})$ is a bilinear, trilinear, fractional, fractional trilinear, univariate concave or general nonconvex term, $\check{t}_j(\mathbf{x}, \mathbf{w})$ is the corresponding convex underestimator, $\mathbf{x}^*$ is the solution vector corresponding to the minimum of the convex lower bounding problem, and $\mathbf{w}^*$ is the solution vector for the new variables at the minimum of the convex lower bounding problem. One of the variables participating in the term with the largest measure μ_j^t is selected for branching.

Variable Measure

A third strategy is based on a variable measure μ_i^v which is computed from the term measures μ_j^t. For variable x_i, this measure is

$$\mu_i^v = \sum_{j \in T_i} \mu_j^t, \tag{16}$$

where T_i is the set of terms in which x_i participates. The variable with the largest measure μ_i^v is branched on.

Variable Bound Updates

The effect of the variable bounds on the convexification/relaxation procedure motivates the tightening of the variable bounds. However, the trade-off between tight underestimators generated at a large computational cost and looser underestimators obtained more rapidly must be taken into account when designing a variable bound update strategy. For this reason, one of several approaches can be adopted, depending on the degree of nonconvexity of the problem [1,3]:

- variable bound updates
  - at the beginning of the algorithmic procedure only; or
  - at each iteration;
- bound updates
  - for all variables in the problem; or
  - bound updates for those variables that most affect the quality of the lower bounds as measured by the variable measure μ_i^v.

Two different techniques can be used to tighten the variable bounds. The first is based on the generation and solution of a series of convex optimization problems while the second is an iterative procedure relying on the interval evaluation of the functions in the nonconvex NLP.

Optimization-Based Approach

In the optimization approach, a new lower or upper bound for variable x_i is obtained by solving the convex

problem

$$\begin{cases} \min_{\mathbf{x},\mathbf{w}} \text{ or } \max_{\mathbf{x},\mathbf{w}} & x_i \\ \text{s.t.} & \breve{f}(\mathbf{x},\mathbf{w}) \le \overline{f}^* \\ & \breve{\mathbf{g}}(\mathbf{x},\mathbf{w}) \le 0 \\ & \breve{\mathbf{h}}_N^+(\mathbf{x},\mathbf{w}) \le 0 \\ & \breve{\mathbf{h}}_N^-(\mathbf{x},\mathbf{w}) \le 0 \\ & \breve{\mathbf{h}}_L(\mathbf{x},\mathbf{w}) = 0 \\ & \mathbf{n}(\mathbf{x},\mathbf{w}) \le 0 \\ & \mathbf{x} \in [\mathbf{x}^L, \mathbf{x}^U], \\ & \mathbf{w} \in [\mathbf{w}^L, \mathbf{w}^U], \end{cases} \tag{17}$$

where $\breve{p}(\mathbf{x},\mathbf{w})$ denotes the convex underestimator of function $p(\mathbf{x})$ as defined in (11), $\overline{f}^*$ denotes the current best upper bound on the global optimum solution, $\mathbf{h}_L(\mathbf{x})$ denotes the set of equality constraints which involve only linear, bilinear, trilinear, fractional and fractional trilinear terms, $\mathbf{h}_N^+(\mathbf{x})$ denotes the set of equality constraints that involve other term types and $\mathbf{h}_N^-(\mathbf{x})$ denotes the negative of that set, $\mathbf{n}(\mathbf{x},\mathbf{w})$ denotes the set of additional constraints that arise from the underestimation of bilinear, trilinear, fractional and fractional trilinear terms, and $\mathbf{w}$ is the corresponding set of new variables.

Interval-Based Approach

In the interval-based approach, an iterative procedure is followed for each variable whose bounds are to be updated. The original functions in the problem are used without any transformations. An inequality constraint $g(\mathbf{x}) \le 0$ is infeasible in the domain $[\mathbf{x}^L, \mathbf{x}^U]$ if its range $[g^L, g^U]$, computed so that $g(\mathbf{x}) \in [g^L, g^U]\ \forall\ \mathbf{x} \in [\mathbf{x}^L, \mathbf{x}^U[$, is such that $g^L > 0$. Similarly, an equality constraint $h(\mathbf{x}) = 0$ is infeasible in this domain if its range $[h^L, h^U]$, computed so that $h(\mathbf{x}) \in [h^L, h^U]$, $\forall\ \mathbf{x} \in [\mathbf{x}^L, \mathbf{x}^U]$, is such that $0 \notin [h^L, h^U]$. The variable bounds are updated based on the feasibility of the constraints in the original problem and the additional constraint that the objective function should be less than or equal to the current best upper bound $\overline{f}^*$. The feasible region is therefore defined as

$$F = \left\{ \mathbf{x}: \quad \begin{array}{c} \mathbf{g}(\mathbf{x}) \le 0, \mathbf{h}(\mathbf{x}) = 0, \\ f(\mathbf{x}) \le \overline{f}^*, \mathbf{x} \in [\mathbf{x}^L, \mathbf{x}^U] \end{array} \right\}.$$

The lower (upper) bound on variable $x_i \in [x_i^L, x_i^U]$ is updated as follows:

```
PROCEDURE interval-based bound update()
    Set initial bounds L = x_i^L and U = x_i^U;
    Set iteration counter k = 0;
    Set maximum number of iterations K;
    DO k < K
        Compute midpoint M = (U + L)/2;
        Set left region {x ∈ F : x_i ∈ [L, M]};
        Set right region {x ∈ F : x_i ∈ [L, M]};
        Test interval feasibility of left (right region);
        IF feasible,
            Set U = M  (L = M);
        ELSE,
            Test interval feasibility of right (left)
            region;
            IF feasible,
                Set L = M  (U = M);
            ELSE,
                Set L = U  (U = L);
                Set U = x_i^U  (U = x_i^L)
        IF k = 0 and L = x_i^U  (U = x_i^L),
            RETURN(infeasible node);
        Set k = k + 1;
    OD;
    RETURN(x_i^L = L  (x_i^U = U));
END interval-based bound update;
```

Interval-based bound update procedure

In general, the interval-based bound update strategy is less computationally expensive than the optimization-based approach. However, at the beginning of the branch and bound search, when the bound updates are most critical and the variable ranges are widest, the overestimations inherent in interval computations often lead to looser updated bounds in the interval-based approach than in the optimization-based technique.

Algorithmic Procedure

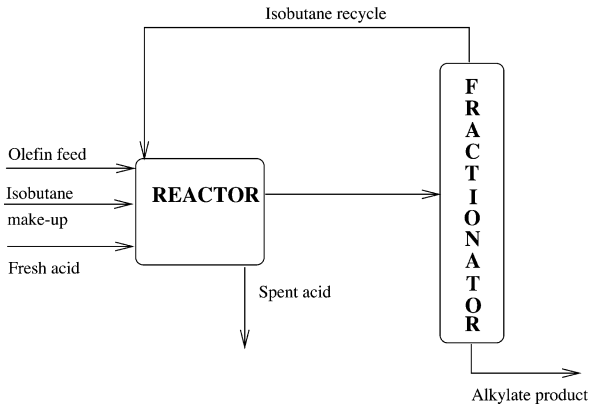
Based on the developments presented in previous sections, the procedure for the αBB algorithm can be summarized by the following pseudocode:

```
PROCEDURE αBB algorithm()
    Decompose functions in problem;
    Set tolerance ϵ;
    Set f* = f^0 = −∞ and f̄* = f̄^0 = +∞;
    Initialize list of lower bounds {f^0};
    DO f̄* − f* > ϵ
        Select node k with smallest lower bound, f^k,
        from list of lower bounds;
        Set f* = f^k;
        (Optional) Update variable bounds for cur-
        rent node using optimization or interval
        approach;
        Select branching variable;
        Partition to create new nodes;
        DO for each new node i
            Generate convex lower bounding NLP
                Introduce new variables, constraints;
                Linearize univariate concave terms;
                Compute interval Hessian matrices;
                Compute α values;
            Find solution f^i of convex lower bound-
            ing NLP;
            IF infeasible or f^i > f̄* + ϵ
                Fathom node;
            ELSE
                Add f^i to list of lower bounds;
                Find a solution f̄^i of nonconvex NLP;
                IF f̄^i < f̄*
                    Set f̄* = f̄^i;
        OD;
    OD;
    RETURN(f̄* and variables values at correspond-
    ing node);
END αBB algorithm;
```

A pseudocode for the αBB algorithm

Computational Experience

Significant computational experience with the αBB algorithm has been acquired through the solution of a wide variety of problems involving different types of nonconvexities and up to 16000 variables [1,2,3,4,6,9,12]. These include problems such as pooling/blending, design of reactor networks, design of batch plants under uncertainty [9], stability studies belonging to the class of generalized geometric programming problems, characterization of phase-equilibrium using activity coefficient models, identification of stable molecular conformations and the determination of all solutions of systems of nonlinear equations.



αBB Algorithm, Figure 1
Simplified alkylation process flowsheet

In order to illustrate the performance of the algorithm and the importance of variable bound updates, a medium-size example is presented. The objective is to maximize the profit for the simplified alkylation process presented in [7] and shown in Fig. 1.

An olefin feed (100% butene), a pure isobutane recycle and a 100% isobutane make up stream are introduced in a reactor together with an acid catalyst. The reactor product stream is then passed through a fractionator where the isobutane and the alkylate product are separated. The spent acid is also removed from the reactor. The formulation used here includes 7 variables and 16 constraints, 12 of which are nonlinear. The variables are defined as follows: x_1 is the olefin feed rate in barrels per day; x_2 is the acid addition rate in thousands of pounds per day; x_3 is the alkylate yield in barrels per day; x_4 is the acid strength (weight percent); x_5 is the motor octane number; x_6 is the external isobutane-to-olefin ratio; x_7 is the F-4 performance number. The profit maximization problem is then expressed as:

$$\begin{aligned}\text{Profit} = -\min(&1.715x_1 + 0.035x_1x_6 \\ &+ 4.0565x_3 + 10.0x_2 - 0.063x_3x_5)\end{aligned}$$

subject to:

$$
\begin{aligned}
&0.0059553571x_6^2x_1 + 0.88392857x_3 \\
&- 0.1175625x_6x_1 - x_1 \leq 0, \\
&1.1088x_1 + 0.1303533x_1x_6 \\
&- 0.0066033x_1x_6^2 - x_3 \leq 0, \\
&6.66173269x_6^2 + 172.39878x_5 \\
&- 56.596669x_4 - 191.20592x_6 \leq 10000, \\
&1.08702x_6 + 0.32175x_4 - 0.03762x_6^2 \\
&- x_5 \leq -56.85075, \\
&0.006198x_7x_4x_3 + 2462.3121x_2 \\
&- 25.125634x_2x_4 - x_3x_4 \leq 0, \\
&161.18996x_3x_4 + 5000.0x_2x_4 \\
&- 489510.0x_2 - x_3x_4x_7 \leq 0, \\
&0.33x_7 - x_5 + 44.333333 \leq 0, \\
&0.022556x_5 - 0.007595x_7 \leq 1, \\
&0.00061x_3 - 0.0005x_1 \leq 1, \\
&0.819672x_1 - x_3 + 0.819672 \leq 0, \\
&24500.0x_2 - 250.0x_2x_4 - x_3x_4 \leq 0, \\
&1020.4082x_4x_2 + 1.2244898x_3x_4 \\
&- 100000x_2 \leq 0, \\
&6.25x_1x_6 + 6.25x_1 - 7.625x_3 \leq 100000, \\
&1.22x_3 - x_6x_1 - x_1 + 1 \leq 0, \\
&1500 \leq x_1 \leq 2000, \\
&1 \leq x_2 \leq 120, \\
&3000 \leq x_3 \leq 3500, \\
&85 \leq x_4 \leq 93, \\
&90 \leq x_5 \leq 95, \\
&3 \leq x_6 \leq 12, \\
&145 \leq x_7 \leq 162.
\end{aligned}
$$

The maximum profit is \$1772.77 per day, and the optimal variable values are $x_1^* = 1698.18$, $x_2^* = 53.66$, $x_3^* = 3031.30$, $x_4^* = 90.11$, $x_5^* = 95.00$, $x_6^* = 10.50$, $x_7^* = 153.53$. In this example, variable bound tightening is performed using the optimization-based approach. An update of all the variable bounds therefore involves the solution of 14 convex NLPs. The computational cost is significant and may not always be justified by the corresponding decrease in number of iterations. Two extreme tightening strategies were used to illustrate this trade-off: an update of all variable bounds at the onset of the algorithm only ('Single Up'), or an update of all bounds at each iteration of the αBB algorithm ('One Up/Iter'). An intermediate strategy might involve bound updates for those variables that affect the underestimators most significantly or bound updates at only a few levels of the branch and bound tree. The results of runs performed on an HP9000/730 are summarized in the table below. t_U denotes the percentage of CPU time devoted to the construction of the convex underestimating problem.

Although the approach relying most heavily on variable bound updates results in tighter underestimators, and hence a smaller number of iterations, the time requirements for each iteration are significantly larger than when no bounds updates are performed. Thus, the overall CPU requirements often increase when all variable bounds are updated at each iteration.

	Single up			One Up/Iter		
Meth	Iter.	CPU sec.	t_U (%)	Iter.	CPU sec.	t_U (%)
I.1	74	37.5	0.5	31	41.6	0.0
I.2a	61	30.6	1.6	25	37.2	0.2
I.2b	61	29.2	1.0	25	35.4	0.1
I.3	69	32.8	1.9	25	31.5	0.2
I.4	61	31.6	1.4	25	33.1	0.2
I.5	61	32.8	12.3	25	36.7	1.7
I.6	59	32.9	1.4	25	32.8	0.5
II.1a	56	24.9	0.3	30	36.5	0.3
II.1b	38	13.6	1.7	17	19.9	0.5
II.2	62	32.7	0.6	25	34.5	0.3
II.3	54	21.8	16.7	23	30.4	5.0

Alkylation process design results

In order to determine the best technique for the construction of convex underestimators, the percentage of computational effort dedicated to this purpose, t_U, is tracked. As can be seen in the above table, the generation of the convex lower bounding does not consume a large share of the computational cost, regardless of the method. It is, however, significantly larger for Methods I.5 and II.3 as they require the solution of a polynomial and a semidefinite programming problem respectively. t_U decreases when bound updates are performed at each iteration as a large amount of time is

spent solving the bound updates problems. In this example, the scaled Gershgorin approach with $d_i = (x_i^U - x_i^L)$ (Method II.1b) gives the best results both in terms of number of iterations and CPU time.

Conclusions

The αBB algorithm is guaranteed to identify the global optimum solution of problems belonging to the broad class of twice continuously differentiable NLPs. It is a branch and bound approach based on a rigorous convex relaxation strategy, which involves the decomposition of the functions into a sum of terms with special mathematical structure and the construction of different convex underestimators for each class of term. In particular, the treatment of general nonconvex terms requires the analysis of their Hessian matrix through interval arithmetic. Efficient branching and variable bound update strategies can be used to enhance the performance of the algorithm.

See also

- ▶ Bisection Global Optimization Methods
- ▶ Continuous Global Optimization: Applications
- ▶ Continuous Global Optimization: Models, Algorithms and Software
- ▶ Convex Envelopes in Optimization Problems
- ▶ D.C. Programming
- ▶ Differential Equations and Global Optimization
- ▶ DIRECT Global Optimization Algorithm
- ▶ Eigenvalue Enclosures for Ordinary Differential Equations
- ▶ Generalized Primal-relaxed Dual Approach
- ▶ Global Optimization Based on Statistical Models
- ▶ Global Optimization in Batch Design Under Uncertainty
- ▶ Global Optimization in Binary Star Astronomy
- ▶ Global Optimization in Generalized Geometric Programming
- ▶ Global Optimization Methods for Systems of Nonlinear Equations
- ▶ Global Optimization in Phase and Chemical Reaction Equilibrium
- ▶ Global Optimization Using Space Filling
- ▶ Hemivariational Inequalities: Eigenvalue Problems
- ▶ Interval Analysis: Eigenvalue Bounds of Interval Matrices
- ▶ Interval Global Optimization
- ▶ MINLP: Branch and Bound Global Optimization Algorithm
- ▶ MINLP: Global Optimization with αBB
- ▶ Reformulation-linearization Methods for Global Optimization
- ▶ Reverse Convex Optimization
- ▶ Semidefinite Programming and Determinant Maximization
- ▶ Smooth Nonlinear Nonconvex Optimization
- ▶ Topology of Global Optimization

References

1. Adjiman CS, Androulakis IP, Floudas CA (1998) A global optimization method, αBB, for general twice-differentiable constrained NLPs – II. Implementation and computational results. Comput Chem Eng 22:1159
2. Adjiman CS, Androulakis IP, Maranas CD, Floudas CA (1996) A global optimization method, αBB, for process design. Comput Chem Eng 20:S419–S424
3. Adjiman CS, Dallwig S, Floudas CA, Neumaier A (1998) A global optimization method, αBB, for general twice-differentiable constrained NLPs – I. Theoretical advances. Comput Chem Eng 22:1137
4. Adjiman CS, Floudas CA (1996) Rigorous convex underestimators for twice-differentiable problems. J Global Optim 9:23–40
5. Al-Khayyal FA, Falk JE (1983) Jointly constrained biconvex programming. Math Oper Res 8:273–286
6. Androulakis IP, Maranas CD, Floudas CA (1995) αBB: A global optimization method for general constrained nonconvex problems. J Global Optim 7:337–363
7. Bracken J, McCormick GP (1968) Selected applications of nonlinear programming. Wiley, New York
8. Deif AS (1991) The interval eigenvalue problem. Z Angew Math Mechanics 71:61–64
9. Harding ST, Floudas CA (1997) Global optimization in multiproduct and multipurpose batch design under uncertainty. Industr Eng Chem Res 36:1644–1664
10. Hertz D (1992) The extreme eigenvalues and stability of real symmetric interval matrices. IEEE Trans Autom Control 37:532–535
11. Kharitonov VL (1979) Asymptotic stability of an equilibrium position of a family of systems of linear differential equations. Differential Eq:1483–1485
12. Maranas CD, Floudas CA (1994) Global minimum potential energy conformations of small molecules. J Global Optim 4:135–170
13. Maranas CD, Floudas CA (1995) Finding all solutions of nonlinearly constrained systems of equations. J Global Optim 7:143–182

14. Maranas CD, Floudas CA (1997) Global optimization in generalized geometric programming. Comput Chem Eng 21:351–370
15. McCormick GP (1976) Computability of global solutions to factorable nonconvex programs: part I – Convex underestimating problems. Math Program 10:147–175
16. McDonald CM, Floudas CA (1994) Decomposition based and branch and bound global optimization approaches for the phase equilibrium problem. J Global Optim 5:205–251
17. McDonald CM, Floudas CA (1995) Global optimization and analysis for the Gibbs free energy function for the UNIFAC, Wilson, and ASOG equations. Industr Eng Chem Res 34:1674–1687
18. McDonald CM, Floudas CA (1995) Global optimization for the phase and chemical equilibrium problem: Application to the NRTL equation. Comput Chem Eng 19:1111–1141
19. McDonald CM, Floudas CA (1995) Global optimization for the phase stability problem. AIChE J 41:1798–1814
20. McDonald CM, Floudas CA (1997) GLOPEQ: A new computational tool for the phase and chemical equilibrium problem. Comput Chem Eng 21:1–23
21. Mori T, Kokame H (1994) Eigenvalue bounds for a certain class of interval matrices. IEICE Trans Fundam E77-A:1707–1709
22. Neumaier A (1992) An optimality criterion for global quadratic optimization. J Global Optim 2:201–208
23. Rohn J (1996) Bounds on eigenvalues of interval matrices. Techn Report Inst Computer Sci Acad Sci Prague 688
24. Stephens C (1997) Interval and bounding Hessians. In: Bonze IM et al (eds) Developments in Global Optimization. Kluwer, Dordrecht, pp 109–199
25. Vandenberghe L, Boyd S (1996) Semidefinite programming. SIAM Rev 38:49–95

Alternative Set Theory

AST

PETR VOPĚNKA, KATEŘINA TRLIFAJOVÁ
Charles University, Prague, Czech Republic

MSC2000: 03E70, 03H05, 91B16

Article Outline

Keywords

Sets; Semisets; Infinity; Countability; Continuum; Topology; Indiscernibility; Motion; Utility theory

Alternative set theory has been created and, together with his colleagues at Charles University, developed by P. Vopěnka since the 1970s. In agreement with Husserl's phenomenology, he based his theory on the natural world and the human view thereof.

The most important for any set theory is the way it treats infinity. A different approach to infinity forms the key difference between AST and classical set theories based on the *Cantor set theory* (CST). Cantor's approach led to the creation of a rigid, abstract world with an enormous scale of infinite cardinalities while Vopěnka's infinity, based on the notion of horizon, is more natural and acceptable.

Another source of inspiration were nonstandard models of Peano arithmetics with infinitely large (nonstandard) numbers. The way to build them in AST is easy and natural.

The basic references are [9,10,11].

Classes, Sets and Semisets

AST, as well as CST, builds on notions of 'set', 'class', 'element of a set' and, in addition, introduces the notion of 'semiset'. A *class* is the most general notion used for any collection of distinct objects. *Sets* are such classes that are so clearly defined and clean-cut that their elements could be, if necessary, included in a list. *Semisets* are classes which are not sets, because their borders are vague, however, they are parts of sets. For example, all living people in the world form a class—some are being born, some are dying, we do not know where all of them are. The citizens of Prague, registered at the given moment in the register, form a set. However, all the beautiful women in Prague or brave men in Prague

form a semiset, since it is not clear who belongs to this collection and who not.

In the real world, we may find many other semisets. Almost each property defines a semiset of objects, e. g., people who are big, happy or sick. Many properties are naturally connected with a vagueness. Also, what we see and perceive can be vague and limited by a horizon. Objects described in this way may form a semiset, e. g. flowers I can see in the blooming meadow, all my friends, sounds I can hear.

Infinity

This interpretation differs from the normal one and corresponds more to the etymological origin of the word infinity. We will call *finite* those classes any part of which is surveyable and forms a set. Any finite class is a set.

$$\mathrm{Fin}(X) \Leftrightarrow (\forall Y)(Y \subseteq X \implies \mathrm{Set}(Y)).$$

On the other side, *infinite classes* include ungrasped parts, semisets. This phenomenon may occur also when watching large sets in the case when it is not possible to capture them clearly as a whole.

There are two different forms of infinity traditionally called denumerability and continuum.

A countable (denumerable) class, in a way, represents a road towards the horizon. Its beginning is clear and definite but it comes less and less clear and its end loses in a vagueness. A *countable class* is defined as an infinite class with a linear ordering such that each initial part (segment) is finite. For instance, a railway track with cross-ties leading straight to the horizon, days of our life we are to live or ever smaller and smaller reflections in two mirrors facing each other. The most important example is a class of natural numbers that will be discussed later.

The phenomenon of denumerability corresponds to a road towards the horizon. Though we get to the last point we can see, we can still go a bit further, the road will not disappear immediately. People have always tried to look a bit behind the horizon, to gain understanding and to overcome it in their thinking. This experience is expressed here by the important *axiom of prolongation* (see Axiom A6).

The other type of infinity, *continuum*, is based on the following experience. If we watch an object, however, are not able to distinguish individual elements which form it since they lie beyond the horizon of our perception. For example, the class of all geometric points in the plane, class of all atoms forming a table or grains of sand which together form a heap.

In fact the classical infinite mathematics, when applied to the real world, then solely to the above two types of infinity.

The intention of AST is to built on the natural world and human intuition. There is no reason for other types of infinity which are enforced in CST by its assumption that natural numbers form a set and that a power set is a set. That is why there are only two infinite cardinalities in AST: denumerability and continuum (see Axiom A8).

All examples from mathematical and real worlds are intentionally set out here together. They serve the purpose of inspiration to see where the idea of infinity comes from, they should be kept in mind when one deals with infinity.

The mathematical world is an ideal one, it is a perfect world of objective truths abstracted from all that is external. There is only little space for subjectivity of perception in it. That is why not all semisets from the real world may be interpreted directly.

The axiomatic system bellow describes that part of the AST which can be expressed in a strictly formal way. This basis provides space for extending AST by semisets which are parts of big, however, classically finite sets and thus make a lot of applications possible.

Axiomatic System of AST

[3] The language of AST uses symbols $\in$ and $=$, symbols X, Y, Z, … for class variables and symbols x, y, z, … for set variables. *Sets* are created by iteration from the empty set by Axiom A3. *Classes* are defined by formulas by Axiom A2. Every set is a class. Formally, a set is a class that is a member of another class:

$$\mathrm{Set}(X) \Leftrightarrow (\exists Y)(X \in Y).$$

AST is a theory with the following axioms:

- A1 (*extensionality*). $(X = Y) \Leftrightarrow (\forall Z)(Z \in X) \Leftrightarrow (Z \in Y)$;
- A2 (*existence of classes*). If ψ is a formula, then

$$(\exists Y)(\forall x)(x \in Y \Leftrightarrow \psi(x, X_1, \ldots, X_n));$$

- A3 (*existence of sets*).

$$\mathrm{Set}(\emptyset) \wedge (\forall x, y) Set(x \cup \{y\}).$$

A *set-formula* is a formula in which only set variables and constants occur.

- A4 (*induction*). If ψ is a set-formula, then $(\psi(\emptyset) \wedge (\forall x, y)(\psi(x) \Rightarrow \psi(x \cup \{y\})) \Rightarrow (\forall x)\, \psi(x)$.
- A5 (*regularity*). If ψ is a set-formula, then $(\exists x)\psi(x) \Rightarrow (\exists x)(\psi(x) \wedge (\forall y \in x)\neg\psi(y))$.

As usual, the class of *natural numbers* N is defined in the von Neumann way

$$N = \left\{ x\colon \begin{array}{c} (\forall y \in x)(y \subseteq x) \\ \wedge(\forall y, z \in x)(y \in z \vee y = z \vee z \in y) \end{array} \right\}$$

The class of *finite natural numbers* (*FN*) consists of the numbers represented by a finite set. They are accessible, easy to overlook and lie before the horizon:

$$FN = \{x \in N\colon\ \mathrm{Fin}(x)\}$$

FN forms a countable class in the sense described above. The class *FN* correspond to classical natural numbers and the class N to their nonstandard model. Both N and *FN* satisfy the axioms of Peano arithmetic.

Two classes X, Y are *equivalent* if there is a one-one mapping of X onto Y, i. e. $X \approx Y$.

- A6 (*prolongation*). Every countable function can be prolonged to a function which is a set, i. e. $(\forall F)((\mathrm{Fnc}(F) \wedge (F \approx FN)) \Rightarrow (\exists f)(\mathrm{Fnc}(f) \wedge F \subseteq f))$.

An easy corollary is that a countable class is a semiset. Also *FN* is a semiset and it can be prolonged to a set which is an element of N and which is greater than all finite natural numbers and so it represents an infinitely large natural number. Consequently, the class N is not countable.

The *universal class* V includes all sets created by iteration from the empty set.

- A7 (*choice*). The universal class V can be well ordered.
- A8 (*two cardinalities*). Every two infinite classes that are not countable are equivalent.

Thus, any infinite class is either equivalent to *FN* or N.

Using ultrapowers, the relative consistency of AST can be proved.

Rational and Real Numbers

Rational numbers Q are constructed in the usual way from N as the quotient field of the class $N \cup \{-n; n \in N\}$ Because N includes infinitely large numbers, Q includes infinitely small numbers.

Finite rational numbers FQ are similarly constructed from finite natural numbers *FN*. They include quantities that are before the horizon with respect to distance and depth. Surely $FQ \subseteq Q$.

We define that $x, y \in Q$ are *infinitely near* by

$$x \doteq y \Leftrightarrow (\forall n \in FN) \begin{cases} |x - y| < \frac{1}{n} \\ \vee(x > n \wedge y > n) \\ \vee(x < -n \wedge y < -n). \end{cases}$$

This relation is an equivalence. The corresponding partition classes are called *monads*. For $x \in Q$

$$\mathrm{Mon}(x) = \{y\colon\ y \doteq x\}\,.$$

Rational numbers x that are elements of Mon(0), i. e. ($x \dot{\in} 0$), are *infinitely small*. All monads are of the same nature except for the two limit ones. These consists of *infinitely large positive* and *negative* numbers. The class of bounded rational numbers is

$$BQ = \{x \in Q\colon\ (\exists n)((n \in FN) \wedge (|x| < n))\}$$

Now, it is easy and natural to construct *real numbers*:

$$\mathbb{R} = \{\mathrm{Mon}(x)\colon\ x \in BQ\}\,.$$

Real numbers built in this way display the same characteristics as real numbers in CST.

This motivation for expressing real numbers as monads of rational numbers corresponds rather to etymology than to the traditional interpretation. Rational numbers are constructed by reason, perfectly exact; their existence is purely abstract. On the other hand, real numbers are more similar to those that are used in the real world. If we say: one eighth of a cake, we surely do not expect it to be the ideal eighth, it is rather a portion which differs from the ideal one by a difference which is beyond the horizon of our perception. A similar situation occurs in the case of a pint of milk or twenty miles.

Infinitesimal Calculus

[12] Infinitesimal calculus in AST is based on the same point of view and intuition as that of its founders, I. Newton and G.W. Leibniz. It is so because infinitely small or infinitesimal quantities are naturally available in AST. For example, the limit of a function and the continuity in $a \in Q$ are defined, respectively, by:

$$\lim_{x \to a} f(x) = b$$
$$\Leftrightarrow (\forall x)((x \doteq a \wedge x \neq a) \Rightarrow f(x) \doteq b));$$
$$(\forall x)(x \doteq a \Rightarrow f(x) \doteq f(a)).$$

This topic is discussed in detail in [9]. As a method, these definitions were successfully used for teaching students.

Topology

Classes described by arbitrary formulas can be complex and difficult to capture. The easiest are sets, also classes described by using set-formulas, so-called *set-definable classes* (*Sd-classes*) can be described well. Semisets which are defined by a positive property (big, blue or happy and also distinguishable or to be a finite natural number) can be described as a countable union of Sd-classes, the so-called *σ-classes*. On the other hand, classes whose definition is based on negation (not big, not happy, indistinguishable), are the so-called *π-classes*—countable intersections of Sd-classes. A class which is at the same time π and σ is an Sd-class. Using combinations of π and σ, a set hierarchy can be described.

One of the most important tasks of mathematics is to handle the notion of the continuum. AST is based on the assumption that this phenomenon is caused by that of the indiscernibility of elements of the observed class. That is why, for the study of topology, the basic notion is a certain *relation of indiscernibility* ($\equiv$). Two elements are indiscernible if, when observed, available criteria that might distinguish them fail. It is a negative feature, therefore it must be a π-class. The relation of indiscernibility is naturally reflexive and symmetric. In pure mathematics, it is in addition transitive (because *FN* is closed under addition), thus it is an equivalence. This relation must also be compact, i. e. for each infinite set $u \subseteq \mathrm{dom}(\equiv)$ there are $x, y \in u$ such that $x \neq y \wedge x \equiv y$. The corresponding topological space is a compact metric space.

The relation of infinite nearness in rational numbers represents a special case of equivalence of indiscernibility.

Monads and *figures* correspond to phenomena of points and shapes, respectively:

$$\mathrm{Mon}(x) = \{y\colon\ y \equiv x\},$$
$$\mathrm{Fig}(X) = \{y\colon\ (\exists x \in X)(y \equiv x)\}.$$

Basic Definitions

Two classes X, Y are *separable*, $\mathrm{Sep}(X, Y) \Leftrightarrow (\exists Z)(\mathrm{Sd}(Z) \wedge \mathrm{Fig}(X) \subseteq Z \wedge \mathrm{Fig}(Y) \cap Z = \emptyset)$.

A *closure* $\overline{X}$ of a class X is defined as $\overline{X} = \{x\colon\ \neg\, \mathrm{Sep}(\{x\}, X)\}$.

A class X is *closed* if $X = \overline{X}$.

A set u is *connected* if $(\forall w)(\emptyset \neq = u \Rightarrow \mathrm{Fig}(w) \cap (u - w) \neq \emptyset)$.

It is quite easy to prove basic topological theorems. Also proofs of some classical theorems are much simpler here. For instance the *Sierpinski theorem*: If v is a connected set then $\mathrm{Fig}(v)$ cannot be expressed as a countable union of disjoint closed sets.

The *fundamental indiscernibility* $\overset{\circ}{=}_c$ is defined as follows. If c is a set then $x \overset{\circ}{=}_c y$ if for any set-formula ψ with the constants from c and for any x, it is $\psi\ (x) \Leftrightarrow \psi\ (y)$.

This relation has a special position. For any relation of indiscernibility $\equiv$ there is a set c such that $\overset{\circ}{=}_{\{c\}}$ is finer than $\equiv$ i. e. $\overset{\circ}{=}_{\{c\{} \subseteq \equiv$.

Motion

Unlike classical mathematics, the motion is captured in AST by any relation of indiscernibility $\equiv$.

Everybody knows the way films work. Pictures coming one after another are almost indiscernible from each other, however, when shown in a rapid sequence, the pictures start to move. The continuous motion may be viewed like this, as a sequence of indiscernible stages in certain time intervals.

A function d is a *motion of a point* in the time $\delta \in N$ if $\mathrm{dom}(f) = \delta \wedge (\forall \alpha < \delta)(d(\alpha) \equiv d(\alpha{+}1))$.

If $\delta \in FN$ then the point does not move, it can move only in an infinitely big time interval.

A sequence $\{d(\alpha)\colon \alpha \in \mathrm{dom}(d)\}$ is a sequence of states. The number $\delta = \mathrm{dom}(d)$ is the number of moments and $\mathrm{rng}(d)$ is the trace of a moving point.

A trace is a connected set and for each nonempty connected set u there is a motion of a point such that u is the trace of d.

A *motion of a set* is defined similarly, only the last condition is different: $(\forall \alpha < \delta)(\mathrm{Fig}(d(\alpha)) = \mathrm{Fig}(d(\alpha{+}1)))$.

The following theorem is proved in [10,11]: Each motion of a set may be divided into motions of points. This does not involve only the mechanical motion, but any motion describing a continuous change. Thus, for example, even the growth of a tree from a planted seed may be divided into movements of individual points while all of their initial stages are already contained in the seed. In addition, it is possible to describe conditions under which such a change is still continuous.

Utility Theory

[7] The utility theory is one of nice examples of applying AST. Its aim is to find a valuation of elements of a class S. There is a preference relation $\succ$ on linear combinations of elements of S with finite rational coefficients, i. e. on the class

$$\left\{ \sum_{i=1}^{n} \alpha_i u_i : \begin{array}{c} (n \in FN) \\ \wedge(\forall i)((i \le n)(u_i \in S) \wedge (\alpha_i \in FQ)) \\ \wedge \sum_{i=1}^{n} \alpha_i = 1 \end{array} \right\}.$$

An interpretation of a combination is a game in which every u_i can be won with the probability α_i. The *preference relation* $\succ$ declares which of the two games is preferred.

The *valuation* is a function F from the class S to Q for which

$$\sum_{i=1}^{n} \alpha_i u_i \succ \sum_{j=1}^{m} \beta_j u_j \Leftrightarrow \sum_{i=1}^{n} \alpha_i F(u_i) > \sum_{j=1}^{m} \beta_j F(u_j).$$

It is not necessary to require the so-called Archimedes property on the relation of preference thanks to the possibility of using infinitely small and infinitely large rational numbers. It is possible to capture finer and more complex relations than in classic mathematics, e. g. the fact that the value of one element is incomparably higher than that of another element or it is possible to compare infinitely small differences of values.

For each class S with a preference relation a valuation may be found. Such a valuation is not uniquely defined, it is possible to construct it so that $\mathrm{rng}(F) \subseteq N$.

Conclusion

The aim of this short survey is to demonstrate the basic ideas of AST. Yet, there are other areas of mathematics which were studied in it, for instance measurability [8], ultrafilters [6], endomorphic universes [5] and automorphisms of natural numbers [2], representability [1] metamathematics [3] and models of AST [4].

See also

- ▶ Boolean and Fuzzy Relations
- ▶ Checklist Paradigm Semantics for Fuzzy Logics
- ▶ Finite Complete Systems of Many-valued Logic Algebras
- ▶ Inference of Monotone Boolean Functions
- ▶ Optimization in Boolean Classification Problems
- ▶ Optimization in Classifying Text Documents

References

1. Mlček J (1979) Valuation of structures. Comment Math Univ Carolinae 20:681–695
2. Mlček J (1985) Some automorphisms of natural numbers in AST. Comment Math Univ Carolinae 26:467–475
3. Sochor A (1992) Metamathematics of AST. From the logical point of view 1:61–75
4. Sochor A, Pudlák P (1984) Models of AST. J Symbolic Logic 49:570–585
5. Sochor A, Vopěnka P (1979) Endomorfic universes and their standard extensions. Comm Math Univ Carolinae 20:605–629
6. Sochor A, Vopěnka P (1981) Ultrafilters of sets. Comment Math Univ Carolinae 22:698–699
7. Trlifajová K, Vopěnka P (1985) Utility theory in AST. Comment Math Univ Carolinae 26:699–711
8. Čuda K (1986) The consistency of measurability of projective semisets. Comment Math Univ Carolinae 27:103–121
9. Čuda K, Sochor A, Vopěnka P, Zlatoš P (1989) Guide to AST. Proc. First Symp. Mathematics in AST, Assoc. Slovak Mathematicians and Physicists, Bratislava
10. Vopénka P (1979) Mathematics in AST. Teubner, Leipzig
11. Vopénka P (1989) Introduction to mathematics in AST. Alfa Bratislava, Bratislava
12. Vopénka P (1996) Calculus infinitesimalis-pars prima. Práh Praha, Praha

Approximation of Extremum Problems with Probability Functionals

APF

Riho Lepp
Tallinn Technical University, Tallinn, Estonia

MSC2000: 90C15

Article Outline

Keywords

Discrete approximation; Probability functionals

To ensure a certain level of reliability for the solution of an extremum problem under uncertainty it has become a spread approach to introduce probabilistic (chance) cost and/or constraints into the model. The stability analysis of chance constraint problems is rather complicated due to complicated properties of the *probability function* $v_t(x)$, defined as

$$v_t(x) = \mathrm{P}\{s\colon\ f(x,s) \le t\}\,. \tag{1}$$

Here $f(x, s)$ is a real valued function, defined on $\mathbf{R}^r \times \mathbf{R}^r$, t is a fixed level of reliability, s is a random vector and P denotes probability. The function $v_t(x)$ is never convex, only in some cases (e. g., $f(x, s)$ linear in s and distribution of the random parameter s normal), it is quasiconvex. Note that for a fixed x function $v_t(x)$, as a function of t, is the distribution function of the random variable $f(x, s)$.

The 'inverse', the *quantile function* $w_\alpha(x)$, to the probability function $v_t(x)$ is defined in such a way that the probability level α, $0 < \alpha < 1$, is fixed earlier, and the purpose is to minimize the reliability level t:

$$w_\alpha(x) = \min_t \{t\colon\ \mathrm{P}\{s\colon\ f(x,s) \le t\} \ge \alpha\}\,. \tag{2}$$

Varied examples of extremum problems with probability and quantile functions are presented in [7] and in [8]. Some of these models have such a complicated structure, see [8, Chap. 1.8], about correction of a satellite orbit, that we are forced to look for a solution x from a certain class of strategies, that means, the solution x itself depends on the random parameter s, $x = x(s)$.

This class of probability functions was introduced to *stochastic programming* by E. Raik, and lower semicontinuity and continuity properties of $v_t(x)$ and $w_\alpha(x)$ in Lebesgue L^p-spaces, $1 \le \mathrm{p} < \infty$, were studied in [12]. Simultaneously, in [4] problems with various classes of solutions $x(s)$ (measurable, continuous, linear, etc) were considered. Since the paper [4] solutions $x(s)$ are called *decision rules*, and we will follow also this terminology.

Differently from [4], here we will consider approximation of a decision rule $x(s)$ by sequences of vectors $\{x_n\}$, $x_n = (x_{1n}, \ldots, x_{nn})$, $n = 1, 2, \ldots$, with increasing dimension in order to maximize the value of the probability functional $v(x)$ under certain set C of decision rules. It will be assumed that the set C will be bounded in the space $L^1(S, \Sigma, \sigma) = L^1(\sigma)$ of integrable functions $x(s)$, $x \in L^1(\sigma)$:

$$\max_{x\in C} v_t(x) = \max_{x\in C} \mathrm{P}\{s\colon\ f(x(s),s) \le t\}\,. \tag{3}$$

Here S is the support of random variable s with distribution (probability measure) $\sigma(\cdot)$ and Σ denotes the sigma-algebra of Borel measurable sets from $\mathbf{R}^r$.

Due to technical reasons we are forced to assume that the random parameter s has bounded support $S \subset \mathbf{R}^r$, diam $S < \infty$, and its distribution σ is atomless,

$$\sigma\{s\colon\ |s - s_0| = \mathrm{const}\} = 0, \quad \forall s_0 \in \mathbb{R}^r. \tag{4}$$

Since the problem (3) is formulated in the function space $L^1(\sigma)$ of σ-integrable functions, the first step in its solution is the approximation step where we will replace the initial problem (3) by a sequence of finite-dimensional optimization problems with increasing dimension. Second step, solution methods were considered in a series of papers of the author (see, e. g., [9]), where the gradient projection method was suggested together with simultaneous Parzen–Rosenblatt kernel-type smooth approximation of the discontinuous integrand from (1).

There are several ways to divide the support S of the probability measure σ into smaller parts in discretization, e. g., taking disjoint subsets S_j, $j = 1, \ldots, k$, of S

from the initial sigma-algebra Σ as in [11], or using in the partition of S only convex sets from Σ, as in [5].

We will divide the support S into smaller parts by using only sets A_{in}, $i = 1, \ldots, n$, $n \in N = \{1, 2, \ldots\}$, with σ-measure zero of their boundary, i. e., $\sigma\ (\mathrm{int} A_{in}) = \sigma\ (A_{in}) = \sigma\ (\mathrm{cl} A_{in})$, where int A and cl A denote topological interior and closure of a set A, respectively. Such division is equivalent to *weak convergence* of a sequence of *discrete measures* $\{(m_n, s_n)\}$ to the initial probability measure σ, see, e. g. [14]:

$$\sum_{i=1}^{n} h(s_{in}) m_{in} \to \int_S h(s)\, \sigma(ds), \quad n \in \mathbb{N}, \tag{5}$$

for any continuous on S function $h(s)$, $h \in C(S)$.

The usage of the weak convergence of discrete measures in stochastic programming has its disadvantages and advantages. An example in [13] shows that, in general, the stability of a probability function with respect to weak convergence cannot be expected without additional smoothness assumptions on the measure σ. This is one of the reasons, why we should use only continuous measures with the property (4). An advantage of the usage of the weak convergence is that it allows us to apply in the approximation process instead of conditional means [11] the more simple, grid point approximation scheme.

Since the functional $v_t(x)$ is not convex, we are not able to exploit in the stability analysis of discrete approximation of the problem (3) the more convenient, weak topology, but only the strong (norm) topology. As the first step we will approximate $v_t(x)$ so, that the discrete analogue of continuous convergence of a sequence of approximate functionals will be guaranteed.

Schemes of *stability analysis* (e. g., finite-dimensional approximations) of extremum problems in Banach spaces require from the sequence of solutions of 'approximate' problems certain kind of compactness. Assuming that the constraint set C is compact in $L^1(\sigma)$, we, as the second step, will approximate the set C by a sequence of finite-dimensional sets $\{C_n\}$ with increasing dimension so, that the sequence of solutions of approximate problems is compact in a certain (discrete convergence) sense in $L^1(\sigma)$. Then the approximation scheme for the discrete approximation of (3) will follow formed schemes of approximation of extremum problems in Banach spaces, see e. g. [2,3,15].

Redefine the functional $v_t(x)$ by using the Heaviside zero-one function χ:

$$v_t(x) = \int_S \chi(t - f(x(s), s))\, \sigma(ds), \tag{6}$$

where

$$\chi(t - f(x(s), s)) = \begin{cases} 1 & \text{if } f(x(s), s) \le t, \\ 0 & \text{if } f(x(s), s) > t. \end{cases}$$

Since the integrand $\chi(\cdot)$ itself, as a zero-one function, is discontinuous, we will assume that the function $f(x, s)$ is continuous both in (x, s) and satisfies following growth and 'platform' conditions:

$$|f(x, s)| \le a(s) + \alpha\, |x|\,, \tag{7}$$

$$a \in L^1(\sigma), \quad \alpha > 0,$$

$$\sigma\,\{s\colon\ f(x, s) = \mathrm{const}\} = 0, \tag{8}$$

$$\forall (x, s) \in \mathbb{R}^r \times S.$$

The continuity assumption is technical in order to simplify the description of the approximation scheme below. The growth condition (7) is essential: without it the superposition operator $f(x) = f(x(s), s)$ will not map an element from L^1 to L^1 (is even not defined). Condition (8) means that the function $f(x, s)$ should not have horizontal platforms with positive measure.

Constraint set C is assumed to be a set of integrable functions $x(s)$, $x \in L^1(\sigma)$, with properties

$$\int_S |x(s)|\ \sigma(ds) \le M < \infty, \quad \forall x \in C \tag{9}$$

for some $M > 0$ (C is bounded in $L^1(\sigma)$);

$$\int_D |x(s)| \le K\sigma(D), \quad \forall x \in C, \quad D \in \Sigma \tag{10}$$

for some $K > 0$;

$$(x(s) - x(t), s - t) \ge 0 \quad \text{for a.a. } s, t \in S \tag{11}$$

(functions $x \in C$ are monotone almost everywhere and a.a. denotes abbreviation of 'almost all').

Conditions (9), (10) guarantee that the set C is weakly compact (i. e., compact in the (L^1, L^∞)-topology, see, e. g., [6, Chap. 9.1.2]). Condition (11) guarantees now, following [1, Lemma 3], that the set C is strongly compact in $L^1(\sigma)$. Then, following [11], we can conclude that assumptions (7)–(11) together with

atomless assumption (4) for the measure σ guarantee the existence of a solution of problem (3) in the Banach space $L^1(\sigma)$ of σ-integrable functions (the cost functional $v_t(x)$ is continuous in x and the constraint set C is compact in $L^1(\sigma)$).

Since approximate problems will be defined in $\mathbf{R}^{rn}$, we should define a system of connection operators $\mathcal{P} = \{p_n\}$ between spaces $L^1(\sigma)$ and $\mathbf{R}^{rn}$, $n \in \mathbf{N}$. In L^p-spaces, $1 \le p \le \infty$, systems of connection operators should be defined in a piecewise integral form (as conditional means):

$$(p_n x)_{in} = \sigma(A_{in})^{-1} \int_{A_{in}} x(s)\, \sigma(ds), \tag{12}$$

where $i = 1, \ldots, n$, and sets A_{in}, $i = 1, \ldots, n$, $n \in \mathbf{N}$, that define connection operators (12), satisfy following conditions A1)–A7):

A1) $\sigma(A_{in}) > 0$;
A2) $A_{in} \cap A_{jn} = \emptyset$, $i \ne j$;
A3) $\cup_{i=1}^{n} A_{in} = S$;
A4) $\sum_{i=1}^{n} |m_{in} - \sigma(A_{in})| \to 0$, $n \in \mathbf{N}$;
A5) $\max_i \operatorname{diam} A_{in} \to 0$, $n \in \mathbf{N}$;
A6) $s_{in} \in A_{in}$;
A7) $\sigma(\operatorname{int} A_{in}) = \sigma(A_{in}) = \sigma(\operatorname{cl} A_{in})$.

Remark 1 Weak convergence (5) is equivalent to the partition $\{\mathcal{A}_n\}$ of S, $\mathcal{A}_n = \{A_{1n}, \ldots, A_{nn}\}$, with properties A1)–A7), see [14].

Remark 2 Collection of sets $\{A_{in}\}$ with the property A7) constitutes an algebra $\Sigma_0 \subset \Sigma$, and if $S = [0, 1]$ and if σ is Lebesgue measure on $[0, 1]$, then integrability relative to $\sigma|_{\Sigma_0}$ means Riemann integrability.

Define now the *discrete convergence* for the space $L^1(\sigma)$ of σ-integrable functions.

Definition 3 A sequence of vectors $\{x_n\}$, $x_n \in \mathbf{R}^{rn}$, *$\mathcal{P}$-converges* (or *converges discretely*) to an integrable function $x(s)$, if

$$\sum_{i=1}^{n} |x_{in} - (p_n x)_{in}|\, m_{in} \to 0, \quad n \in \mathbb{N}. \tag{13}$$

Remark 4 Note that in the space $L^1(\sigma)$ of σ-integrable functions we are also able to use the projection methods approach, defining convergence of $\{x_n\}$ to $x(s)$ as follows:

$$\int_S \left| x(s) - \sum_{i=1}^{n} x_{in} \chi_{A_{in}}(s) \right| \sigma(ds) \to 0, \quad n \in \mathbb{N}.$$

Remark 5 Projection methods approach does not work in the space $L^\infty(\sigma)$ of essentially bounded measurable functions with vraisup-norm topology ($L^\infty(\sigma)$ is a non-separable Banach space and the space $C(S)$ of continuous functions is not dense there).

We need the space $L^\infty(\sigma)$, which is the topological dual to the space $L^1(\sigma)$ of σ-integrable functions, in order to define also the discrete analogue of the weak convergence in $L^1(\sigma)$.

Definition 6 A Sequence of vectors $\{x_n\}$, $x_n \in \mathbf{R}^{rn}$, $n \in \mathbf{N}$, *$w\mathcal{P}$-converges* (or *converges weakly discretely*) to an integrable function $x(s)$, $x \in L^1(\sigma)$, if

$$\sum_{i=1}^{n} (z_{in}, x_{in}) m_{in} \to \int_S (z(s), x(s))\, \sigma(ds), \quad n \in \mathbb{N}, \tag{14}$$

for any sequence $\{z_n\}$ of vectors, $z_n \in \mathbf{R}^{rn}$, $n \in \mathbf{N}$, and function $z(s)$, $z \in L^\infty(\sigma)$, such that

$$\max_{1 \le i \le n} |z_{in} - (p_n z)_{in}| \to 0, \quad n \in \mathbb{N}. \tag{15}$$

In order to formulate the discretized problem and to simplify the presentation, we will assume that in partition $\{\mathcal{A}_n\}$ of S, where $\mathcal{A}_n = \{A_{1n}, \ldots, A_{nn}\}$, with properties A1)–A7), in property A4) we will identify m_{in} and $\sigma(A_{in})$, i. e. $m_{in} = \sigma(A_{in})$ (e. g. squares with decreasing diagonal in $\mathbf{R}^2$).

Discretize now the probability functional $v_t(x)$:

$$v_{tn}(x_n) = \sum_{i=1}^{n} \chi(t - f(x_{in}, s_{in})) m_{in}, \tag{16}$$

and formulate the discretized problem:

$$\max_{x_n \in C_n} v_{tn}(x_n) = \max_{x_n \in C_n} \sum_{i=1}^{n} \chi(t - f(x_{in}, s_{in})) m_{in}, \tag{17}$$

where constraint set C_n will satisfy discrete analogues of conditions (9)–(11), covered to the set C:

$$\sum_{i=1}^{n} |x_{in}|\, m_{in} \le M \quad \forall x_n \in C_n, \tag{18}$$

$$\sum_{i \in I_n} |x_{in}|\, m_{in} \le K \sum_{i \in I_n} m_{in}, \quad \forall x_n \in C_n, \quad \forall I_n \subset \{1, \ldots, n\}, \tag{19}$$

$$\sum_{k=1}^{r}(x^k_{i_k n} - x^k_{j_k n})(i_k - j_k) \geq 0, \quad \forall i_k, j_k : \quad i_k < j_k \ , \tag{20}$$

and such that $0 \leq i_k, j_k \leq n$, $\forall n \in \mathbf{N}$.

Definition 7 A sequence of sets $\{C_n\}$, $C_n \subset \mathbf{R}^{rn}$, $n \in \mathbf{N}$, converges to the set $C \subset L^1(\sigma)$ in the *discrete Mosco sense* if

1) for any subsequence $\{x_n\}$, $n \in \mathbf{N}' \subset \mathbf{N}$, such that $x_n \in C_n$, from convergence $w\mathcal{P}\text{-lim } x_n = x$, $n \in \mathbf{N}$, it follows that $x \in C$;
2) for any $x \in C$ there exists a sequence $\{x_n\}$, $x_n \in C_n$, which $\mathcal{P}$-converges to x, $\mathcal{P}\text{-lim } x_n = x$, $n \in \mathbf{N}$.

Remark 8 If in the above definition also 'for any' part 1) is defined for $\mathcal{P}$-convergence of vectors, then it is said that sequence of sets $\{C_n\}$ converges to the set C in the *discrete Painlevé–Kuratowski sense*.

Denote optimal values and optimal solutions of problems (3) and (17) by v^*, x^* and v^*_n, x^*_n, respectively.

Let function $f(x, s)$ be continuous in both variables (x, s) and satisfy growth and platform conditions (7) and (8). Then from convergence $\mathcal{P}\text{-lim } x_n = x$, $n \in \mathbf{N}$, for any monotone a.e. function $x(s)$, it follows convergence $v_n(x_n) \to v(x)$, $n \in \mathbf{N}$.

Verification of this statement is quite lengthy and technically complicated: we should first approximate discontinuous function $\chi(t - f(x, s))$ by continuous function $\chi_c(t - f(x, s))$ in the following way:

$$\chi_c(t - f(x,s)) = \begin{cases} 1 & \text{if } f(x,s) \leq t, \\ 1 - \delta^{-1}[f(x,s) - t] & \text{if } t < f(x,s) \leq t + \delta, \\ 0 & \text{if } f(x,s) > t + \delta \end{cases}$$

for some (small) δ, and then a discontinuous solution $x(s)$, $x \in L^1(\sigma)$, by continuous function $x_c(s)$ (in L^1-norm topology).

Let constraint sets C and C_n satisfy conditions (9)–(11) and (18)–(20), respectively. Let discrete measures $\{(m_n, s_n)\}$ converge weakly to the measure σ. Then the sequence of sets $\{C_n\}$ converges to the set C in the discrete Painlevé–Kuratowski sense.

Verification of this statement relies on the two following convergences:

1) sequence of sets, determined by inequalities (18), (19) converges, assuming weak convergence of discrete measures (5), in discrete Mosco sense to the weakly compact in $L^1(\sigma)$ set, determined by inequalities (9), (10);
2) adding to both, approximate and initial sets of admissible solutions monotonicity conditions (20) and (11), respectively, we can guarantee the discrete convergence of sequence $\{C_n\}$ to C in Painlevé–Kuratowski sense.

Now we can formulate the discrete approximation conditions for a stochastic programming problem with probability cost function in the class of integrable decision rules.

Let function $f(x, s)$ be continuous in both variables (x, s) and satisfy growth and platform conditions (7) and (8), constraint set C satisfy conditions (9)–(11) and let discrete measures $\{(m_n, s_n)\}$ converge weakly to the atomless measure σ. Then $v^*_n \to v^*$, $n \in \mathbf{N}$, and sequence of solutions $\{x^*_n\}$ of approximate problems (17) has a subsequence, which converges discretely to a solution of the initial problem (3).

Remark 9 The usage of the space $L^1(\sigma)$ of integrable functions is essential. In reflexive L^p-spaces, $1 < p < \infty$, serious difficulties arise with application of the strong (norm) compactness criterion for a maximizing sequence.

As a rule, problems with probability cost function are maximized, whereas stochastic programs with quantile cost are minimized, see, e. g., [8,10].

Consider at last discrete approximation of the quantile minimization problem (2):

$$\min_{x \in C} w_\alpha(x) = \min_{x \in C} \min_t \{\mathsf{P}(f(x(s), s) \leq t) \geq \alpha\}, \tag{21}$$

It was verified in [10] that under certain (quasi)-convexity-concavity assumptions the quantile minimization problem (21) is equivalent to the following Nash game:

$$\max_{x \in C} v_t(x) = J^*_1, \tag{22}$$

$$\min_t (v_t(x) - \alpha)^2 = J^*_2. \tag{23}$$

Discretizing $v_t(x)$ as in (16) and $w_\alpha(x)$ as

$$w_{\alpha n}(x_n) = \min_t \left\{ \sum_{i=1}^{n} \chi(t - f(x_{in}, s_{in}))m_{in} \geq \alpha \right\},$$

we can, analogously to the probability functional approximation, approximate the quantile minimization problem (21) too. In other words, to replace the Nash game (22), (23) with the following finite-dimensional game:

$$\max_{x_n \in C_n} v_{tn}(x_n) = J_{1n}^*, \tag{24}$$

$$\min_t (v_{tn}(x_n) - \alpha)^2 = J_{2n}^*. \tag{25}$$

Verification of convergences $J_{1n}^* \to J_1^*$ and $J_{2n}^* \to J_2^*$, $n \in \mathbf{N}$, is a little bit more labor-consuming compared with approximate maximization of probability functional $v_t(x)$, since we should guarantee also convergence of the sequence of optimal quantiles $\{t_n^*\}$ of minimization problems (25).

See also

- ▶ Approximation of Multivariate Probability Integrals
- ▶ Discretely Distributed Stochastic Programs: Descent Directions and Efficient Points
- ▶ Extremum Problems with Probability Functions: Kernel Type Solution Methods
- ▶ General Moment Optimization Problems
- ▶ Logconcave Measures, Logconvexity
- ▶ Logconcavity of Discrete Distributions
- ▶ L-Shaped Method for Two-Stage Stochastic Programs with Recourse
- ▶ Multistage Stochastic Programming: Barycentric Approximation
- ▶ Preprocessing in Stochastic Programming
- ▶ Probabilistic Constrained Linear Programming: Duality Theory
- ▶ Probabilistic Constrained Problems: Convexity Theory
- ▶ Simple Recourse Problem: Dual Method
- ▶ Simple Recourse Problem: Primal Method
- ▶ Stabilization of Cutting Plane Algorithms for Stochastic Linear Programming Problems
- ▶ Static Stochastic Programming Models
- ▶ Static Stochastic Programming Models: Conditional Expectations
- ▶ Stochastic Integer Programming: Continuity, Stability, Rates of Convergence
- ▶ Stochastic Integer Programs
- ▶ Stochastic Linear Programming: Decomposition and Cutting Planes
- ▶ Stochastic Linear Programs with Recourse and Arbitrary Multivariate Distributions
- ▶ Stochastic Network Problems: Massively Parallel Solution
- ▶ Stochastic Programming: Minimax Approach
- ▶ Stochastic Programming Models: Random Objective
- ▶ Stochastic Programming: Nonanticipativity and Lagrange Multipliers
- ▶ Stochastic Programming with Simple Integer Recourse
- ▶ Stochastic Programs with Recourse: Upper Bounds
- ▶ Stochastic Quasigradient Methods in Minimax Problems
- ▶ Stochastic Vehicle Routing Problems
- ▶ Two-stage Stochastic Programming: Quasigradient Method
- ▶ Two-Stage Stochastic Programs with Recourse

References

1. Banaš J (1989) Integrable solutions of Hammerstein and Urysohn integral equations. J Austral Math Soc (Ser A) 46:61–68
2. Daniel JW (1971) The approximate minimization of functionals. Prentice-Hall, Englewood Cliffs
3. Esser H (1973) Zur Diskretisierung von Extremalproblemen. Lecture Notes Math, vol 333. Springer, Berlin, pp 69–88
4. Garstka J, Wets RJ-B (1974) On decision rules in stochastic programming. Math Program 7:117–143
5. Hernandez-Lerma O, Runggaldier W (1994) Monotone approximations for convex stochastic control problems. J Math Syst, Estimation and Control 4:99–140
6. Ioffe AD, Tikhomirov VM (1979) Theory of extremal problems. North-Holland, Amsterdam
7. Kall P, Wallace SW (1994) Stochastic programming. Wiley, New York
8. Kibzun AI, Kan YS (1995) Stochastic programming problems with probability and quantile functions. Wiley, New York
9. Lepp R (1983) Stochastic approximation type algorithm for the maximization of the probability function. Proc Acad Sci Estonian SSR Phys Math 32:150–156

10. Malyshev VV, Kibzun AI (1987) Analysis and synthesis of high precision aircraft control. Mashinostroenie, Moscow, Moscow
11. Olsen P (1976) Discretization of multistage stochastic programming problems. Math Program Stud 6:111–124
12. Raik E (1972) On stochastic programming problem with probability and quantile functionals. Proc Acad Sci Estonian SSR Phys Math 21:142–148
13. Römisch W, Schultz R (1988) On distribution sensitivity in chance constrained programming. Math Res 45:161–168. Advances in Mathematical Optimization, In: Guddat J et al (eds)
14. Vainikko GM (1971) On convergence of the method of mechanical curvatures for integral equations with discontinuous kernels. Sibirsk Mat Zh 12:40–53
15. Vasin VV (1982) Discrete approximation and stability in extremal problems. USSR Comput Math Math Phys 22:57–74

Approximation of Multivariate Probability Integrals

TAMÁS SZÁNTAI
Technical University, Budapest, Hungary

MSC2000: 65C05, 65D30, 65Cxx, 65C30, 65C40, 65C50, 65C60, 90C15

Article Outline

Keywords

Boole–Bonferroni bounds; Hunter–Worsley bounds; Approximation; Probability integrals; Variance reduction; Probabilistic constrained stochastic programming

Approximation of *multivariate probability integrals* is a hard problem in general. However, if the domain of the probability integral is multidimensional interval, then the problem reduces to the approximation of *multivariate probability distribution function* values.

Lower and Upper Bounds

Let $\xi^{\mathsf{T}} = (\xi_1, \ldots, \xi_n)$ be a random vector with given multivariate probability distribution. Introducing the events

$$A_1 = \{\xi_1 < x_1\}, \ldots, A_n = \{\xi_n < x_n\},$$

where $x_1, \ldots, x_n$ are arbitrary real values the multivariate probability distribution function of the random vector ξ can be expressed in the following way:

$$\begin{aligned} &F(x_1, \ldots, x_n) \\ &= \mathsf{P}(\xi_1 < x_1, \ldots, \xi_n < x_n) \\ &= \mathsf{P}(A_1 \cap \cdots \cap A_n) \\ &= 1 - \mathsf{P}(\overline{A}_1 \cup \cdots \cup \overline{A}_n) \\ &= 1 - \overline{S}_1 + \overline{S}_2 - \cdots + (-1)^n \overline{S}^n, \end{aligned}$$

where

$$\overline{A}_i = \{\xi_i \geq x_i\}, \quad i = 1, \ldots, n,$$

and

$$\overline{S}_k = \sum_{1 \leq i_1 < \cdots < i_k \leq n} \mathsf{P}(\overline{A}_{i_1} \cap \cdots \cap \overline{A}_{i_k}), k = 1, \ldots, n\,.$$

First one shows that $\overline{S}_1, \overline{S}_2$ and so the individual probabilities $\mathsf{P}(\overline{A}_i)$, $i = 1, \ldots, n$, $\mathsf{P}(\overline{A}_i \cap \overline{A}_j)$, $i = 1, \ldots, n-1$, $j = i+1, \ldots, n$, involved in them can be expressed by $F_i(x_i)$, $i = 1, \ldots, n$, and $F_{ij}(x_i, x_j)$, $i = 1, \ldots, n-1, j = i+1, \ldots, n$, the *one-* and *two-dimensional marginal probability distribution functions* of the random vector ξ. One has

$$\begin{aligned} \overline{S}_1 &= \sum_{i=1}^{n} \mathsf{P}(\overline{A}_i) = \sum_{i=1}^{n} \mathsf{P}(\xi_i \geq x_i) \\ &= n - \sum_{i=1}^{n} \mathsf{P}(\xi_i < x_i) = n - \sum_{i=1}^{n} F_i(x_i) \end{aligned}$$

and

$$\begin{aligned}\overline{S}_2 &= \sum_{1\le i<j\le n} \mathsf{P}(\overline{A}_i \cap \overline{A}_j)\\ &= \sum_{1\le i<j\le n} \mathsf{P}(\xi_i \ge x_i,\ \xi_j \ge x_j)\\ &= \sum_{1\le i<j\le n} \{1-\mathsf{P}(\xi_i < x_i)\\ &\quad -\mathsf{P}(\xi_j < x_j) + \mathsf{P}(\xi_i < x_i, \xi_j < x_j)\}\\ &= \frac{n(n-1)}{2} - (n-1)\sum_{i=1}^{n} F_i(x_i)\\ &\quad + \sum_{1\le i<j\le n} F_{ij}(x_i, x_j).\end{aligned}$$

So if one can calculate the one- and two-dimensional marginal probability distribution functions of the random vector ξ then one can bound the multivariate probability distribution function by the very simple bounds given by C.E. Bonferroni [1]:

$$1-\overline{S}_1 \le F(x_1,\ldots,x_n) \le 1-\overline{S}_1+\overline{S}_2,$$

or by the sharp bounds, called *Boole–Bonferroni bounds* discovered independently by many authors (see [11] for a summary):

$$\begin{aligned}&1-\overline{S}_1+\frac{2}{n}\overline{S}_2\\ &\le F(x_1,\ldots,x_n)\\ &\le 1-\frac{2}{k^*+1}\overline{S}_1+\frac{2}{k^*(k^*+1)}\overline{S}_2,\end{aligned}$$

where

$$k^* = 1+\left\lfloor \frac{2\overline{S}_2}{\overline{S}_1}\right\rfloor.$$

When applying the above bounds usually the upper bound proves to be sharper. However one can improve the lower bound by the application of the bound discovered independently by D. Hunter [5] and K.J. Worsley [18]. This bound is an upper bound for $\mathsf{P}(\overline{A}_1\cup\cdots\cup\overline{A}_n)$ by the use of $\overline{S}_1$ and the individual probabilities $\mathsf{P}(\overline{A}_i\cap\overline{A}_j)$, $1\le i<j\le n$. It is constructed in the following way. Construct a nonoriented complete graph with n nodes and assign to node i the event $\overline{A}_i$ (or the probability $\mathsf{P}(\overline{A}_i)$) and to arc (i,j) the weight $\mathsf{P}(\overline{A}_i\cap\overline{A}_j)$. Let T^* be a maximum weight spanning tree in this nonoriented complete graph then one has

$$\mathsf{P}(\overline{A}_1\cup\cdots\cup\overline{A}_n) \le \overline{S}_1 - \sum_{(i,j)\in T^*}\mathsf{P}(\overline{A}_i\cap\overline{A}_j),$$

which is called the *Hunter–Worsley upper bound*. This results the following lower bound on the multivariate probability distribution function:

$$1-\overline{S}_1+\sum_{(i,j)\in T^*}\mathsf{P}(\overline{A}_i\cap\overline{A}_j) \le F(x_1,\ldots,x_n).$$

The individual probabilities $\mathsf{P}(\overline{A}_i\cap\overline{A}_j)$, $1\le i<j\le n$, can be stored when one calculates the value of $\overline{S}_2$ and the maximum weight spanning tree can be found by several fast algorithms, for example by *Kruskal's algorithm*, see [9]. Now one has three lower and two upper bounds on the multivariate probability distribution function and all of them are computable if the one- and two-dimensional marginal probability distribution functions are known. Let us denote these bounds in the following way:

$$\begin{aligned}L_1 &= 1-\overline{S}_1,\\ L_2 &= 1-\overline{S}_1+\frac{2}{n}\overline{S}_2,\\ L_3 &= 1-\overline{S}_1+\sum_{(i,j)\in T^*}\mathsf{P}(\overline{A}_i\cap\overline{A}_j),\\ U_1 &= 1-\overline{S}_1+\overline{S}_2,\\ U_2 &= 1-\frac{2}{k^*+1}\overline{S}_1+\frac{2}{k^*(k^*+1)}\overline{S}_2.\end{aligned}$$

As one has $L_1\le L_2\le L_3$ and $U_2\le U_1$, the best lower bound is L_3 and the best upper bound is U_2.

Monte-Carlo Simulation Algorithm

One can take the differences between the multivariate probability distribution function and its lower and upper bounds introduced before:

$$\begin{aligned}&F(x_1,\ldots,x_n)-L_1 = \overline{S}_2-\overline{S}_3+\cdots+(-1)^n\overline{S}_n,\\ &F(x_1,\ldots,x_n)-L_2\\ &\quad=\left(1-\frac{2}{n}\right)\overline{S}_2-\overline{S}_3+\cdots+(-1)^n\overline{S}_n,\end{aligned}$$

$$F(x_1,\dots,x_n) - L_3$$
$$= -\sum_{(i,j)\in T^*} \mathsf{P}(\overline{A}_i \cap \overline{A}_j) + \overline{S}_2 - \overline{S}_3 + \cdots + (-1)^n \overline{S}_n,$$
$$F(x_1,\dots,x_n) - U_1 = \overline{S}_3 + \cdots + (-1)^n \overline{S}_n,$$
$$F(x_1,\dots,x_n) - U_2$$
$$= \left(\frac{2}{k^*+1} - 1\right)\overline{S}_1 + \left(1 - \frac{2}{k^*(k^*+1)}\right)\overline{S}_2 - \overline{S}_3 + \cdots + (-1)^n \overline{S}_n .$$

A *Monte-Carlo simulation procedure* of the multivariate probability distribution function value based on the estimation of the differences above will be given. First however the so called crude Monte-Carlo simulation procedure will be described. Let the random vectors $(\xi_1^s, \dots, \xi_n^s)$, $s = 1, \dots, S$, be distributed according to the multivariate probability distribution function to be approximated. One must check the inequalities $\xi_1^s < x_1, \dots, \xi_n^s < x_n$ for all sample elements, $s = 1, \dots, S$. For this purpose let be defined the random values

$$\nu_0^s = \begin{cases} 1 & \text{if } \xi_1^s < x_1, \dots, \xi_n^s < x_n, \\ 0 & \text{otherwise,} \end{cases} \quad s = 1, \dots, S.$$

These random values are identically distributed and stochastically independent. All of them take on the value 1 with probability equal to the approximated multivariate probability distribution function value. The sum of them has binomial probability distribution with parameters S and $F(x_1, \dots, x_n)$. So the random variable

$$\nu_0 = \frac{1}{S}(\nu_0^1 + \cdots + \nu_0^S)$$

has expected value $\mathsf{P} = F(x_1, \dots, x_n)$ and variance $\frac{\mathsf{P}(1-\mathsf{P})}{S}$. This is why ν_0 can be regarded as an estimate, the so called crude Monte-Carlo estimate of $F(x_1, \dots, x_n)$. If one introduces κ^s as the number of those $\xi_1^s < x_1, \dots, \xi_n^s < x_n$ inequalities which are not fulfilled, i. e. the number of those $\xi_1^s \geq x_1, \dots, \xi_n^s \geq x_n$ inequalities which are fulfilled, or the number of those $\overline{A}_1^s, \dots, \overline{A}_n^s$ events which occur, $s = 1, \dots, S$, the ν_0^s random values can be expressed as

$$\nu_0^s = \begin{cases} 1 & \text{if } \kappa^s = 0, \\ 0 & \text{otherwise} \end{cases} \quad s = 1, \dots, S,$$

and on the other hand for the *binomial moments* of κ^s one has

$$\mathsf{E}\left[\binom{\kappa^s}{k}\right] = \overline{S}_k, \; k = 0, \dots, n, \quad s = 1, \dots, S.$$

The simplest proof of these equalities was given by L. Takács [17] and it was reproduced by A. Prékopa in [11]. If the random numbers λ^s, $s = 1, \dots, S$, are also introduced as the number of those $\overline{A}_i \cap \overline{A}_j = \{\xi_i^s \geq x_i, \xi_j^s \geq x_j\}$, $(i,j) \in T^*$, events which occur then for the expected value of λ^s one has

$$\mathsf{E}(\lambda^s) = \sum_{(i,j)\in T^*} \mathsf{P}(\overline{A}_i \cap \overline{A}_j), \quad s = 1, \dots, S.$$

Using these equalities one easily can see that the following random values have expected values equal to the differences between the multivariate probability distribution function and its bounds:

$$\nu_{L_1}^s = \binom{\kappa^s}{2} - \binom{\kappa^s}{3} + \cdots + (-1)^n \binom{\kappa^s}{n},$$
$$\nu_{L_2}^s = \left(1 - \frac{2}{n}\right)\binom{\kappa^s}{2} - \binom{\kappa^s}{3} + \cdots + (-1)^n \binom{\kappa^s}{n},$$
$$\nu_{L_3}^s = -\lambda^s + \binom{\kappa^s}{2} - \binom{\kappa^s}{3} + \cdots + (-1)^n \binom{\kappa^s}{n},$$
$$\nu_{U_1}^s = -\binom{\kappa^s}{3} + \cdots + (-1)^n \binom{\kappa^s}{n},$$
$$\nu_{U_2}^s = \left(\frac{2}{k^*+1} - 1\right)\binom{\kappa^s}{1} + \left(1 - \frac{2}{k^*(k^*+1)}\right)\binom{\kappa^s}{2} - \binom{\kappa^s}{3} + \cdots + (-1)^n \binom{\kappa^s}{n}.$$

By the binomial theorem one has

$$\binom{\kappa^s}{0} - \binom{\kappa^s}{1} + \cdots + (-1)^n \binom{\kappa^s}{n} = 0$$

and the above random values can be expressed as

$$\nu_{L_1}^s = \begin{cases} \kappa^s - 1 & \text{if } \kappa^s \geq 2, \\ 0 & \text{otherwise,} \end{cases}$$

$$\nu_{L_2}^s = \begin{cases} \frac{1}{n}(\kappa^s - 1)(n - \kappa^s) & \text{if } \kappa^s \geq 2, \\ 0 & \text{otherwise,} \end{cases}$$

$$\nu_{L_3}^s = \begin{cases} \kappa^s - 1 - \lambda^s & \text{if } \kappa^s \geq 2, \\ 0 & \text{otherwise,} \end{cases}$$

$$\nu_{U_1}^s = \begin{cases} \frac{1}{2}(\kappa^s - 1)(2 - \kappa^s) & \text{if } \kappa^s \geq 3, \\ 0 & \text{otherwise,} \end{cases}$$

$$\nu_{U_2}^s = \begin{cases} \frac{(k^* - \kappa^s)(\kappa^s - k^* - 1)}{k^*(k^* + 1)} & \text{if } \kappa^s \geq 1, \\ 0 & \text{otherwise.} \end{cases}$$

Taking the new random values ν_{L_1}, ν_{L_2}, ν_{L_3}, ν_{U_1}, ν_{U_2} and the estimate ν_0 introduced before:

$$\nu_0 = \frac{1}{S}(\nu_0^1 + \cdots + \nu_0^S),$$

$$\nu_{L_1} = L_1 + \frac{1}{S}(\nu_{L_1}^1 + \cdots + \nu_{L_1}^S),$$

$$\nu_{L_2} = L_2 + \frac{1}{S}(\nu_{L_2}^1 + \cdots + \nu_{L_2}^S),$$

$$\nu_{L_3} = L_3 + \frac{1}{S}(\nu_{L_3}^1 + \cdots + \nu_{L_3}^S),$$

$$\nu_{U_1} = U_1 + \frac{1}{S}(\nu_{U_1}^1 + \cdots + \nu_{U_1}^S),$$

$$\nu_{U_2} = U_2 + \frac{1}{S}(\nu_{U_2}^1 + \cdots + \nu_{U_2}^S),$$

one gets altogether six estimates of the multivariate probability distribution function. These estimates obviously are not stochastically independent so one can mix them to get a new estimate with minimal possible variance. This technique is called *regression method* and it means forming the estimate

$$\nu = w_0\nu_0 + w_{L_1}\nu_{L_1} + w_{L_2}\nu_{L_2} + w_{L_3}\nu_{L_3} + w_{U_1}\nu_{U_1} + w_{U_2}\nu_{U_2}$$

with $w_0 + w_{L_1} + w_{L_2} + w_{L_3} + w_{U_1} + w_{U_2} = 1$, where w_0, w_{L_1}, w_{L_2}, w_{L_3}, w_{U_1}, w_{U_2} are chosen so that the variance of ν be minimized. Let

$$\begin{pmatrix} c_{00} & c_{0L_1} & c_{0L_2} & c_{0L_3} & c_{0U_1} & c_{0U_2} \\ c_{L_10} & c_{L_1L_1} & c_{L_1L_2} & c_{L_1L_3} & c_{L_1U_1} & c_{L_1U_2} \\ c_{L_20} & c_{L_2L_1} & c_{L_2L_2} & c_{L_2L_3} & c_{L_2U_1} & c_{L_2U_2} \\ c_{L_30} & c_{L_3L_1} & c_{L_3L_2} & c_{L_3L_3} & c_{L_3U_1} & c_{L_3U_2} \\ c_{U_10} & c_{U_1L_1} & c_{U_1L_2} & c_{U_1L_3} & c_{U_1U_1} & c_{U_1U_2} \\ c_{U_20} & c_{U_2L_1} & c_{U_2L_2} & c_{U_2L_3} & c_{U_2U_1} & c_{U_2U_2} \end{pmatrix}$$

be the covariance matrix C of the six estimates, where C is a symmetrical matrix. Then the variance of ν is $w^\mathsf{T} Cw$, where $w = (w_0, w_{L_1}, w_{L_2}, w_{L_3}, w_{U_1}, w_{U_2})^\mathsf{T}$. The Lagrangian problem:

$$\begin{cases} \min & w^\top Cw \\ \text{s.t.} & w_0 + w_{L_1} + w_{L_2} + w_{L_3} + w_{U_1} + w_{U_2} = 1 \end{cases}$$

can easily be solved. In fact, the gradient of $w^\mathsf{T}\, Cw$ equals $2\, w^\mathsf{T}\, C$, hence one has to solve the system of linear equations

$$\begin{aligned} &c_{00}w_0 + c_{0L_1}w_{L_1} + c_{0L_2}w_{L_2} + c_{0L_3}w_{L_3} \\ &+ c_{0U_1}w_{U_1} + c_{0U_2}w_{U_2} - \lambda = 0, \\ &c_{L_10}w_0 + c_{L_1L_1}w_{L_1} + c_{L_1L_2}w_{L_2} + c_{L_1L_3}w_{L_3} \\ &+ c_{L_1U_1}w_{U_1} + c_{L_1U_2}w_{U_2} - \lambda = 0, \\ &c_{L_20}w_0 + c_{L_2L_1}w_{L_1} + c_{L_2L_2}w_{L_2} + c_{L_2L_3}w_{L_3} \\ &+ c_{L_2U_1}w_{U_1} + c_{L_2U_2}w_{U_2} - \lambda = 0, \\ &c_{L_30}w_0 + c_{L_3L_1}w_{L_1} + c_{L_3L_2}w_{L_2} + c_{L_3L_3}w_{L_3} \\ &+ c_{L_3U_1}w_{U_1} + c_{L_3U_2}w_{U_2} - \lambda = 0, \\ &c_{U_10}w_0 + c_{U_1L_1}w_{L_1} + c_{U_1L_2}w_{L_2} + c_{U_1L_3}w_{L_3} \\ &+ c_{U_1U_1}w_{U_1} + c_{U_1U_2}w_{U_2} - \lambda = 0, \\ &c_{U_20}w_0 + c_{U_2L_1}w_{L_1} + c_{U_2L_2}w_{L_2} + c_{U_2L_3}w_{L_3} \\ &+ c_{U_2U_1}w_{U_1} + c_{U_2U_2}w_{U_2} - \lambda = 0, \\ &w_0 + w_{L_1} + w_{L_2} + w_{L_3} + w_{U_1} + w_{U_2} - \lambda = 1. \end{aligned}$$

for the unknowns w_0, w_{L_1}, w_{L_2}, w_{L_3}, w_{U_1}, w_{U_2}, λ. As the covariance matrix C is not known in advance, so one must estimate its elements from the random sample during the Monte-Carlo simulation procedure. This means that one must sum up not only the individual random values ν_0^s, $\nu_{L_1}^s$, $\nu_{L_2}^s$, $\nu_{L_3}^s$, $\nu_{U_1}^s$, $\nu_{U_2}^s$ but their crossproducts, too. The crossproducts are many times trivial, so their calculation is not necessary. For example ν_0^s equals $\nu_0^s\nu_0^s$, further when ν_0^s equals nonzero ($\kappa^s = 0$) then all other random values $\nu_{L_1}^s$, $\nu_{L_2}^s$, $\nu_{L_3}^s$, $\nu_{U_1}^s$, $\nu_{U_2}^s$ are equal zero, so the corresponding crossproducts are all zero. One should also notice that the random values $\nu_{L_1}^s$, $\nu_{L_2}^s$, $\nu_{L_3}^s$ are always nonnegative while the random values $\nu_{U_1}^s$, $\nu_{U_2}^s$ are always nonpositive. So the corresponding crossproducts cannot be positive even they are many times negative yielding real variance reduction in the final estimate.

One- and Two-Dimensional Marginal Distribution Functions

For the applicability of the Monte-Carlo simulation algorithm of the previous section one has to show that the one- and two-dimensional marginal distribution function values can be evaluated efficiently. As in the cases of the *multivariate normal distribution*, (one parameter) gamma and Dirichlet distributions the marginal distributions are also normal, gamma and Dirichlet and the one-dimensional Dirichlet distribution is the beta distribution, the one-dimensional marginal probability distribution functions can be evaluated by known algorithms. For example in the *IMSL subroutine library* [6] the subroutines MDNOR, MDGAM and MDBETA provide these calculations. In the case of the normal distribution the two-dimensional marginal probability distribution function also can be evaluated by a standard IMSL subroutine called MDBNOR. Some details of the calculations provided by these subroutines can be found in [8].

In the case of the *multivariate gamma distribution*, introduced by Prékopa and T. Szántai in [12], only the evaluation of the joint probability distribution function of the random variables

$$\xi_1 = \eta_1 + \eta_2,$$
$$\xi_2 = \eta_1 + \eta_3$$

is necessary. Here the random variables η_1, η_2 and η_3 are independent and gamma distributed with parameters ϑ_1, ϑ_2 and ϑ_3. Taking the joint characteristic function of ξ_1 and ξ_2 and applying the inversion formula one easily gets the joint probability density function of them. This is in the form of series expansion involving Laguerre polynomials. Using some integral formulae of these orthogonal polynomials one can integrate the joint probability density function to get the final formula for the evaluation of the joint probability distribution function in the following form

$$\begin{aligned} F(z_1, z_2) = {} & F_{\vartheta_1+\vartheta_2}(z_1) F_{\vartheta_1+\vartheta_3}(z_2) \\ & + \sum_{k=1}^{\infty} C(\vartheta_1, \vartheta_2, \vartheta_3, k) \\ & \times f_{\vartheta_1+\vartheta_2+1}(z_1) L_{k-1}^{\vartheta_1+\vartheta_2}(z_1) \\ & \times f_{\vartheta_1+\vartheta_3+1}(z_2) L_{k-1}^{\vartheta_1+\vartheta_3}(z_2), \end{aligned}$$

where

$$\begin{aligned} C(\vartheta_1, \vartheta_2, \vartheta_3, k) = {} & \frac{(k-1)!}{k} \frac{\Gamma(\vartheta_1 + k)}{\Gamma(\vartheta_1)} \\ & \times \frac{\Gamma(\vartheta_1 + \vartheta_2 + 1)}{\Gamma(\vartheta_1 + \vartheta_2 + k)} \frac{\Gamma(\vartheta_1 + \vartheta_3 + 1)}{\Gamma(\vartheta_1 + \vartheta_3 + k)} \end{aligned}$$

and $f_\vartheta(z)$ and $F_\vartheta(z)$ are the one-dimensional gamma probability density, respectively distribution, functions. For the calculation of the Laguerre polynomial the following recursion formula can be used

$$\begin{aligned} & (k+1) L_{k+1}^{\vartheta}(z) \\ & = (2k + \vartheta + 1 - z) L_k^{\vartheta}(z) - (k + \vartheta) L_{k-1}^{\vartheta}(z), \\ & k = 0, 1, \ldots, \end{aligned}$$

where $L_0^\vartheta(z) = 1$ and $L_1^\vartheta(z) = \vartheta + 1 - z$. The convergence of the series for calculation of $F(z_1, z_2)$ has been established by Szántai in [14].

In the case of *Dirichlet distribution* the two-dimensional marginal probability density function of the components ξ_i, ξ_j is given by

$$f(z_1, z_2) = \frac{\Gamma(a)\Gamma(b)\Gamma(c)}{\Gamma(a+b+c)} \cdot z_1^{a-1} z_2^{b-1} (1 - z_1 - z_2)^{c-1},$$
$$\text{if } z_1 + z_2 \le 1,\ z_1 \ge 0,\ z_2 \ge 0,$$

where $a = \vartheta_i$, $b = \vartheta_j$ and $c = \sum_{k=1}^{n+1} \vartheta_k - \vartheta_i - \vartheta_j$. One obtains by direct calculation for the two-dimensional probability distribution function

$$\begin{aligned} F(z_1, z_2) = {} & \frac{\Gamma(a+b+c)}{\Gamma(a)\Gamma(b)\Gamma(c)} \\ & \times \int_0^{z_1} \int_0^{z_2} t_1^{a-1} t_2^{b-1} (1 - t_1 - t_2)^{c-1} \, dt_2 \, dt_1 \\ = {} & \frac{z_1^a}{a} \frac{z_2^b}{b} + \sum_{m=1}^{\infty} (1-c) \cdots (m-c) \\ & \times \left[\frac{z_1^a}{a} \frac{z_2^{b+m}}{(b+m)m!} \right. \\ & \left. + \sum_{k=0}^{m} \frac{z_1^{a+k}}{(a+k)k!} \frac{z_2^{b+m-k}}{(b+m-k)(m-k)!} \right]. \end{aligned}$$

The above formula is valid only if $z_1 + z_2 \le 1$, $z_1 \ge 0$, $z_2 \ge 0$; otherwise the statement a) of the following more general theorem can be applied.

Theorem 1 *Let $z_1^* \le \cdots \le z_n^*$ be the ordered sequence of $z_1, \ldots, z_n$, the arguments of the n-dimensional Dirichlet distribution function.*

a) If $z_1^* + z_2^* > 1$ then one has

$$F(z_1, \ldots, z_n) = 1 - n + \sum_{i=1}^{n} F_i(z_i).$$

b) If $z_1^* + z_2^* + z_3^* > 1$ then one has

$$F(z_1, \ldots, z_n) = \frac{n-1}{2} - (n-2) \sum_{i=1}^{n} F_i(z_i) + \sum_{1 \le i < j \le n} F_{ij}(z_i, z_j).$$

Here $F_i(z_i)$ and $F_{ij}(z_i, z_j)$ are the one- and two-dimensional marginal probability distribution functions.

This theorem was formulated and proved by Szántai in [13]. It also can be found in [11].

Examples

For illustrating the lower and upper bounds on the multivariate normal probability distribution function value and the efficiency of the *variance reduction technique* described before one can regard the following examples.

Example 2

$$\begin{aligned}
&n = 10,\\
&x_1 = 1.7, \quad x_2 = 0.8, \quad x_3 = 5.1,\\
&x_4 = 3.2, \quad x_5 = 2.4, \quad x_6 = 1.8,\\
&x_7 = 2.7, \quad x_8 = 1.5, \quad x_9 = 1.2,\\
&x_{10} = 2.6,\\
&r_{ij} = 0.0, \quad i = 2, \ldots, 10, \quad j = 1, \ldots, i-1,
\end{aligned}$$

except $r_{21} = -0.6$, $r_{43} = 0.9$, $r_{65} = 0.4$, $r_{87} = 0.2$, $r_{10,9} = -0.8$.

Number of trials: 10000.

Lower bound by S1, S2	0.524736
Lower bound by Hunter	0.563719
Upper bound by S1, S2	0.588646
Estimated value	0.582743
Standard deviation	0.000608
Time in seconds (PC-586)	0.77
Efficiency	65.73

Example 3

$$\begin{aligned}
&n = 15,\\
&x_1 = 2.9, \quad x_2 = 2.9, \quad x_3 = 2.9,\\
&x_4 = 2.9, \quad x_5 = 2.9, \quad x_6 = 2.9,\\
&x_7 = 2.9, \quad x_8 = 2.9, \quad x_9 = 2.9,\\
&x_{10} = 2.9, \quad x_{11} = 2.9, \quad x_{12} = 2.7\\
&x_{13} = 1.6, \quad x_{14} = 1.2, \quad x_{15} = 2.1,\\
&r_{ij} = 0.2, \quad i = 2, \ldots, 10, \quad j = 1, \ldots, i-1,\\
&r_{ij} = 0.0, \quad i = 11, \ldots, 15, \quad j = 1, \ldots, i-1
\end{aligned}$$

except $r_{13,12} = 0.3$, $r_{15,14} = -0.95$.

Number of trials = 10000.

Lower bound by S1, S2	0.790073
Lower bound by Hunter	0.798730
Upper bound by S1, S2	0.801745
Estimated value	0.801304
Standard deviation	0.000193
Time in seconds (PC-586)	1.38
Efficiency	417.84

Both of the above examples are taken from [2, Exam. 4; 6] and they are according to standard multivariate normal probability distributions, i. e. all components of the normally distributed random vector have expected value zero and variance one. The efficiency of the Monte-Carlo simulation algorithm was calculated according to the crude Monte-Carlo algorithm in the usual way, i. e. it equals to the fraction $(t_0\sigma_0^2)/(t_1\sigma_1^2)$ where t_0, t_1 are the calculation times and σ_0^2, σ_1^2 are the variances of the crude and the compared simulation algorithms.

Remarks

In many applications one may need finding the *gradient of multivariate distribution functions*, too. As one has the general formula

$$\frac{\partial F(z_1, \ldots, z_n)}{\partial z_i} = F(z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_n | z_i) \cdot f_i(z_i),$$

where $F(z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_n \mid z_i)$ is the conditional probability distribution function of the random variables $\xi_1, \ldots, \xi_{i-1}, \xi_{i+1}, \ldots, \xi_n$, given that $\xi_i = z_i$, and

$f_i(z)$ is the probability density function of the random variable ξ_i, finding the gradient of a multivariate probability distribution function can be reduced to finding conditional distribution functions. In the cases of multivariate normal and Dirichlet distributions the conditional distributions are also multivariate normal and Dirichlet, and in the case of multivariate gamma distribution they are different and more complicated as it was obtained by Prékopa and Szántai [12].

In the case of multivariate normal probability distribution I. Deák [2] proposed another simulation technique which proved to be as efficient as the method described here. The main advantage of Deák's method is that it easily can be generalized for calculation the probability content of more general sets in the multidimensional space, like convex polyhedrons, hyperellipsoids, circular cones, etc. Its main drawback is that it works only for the multivariate normal probability distribution. The methods of Szántai and Deák have been combined by H. Gassmann to compute the probability of an n-dimensional rectangle in the case of multivariate normal distribution (see [3]). Also in the case of multivariate normal probability distribution A. Genz proposed the transformation of the original integration region to the unit hypercube $[0, 1]^n$ and then the application of a crude Monte-Carlo method or some lattice rules for the numerical integration of the resulting multidimensional integral. A comparison of methods for the computation of multivariate normal probabilities can be found in [4]. When the three-dimensional marginal probability distribution function values are also calculated by numerical integration there exist some new, sharper bounds. See [16] for these bounds and their effect on the efficiency of the Monte-Carlo simulation algorithm.

Approximation of multivariate probability integrals has a central role in probabilistic constrained stochastic programming when the probabilistic constraints are joint. The *computer code PCSP* (probabilistic constrained stochastic programming) originally was developed for handling the multivariate normal probability distributions in this framework (see [15]). A new version of the code now can handle multivariate gamma and Dirichlet distributions as well. The calculation procedures of this paper also has been applied by J. Mayer in his code solving this type of stochastic programming problems by reduced gradient algorithm (see [10]). These codes have been integrated by P. Kall and Mayer into a more advanced computer system for modeling in *stochastic linear programming* (see [7]).

See also

- ▶ Approximation of Extremum Problems with Probability Functionals
- ▶ Discretely Distributed Stochastic Programs: Descent Directions and Efficient Points
- ▶ Extremum Problems with Probability Functions: Kernel Type Solution Methods
- ▶ General Moment Optimization Problems
- ▶ Logconcave Measures, Logconvexity
- ▶ Logconcavity of Discrete Distributions
- ▶ L-shaped Method for Two-stage Stochastic Programs with Recourse
- ▶ Multistage Stochastic Programming: Barycentric Approximation
- ▶ Preprocessing in Stochastic Programming
- ▶ Probabilistic Constrained Linear Programming: Duality Theory
- ▶ Probabilistic Constrained Problems: Convexity Theory
- ▶ Simple Recourse Problem: Dual Method
- ▶ Simple Recourse Problem: Primal Method
- ▶ Stabilization of Cutting Plane Algorithms for Stochastic Linear Programming Problems
- ▶ Static Stochastic Programming Models
- ▶ Static Stochastic Programming Models: Conditional Expectations
- ▶ Stochastic Integer Programming: Continuity, Stability, Rates of Convergence
- ▶ Stochastic Integer Programs
- ▶ Stochastic Linear Programming: Decomposition and Cutting Planes
- ▶ Stochastic Linear Programs with Recourse and Arbitrary Multivariate Distributions
- ▶ Stochastic Network Problems: Massively Parallel Solution
- ▶ Stochastic Programming: Minimax Approach
- ▶ Stochastic Programming Models: Random Objective
- ▶ Stochastic Programming: Nonanticipativity and Lagrange Multipliers
- ▶ Stochastic Programming with Simple Integer Recourse
- ▶ Stochastic Programs with Recourse: Upper Bounds

- ▶ Stochastic Quasigradient Methods in Minimax Problems
- ▶ Stochastic Vehicle Routing Problems
- ▶ Two-stage Stochastic Programming: Quasigradient Method
- ▶ Two-stage Stochastic Programs with Recourse

References

1. Bonferroni CE (1937) Teoria statistica delle classi e calcolo delle probabilita. Volume in onordi Riccardo Dalla Volta: 1–62
2. Deák I (1980) Three digit accurate multiple normal probabilities. Numerische Math 35:369–380
3. Gassmann H (1988) Conditional probability and conditional expectation of a random vector. In: Ermoliev Y, Wets RJ-B (eds) Numerical Techniques for Stochastic Optimization. Springer, Berlin, pp 237–254
4. Genz A (1993) Comparison of methods for the computation of multivariate normal probabilities. Computing Sci and Statist 25:400–405
5. Hunter D (1976) Bounds for the probability of a union. J Appl Probab 13:597–603
6. IMSL (1977) Library 1 reference manual. Internat. Math. Statist. Library
7. Kall P, Mayer J (1995) Computer support for modeling in stochastic linear programming. In: Marti K, Kall P (eds) Stochastic Programming: Numerical Methods and Techn. Applications. Springer, Berlin, pp 54–70
8. Kennedy WJ Jr, Gentle JE (1980) Statistical computing. M. Dekker, New York
9. Kruskal JB (1956) On the shortest spanning subtree of a graph and the travelling salesman problem. Proc Amer Math Soc 7:48–50
10. Mayer J (1988) Probabilistic constrained programming: A reduced gradient algorithm implemented on PC. Working Papers IIASA WP-88-39
11. Prékopa A (1995) Stochastic programming. Akad. Kiadó and Kluwer, Budapest–Dordrecht
12. Prékopa A, Szántai T (1978) A new multivariate gamma distribution and its fitting to empirical streamflow data. Water Resources Res 14:19–24
13. Szántai T (1985) Numerical evaluation of probabilities concerning multivariate probability distributions. Thesis Candidate Degree Hungarian Acad Sci (in Hungarian)
14. Szántai T (1986) Evaluation of a special multivariate gamma distribution function. Math Program Stud 27:1–16
15. Szántai T (1988) A computer code for solution of probabilistic-constrained stochastic programming problems. In: Ermoliev Y, Wets RJ-B (eds) Numerical Techniques for Stochastic Optimization. Springer, Berlin, pp 229–235
16. Szántai T: Improved bounds and simulation procedures on the value of multivariate normal probability distribution functions. Ann Oper Res (to appear) Special Issue: Research in Stochastic Programming (Selected refereed papers from the VII Internat. Conf. Stochastic Programming, Aug. 10–14, Univ. British Columbia, Vancouver, Canada).
17. Takács L (1955) On the general probability theorem. Comm Dept Math Physics Hungarian Acad Sci 5:467–476 (In Hungarian.)
18. Worsley KJ (1982) An improved Bonferroni inequality and applications. Biometrika 69:297–302

Approximations to Robust Conic Optimization Problems

MELVYN SIM
NUS Business School, National University of Singapore, Singapore, Republic of Singapore

Article Outline

Introduction

We consider a general conic optimization problem under parameter uncertainty is as follows:

$$\begin{aligned} \max \quad & \boldsymbol{c}'\boldsymbol{x} \\ \text{s.t.} \quad & \sum_{j=1}^{n} \tilde{\boldsymbol{A}}_j x_j - \tilde{\boldsymbol{B}} \in \mathcal{K} \\ & \boldsymbol{x} \in X\,, \end{aligned} \tag{1}$$

where the cone $\mathcal{K}$ is a regular cone, i. e., a closed, convex and pointed cone. The space of the data $(\tilde{\boldsymbol{A}}_1, \ldots, \tilde{\boldsymbol{A}}_n, \tilde{\boldsymbol{B}})$ depends on the cone, $\mathcal{K}$. The most common cone is the cone of non-negative orthant, $\Re^m_+$ in which the conic constraint in Problem (1) becomes a set of m linear constraints. Two important cones, which have many applications, include the second-order cone,

$$\mathcal{L}^{m+1} = \{(y_0, \boldsymbol{y}) \colon \|\boldsymbol{y}\|_2 \le y_0,\ \boldsymbol{y} \in \Re^m\}$$

and the cone of symmetric positive semidefinite matrix,

$$S^m = \{\boldsymbol{Y}\colon \boldsymbol{Y} \text{ is a symmetric postive semidefinite matrix}\}\,.$$

The interested reader may refer to the references of Ben-Tal and Nemirovski [3] and Pardalos and Wolkowicz [13].

In the uncertain conic optimization problem (1), the data $(\tilde{\boldsymbol{A}}_1, \ldots, \tilde{\boldsymbol{A}}_n, \tilde{\boldsymbol{B}})$ are uncertain. It is therefore conceivable that as the data take values different than the nominal ones, the conic constraint may be violated, and the optimal solution found using the nominal data may no longer be feasible at the conic constraint. To control the feasibility level of the conic constraint, one may consider a conic chance constrained model as follows:

$$\begin{aligned} \max \quad & \boldsymbol{c}'\boldsymbol{x} \\ \text{s.t.} \quad & \mathrm{P}\Big(\sum_{j=1}^{n} \tilde{\boldsymbol{A}}_j x_j - \tilde{\boldsymbol{B}} \in \mathcal{K}\Big) \geq 1-\epsilon \\ & \boldsymbol{x} \in X\,, \end{aligned} \tag{2}$$

in which the level of constraint violation is controlled probabilistically. Unfortunately, the chance constrained conic optimization problem (2) destroys the convexity of the problem and hence its computational tractability.

Formulation

In modern robust optimization, we represent data uncertainty using uncertainty sets instead of probability distributions. We allow the data $(\tilde{\boldsymbol{A}}_1, \ldots, \tilde{\boldsymbol{A}}_n, \tilde{\boldsymbol{B}})$ to vary within an uncertainty set $\mathcal{U}$ without having to violate the conic constraint. We call the following problem the *robust counterpart* of Problem (1)

$$\begin{aligned} \max \quad & \boldsymbol{c}'\boldsymbol{x} \\ \text{s.t.} \quad & \sum_{j=1}^{n} \boldsymbol{A}_j x_j - \boldsymbol{B} \in \mathcal{K} \\ & \quad \forall (\boldsymbol{A}_1, \ldots, \boldsymbol{A}_n, \boldsymbol{B}) \in \mathcal{U} \\ & \boldsymbol{x} \in X\,. \end{aligned} \tag{3}$$

The robust counterpart is introduced by Ben-Tal and Nemirovski [1] and independently by El-Ghoui et al. [9]. An immediate consequence of the robust counterpart is the preservation of the convexity. Unfortunately, due to the possibly infinite number of scenarios corresponding to the extreme points of the uncertainty set $\mathcal{U}$, optimizing the robust counterpart for general conic optimization problems is intractable.

It is noteworthy that in robust optimization, the ellipsoidal uncertainty set is a popular choice because of the motivation from the laws of large numbers and normal distributions. Under the assumption of normality, we could design an ellipsoidal set that is large enough so that the robust model will remain feasible with high probability. However, it turns out this approach can grossly over estimate the size of ellipsoid necessary to ensure the same level of robustness. To illustrate this issue, consider a linear constraint $\tilde{\boldsymbol{a}}'\boldsymbol{x} \geq b$ such that $\tilde{\boldsymbol{a}}$ is a multivariate normal with mean $\bar{\boldsymbol{a}}$ and covariance $\sum$. It is natural to design an ellipsoidal uncertainty set of the form $\mathcal{U} = \{\boldsymbol{a}\colon (\boldsymbol{a}-\bar{\boldsymbol{a}})\Sigma^{-1}(\boldsymbol{a}-\bar{\boldsymbol{a}}) \leq \alpha^2\}$ so that the problem remains feasible if $\tilde{\boldsymbol{a}} \in \mathcal{U}$, which has a probability of $\chi_n^2(\alpha^2)$. However, when solving the equivalent robust counterpart, $\bar{\boldsymbol{a}}'\boldsymbol{x} - \alpha\sqrt{\boldsymbol{x}'\Sigma\boldsymbol{x}} \geq b$, the robust solution has a feasibility probability of at least $\Phi(\alpha)$, where $\Phi(\alpha)$ is the standard normal function. Clearly, the value $\chi_n^2(\alpha^2)$ would be a gross over estimate of the robustness of the uncertain linear constraint compared to the value $\Phi(\alpha)$. The reason for this disparity is the fact that the uncertainty set chosen does not take into account the structure of cone.

We focus on the robust optimization framework proposed by Bertsimas and Sim [5], which offers a simple and tractable approximation of uncertain conic optimization problems. Moreover, under reasonable probabilistic assumptions on data variation, the framework approximates the conic chance constraint problem (2) by relating its feasibility probability with the size of the uncertainty set and the structure of the cone. Note that more refined approximations of chance constrained problem are available for the case of linear cones, $\mathcal{K} = \Re_+^m$. Interested readers can refer to Ben-Tal and Nemirovski [2], Bertsimas and Sim [4], Chen, Sim and Sun [8], Chen and Sim [6], Chen et al.[7], Lin et al. [10] and Janak et al. [11].

Affine Data Dependency

We first assume that uncertain data $(\tilde{\boldsymbol{A}}_1, \ldots, \tilde{\boldsymbol{A}}_n, \tilde{\boldsymbol{B}})$ are affinely dependent on some primitive uncertainty

vector, $\tilde{\boldsymbol{z}} \in \Re^N$, as follows

$$\tilde{\boldsymbol{A}}_i = \boldsymbol{A}_i(\tilde{\boldsymbol{z}}) \triangleq \boldsymbol{A}_i^0 + \sum_{j=1}^{N} \boldsymbol{A}_i^j \tilde{z}_j \quad i = 1, \ldots, n$$
$$\tilde{\boldsymbol{B}} = \boldsymbol{B}(\tilde{\boldsymbol{z}}) \triangleq \boldsymbol{B}^0 + \sum_{j=1}^{N} \boldsymbol{B}^j \tilde{z}_j \,.$$

Note that we can always define a bijection mapping from a vector space of $\tilde{\boldsymbol{z}}$ to the data space of $(\tilde{\boldsymbol{A}}_1, \ldots, \tilde{\boldsymbol{A}}_n, \tilde{\boldsymbol{B}})$. Therefore, under the affine data dependency, it is always possible to map all the data uncertainties affecting the conic constraint to the primitive uncertainty vector, $\tilde{\boldsymbol{z}}$. It is more convenient to define the following linear function mapping with respect to $(z_0, \boldsymbol{z})$,

$$\boldsymbol{Y}((z_0, \boldsymbol{z})) = \sum_{j=0}^{N} \boldsymbol{Y}_j z_j \,,$$

in which the variables $\boldsymbol{x}$ are affinely mapped to the variables $(\boldsymbol{Y}_0, \ldots, \boldsymbol{Y}_N)$ as follows

$$\boldsymbol{Y}_j = \sum_{i=1}^{n} \boldsymbol{A}_i^j x_i - \boldsymbol{B}^j \quad \forall j = 0, \ldots, N \,.$$

For instance, under such transformation, Problem (2) is equivalent to

$$\begin{array}{ll} \max & \boldsymbol{c}'\boldsymbol{x} \\ \text{s.t.} & \boldsymbol{Y}_j = \sum_{i=1}^{n} \boldsymbol{A}_i^j x_i - \boldsymbol{B}^j \qquad \forall j = 0, \ldots, N \\ & \mathrm{P}(\boldsymbol{Y}((1, \tilde{\boldsymbol{z}})) \in \mathcal{K}) \geq 1 - \epsilon \\ & \boldsymbol{x} \in X \,, \end{array} \tag{4}$$

and Problem (3) is the same as

$$\begin{array}{lll} \max & \boldsymbol{c}'\boldsymbol{x} & \\ \text{s.t.} & \boldsymbol{Y}_j = \sum_{i=1}^{n} \boldsymbol{A}_i^j x_i - \boldsymbol{B}^j & \forall j = 0, \ldots, N \\ & \boldsymbol{Y}((1, \tilde{\boldsymbol{z}})) \in \mathcal{K} & \forall \boldsymbol{z} \in \mathcal{V} \\ & \boldsymbol{x} \in X \,, & \end{array} \tag{5}$$

in which the uncertainty set $\mathcal{U}$ is mapped accordingly to the uncertainty set $\mathcal{V}$.

Example: Quadratic Chance Constraint Consider the following quadratic chance constraint,

$$\mathrm{P}(\|\boldsymbol{A}(\tilde{\boldsymbol{z}})\boldsymbol{x}\|_2^2 + \boldsymbol{b}(\tilde{\boldsymbol{z}})'\boldsymbol{x} + c(\tilde{\boldsymbol{z}}) \leq 0) \geq 1 - \epsilon \,,$$

where $\boldsymbol{x} \in \Re^n$ is the decision variable and $(\boldsymbol{A}(\tilde{\boldsymbol{z}}),\ \boldsymbol{b}(\tilde{\boldsymbol{z}}),\ c(\tilde{\boldsymbol{z}})) \in \Re^{m \times n} \times \Re^n \times \Re$ are the input data, which are affinely dependent on its primitive uncertainties as follows:

$$\begin{array}{rcl} \boldsymbol{A}(\tilde{\boldsymbol{z}}) & \triangleq & \boldsymbol{A}^0 + \sum_{j=1}^{N} \boldsymbol{A}^j \tilde{z}_j \\ \boldsymbol{b}(\tilde{\boldsymbol{z}}) & \triangleq & \boldsymbol{b}^0 + \sum_{j=1}^{N} \boldsymbol{b}^j \tilde{z}_j \\ c(\tilde{\boldsymbol{z}}) & \triangleq & c^0 + \sum_{j=1}^{N} c^j \tilde{z}_j \,. \end{array}$$

Note that a quadratic constraint

$$\|\boldsymbol{A}(\tilde{\boldsymbol{z}})\boldsymbol{x}\|_2^2 + \boldsymbol{b}(\tilde{\boldsymbol{z}})'\boldsymbol{x} + c(\tilde{\boldsymbol{z}}) \leq 0$$

is second-order cone representable as follows

$$\begin{bmatrix} \frac{1 - \boldsymbol{b}(\tilde{\boldsymbol{z}})'\boldsymbol{x} - c(\tilde{\boldsymbol{z}})}{2} \\ \boldsymbol{A}(\tilde{\boldsymbol{z}})\boldsymbol{x} \\ \frac{1 + \boldsymbol{b}(\tilde{\boldsymbol{z}})'\boldsymbol{x} + c(\tilde{\boldsymbol{z}})}{2} \end{bmatrix} \in \mathcal{L}^{m+2} \,.$$

Therefore, under the affine relation,

$$\boldsymbol{y}_0 = \begin{bmatrix} \boldsymbol{A}^0\boldsymbol{x} \\ \frac{1 + \boldsymbol{b}^{0\prime}\boldsymbol{x} + c^0}{2} \\ \frac{1 - \boldsymbol{b}^{0\prime}\boldsymbol{x} - c^0}{2} \end{bmatrix},$$

and

$$\boldsymbol{y}_j = \begin{bmatrix} \boldsymbol{A}^j\boldsymbol{x} \\ \frac{\boldsymbol{b}^{j\prime}\boldsymbol{x} + c^j}{2} \\ \frac{-\boldsymbol{b}^{j\prime}\boldsymbol{x} - c^j}{2} \end{bmatrix} \quad \forall j = 1, \ldots, N$$

we transform the quadratic chance constraint problem into the following conic chance constraint

$$\mathrm{P}\left(\boldsymbol{y}_0 + \sum_{j=1}^{n} \boldsymbol{y}_j \tilde{z}_j \in \mathcal{L}^{m+2}\right) .$$

Hence, we treat the quadratic constraint as a special case of second-order cone constraint.

Tractable Approximations of a Conic Chance Constrained Problem

We focus on deriving a tractable approximation on the following conic chance constraint:

$$\mathrm{P}(\boldsymbol{Y}((1, \tilde{\boldsymbol{z}})) \in \mathcal{K}) \geq 1 - \epsilon \,. \tag{6}$$

For notational convenience, we define

$$\boldsymbol{X} \triangleq (\boldsymbol{Y}_0, \ldots, \boldsymbol{Y}_N) \,.$$

For a given a reference vector (or matrix), $\boldsymbol{V} \in \text{int}(\mathcal{K})$, where int($\mathcal{K}$) denotes the interior of the cone $\mathcal{K}$, we can define the function

$$f(\boldsymbol{X},\,(z_0,\boldsymbol{z})) \triangleq \max\{\theta : \boldsymbol{Y}((z_0,\,\boldsymbol{z})) - \theta \boldsymbol{V} \in \mathcal{K}\}\,,$$

which has the following properties:

Proposition 1 *For any $\boldsymbol{V} \in$ int($\mathcal{K}$), the function $f(\boldsymbol{X},\,(z_0,\,\boldsymbol{z}))$ satisfies the properties:*

(a) $f(\boldsymbol{X},\,(z_0,\,\boldsymbol{z}))$ *is bounded and concave in* $\boldsymbol{X}$ *and* $(z_0,\,\boldsymbol{z})$.

(b) $f(\boldsymbol{X},\,k(z_0,\,\boldsymbol{z})) = kf(\boldsymbol{X},\,(z_0,\,\boldsymbol{z}))\,,\ \forall k \geq 0.$

(c) $f(\boldsymbol{X},\,(z_0,\,\boldsymbol{z})) \geq s$ *if and only if* $\boldsymbol{Y}((z_0,\,\boldsymbol{z})) - s\,\boldsymbol{V} \in \mathcal{K}$.

(d) $f(\boldsymbol{X},\,(z_0,\,\boldsymbol{z})) > s$ *if and only if* $\boldsymbol{Y}((z_0,\,\boldsymbol{z})) - s\,\boldsymbol{V} \in$ *int*($\mathcal{K}$).

Hence, the conic chance constraint of (6) is equivalent to the following chance constraint

$$\mathrm{P}(f(\boldsymbol{X},\,(1,\,\tilde{\boldsymbol{z}})) \geq 0) \geq 1 - \epsilon\,. \tag{7}$$

In order to build a tractable framework that approximates the conic chance constraint problem, we first analyze the robust counterpart approach to uncertainty. Given an ellipsoidal uncertainty set

$$\mathcal{E}(\rho) = \{\boldsymbol{z}\colon \|\boldsymbol{z}\|_2 \leq \rho\}\,,$$

the robust counterpart

$$f(\boldsymbol{X},(1,\boldsymbol{z})) \geq 0 \qquad \forall \boldsymbol{z} \in \mathcal{E}(\rho)\,, \tag{8}$$

despite its convexity, is generally intractable. Instead we consider the following robust counterpart:

$$\begin{aligned} f(\boldsymbol{X},\,(1,\,\boldsymbol{0})) + \sum_{j=1}^{N} \{f(\boldsymbol{X},\,(0,\,\boldsymbol{e}_j))v_j \\ + f(\boldsymbol{X},\,(0,\,-\boldsymbol{e}_j))w_j\} \geq 0, \\ \forall (\boldsymbol{v},\,\boldsymbol{w}) \in \mathcal{V}(\rho) \end{aligned} \tag{9}$$

where $\boldsymbol{e}_j \in \Re^N$ is a unit vector with one at the jth entry and the uncertainty set

$$\mathcal{V}(\rho) = \left\{(\boldsymbol{v},\,\boldsymbol{w}) \in \Re^N_+ \times \Re^N_+ \mid \|\boldsymbol{v} + \boldsymbol{w}\|_2 \leq \rho\right\}\,. \tag{10}$$

Proposition 2 *The robust counterpart (9) is tractable relaxation of the robust counterpart, (8).*

Theorem 1

(a) The constraint (9) is equivalent to

$$f(\boldsymbol{X},\,(1,\,\boldsymbol{0})) \geq \rho\|\boldsymbol{s}\|_2\,, \tag{11}$$

where

$$\begin{aligned} s_j = \max\{-f(\boldsymbol{X},\,(0,\,\boldsymbol{e}_j)),\,-f(\boldsymbol{X},\,(0,\,-\boldsymbol{e}_j))\}, \\ \forall j = 1,\,\ldots\,,\,N\,. \end{aligned}$$

(b) Eq. (11) can be written as:

$$\begin{aligned} & f(\boldsymbol{X},\,(1,\,\boldsymbol{0})) \geq \rho y \\ & f(\boldsymbol{X},\,(0,\,\boldsymbol{e}_j)) + t_j \geq 0, \qquad \forall j \in N \\ & f(\boldsymbol{X},\,(0,\,-\boldsymbol{e}_j)) + t_j \geq 0, \qquad \forall j \in N \\ & \|\boldsymbol{t}\|_2 \leq y \\ & \text{for some } y \in \Re,\ \boldsymbol{t} \in \Re^N\,. \end{aligned} \tag{12}$$

From Proposition 1 and noting that

$$\boldsymbol{Y}((1,\,\boldsymbol{0})) = \boldsymbol{Y}_0$$

and

$$\boldsymbol{Y}((0,\,\pm\boldsymbol{e}_j)) = \pm\boldsymbol{Y}_j\,,$$

we can also represent the formulation (12) explicitly in conic constraints as follows:

$$\begin{aligned} & \boldsymbol{Y}_0 - \rho y \boldsymbol{V} \in \mathcal{K} \\ & \boldsymbol{Y}_j + t_j \boldsymbol{V} \in \mathcal{K}, \qquad \forall j \in N \\ & -\boldsymbol{Y}_j + t_j \boldsymbol{V} \in \mathcal{K}, \qquad \forall j \in N \\ & \|\boldsymbol{t}\|_2 \leq y \\ & \text{for some } y \in \Re,\ \boldsymbol{t} \in \Re^N, \end{aligned} \tag{13}$$

for a given reference vector, $\boldsymbol{V}$ in the interior of the cone, $\mathcal{K}$. The formulation (12) becomes a cartesian product of $2N + 1$ cones of the nominal problem plus an additional second-order cone, which is a computationally tractable cone. Hence, in theory the formulation (12) is not much harder to solve compared with its nominal problem.

One natural question is whether the simple approximation is overly conservative with respect to Problem (8). While there is lack of theoretical evidence on the closeness of the approximation, the framework does lead to an approximation of the conic chance constraint problem. An important component of the analysis is the relation among different norms, which we will subsequently present.

Recall that a norm satisfies $\|\boldsymbol{A}\| \geq 0$, $\|k\boldsymbol{A}\| = |k| \cdot \|\boldsymbol{A}\|$, $\|\boldsymbol{A}+\boldsymbol{B}\| \leq \|\boldsymbol{A}\|+\|\boldsymbol{B}\|$, and $\|\boldsymbol{A}\| = 0$, implies that $\boldsymbol{A} = \boldsymbol{0}$. For a given regular cone, $\mathcal{K}$, and its interior, $\boldsymbol{V}$, we define the following cone induced norm

$$\|\boldsymbol{Y}\|_{\mathcal{K},\boldsymbol{V}} \triangleq \min\{y,\ y\boldsymbol{V}-\boldsymbol{Y} \in \mathcal{K},\ \boldsymbol{Y}+y\boldsymbol{V} \in \mathcal{K}\}\,. \quad (14)$$

Proposition 3

$$\max\{-f(\boldsymbol{X},\ (z_0, \boldsymbol{z})),\ -f(\boldsymbol{X},\ -(z_0,\ \boldsymbol{z}))\} = \|\boldsymbol{Y}((z_0,\ \boldsymbol{z}))\|_{\mathcal{K},\boldsymbol{V}}\,.$$

We consider the common cones and the respective norms.

(a) Second-order cone:
Let $\boldsymbol{e}_1 \in \text{int}(\mathcal{L}^{n+1})$ be the reference vector, we have for any vector $(y_0,\ \boldsymbol{y}) \in \Re^{n+1}$

$$\begin{aligned}&\|(y_0,\ \boldsymbol{y})\|_{\mathcal{L}^{n+1},\boldsymbol{e}_{n+1}}\\ &= \min\{\theta\colon \|\boldsymbol{y}\|_2 \leq \theta - y_0, \|\boldsymbol{y}\|_2 \leq \theta + y_0\}\\ &= \|\boldsymbol{y}\|_2 + |y_0|\end{aligned}$$

(b) Cone of symmetric positive definite matrix:
Let the identity matrix $\boldsymbol{I}$ be the reference matrix, then for any $m \times m$ symmetric matrix, $\boldsymbol{Y}$,

$$\begin{aligned}\|\boldsymbol{Y}\|_{S_+^m,\boldsymbol{I}} &= \min\{y,\ y\boldsymbol{I}-\boldsymbol{Y} \in S_+^m,\ \boldsymbol{Y}-y\boldsymbol{I} \in S_+^m\}\\ &= \|\boldsymbol{Y}\|_2\,.\end{aligned}$$

Proposition 4 *Suppose $\boldsymbol{X}$ is feasible in Problem (12) then*

$$\begin{aligned}&P(f(\boldsymbol{X},\ (1,\ \tilde{\boldsymbol{z}})) < 0)\\ &\leq\quad P\left(\left\|\sum_{j=1}^N \boldsymbol{Y}_j\tilde{z}_j\right\|_{\mathcal{K},\boldsymbol{V}} > \rho\sqrt{\sum_{j\in N}\|\boldsymbol{Y}_j\|^2_{\mathcal{K},\boldsymbol{V}}}\right)\,.\end{aligned}$$

To obtain explicit bounds, we focus on primitive uncertainties, $\tilde{\boldsymbol{z}}$ that are normally and independently distributed with mean zero and variance one. For a sum of random scalers, we have

$$\mathrm{P}\left(\left|\sum_{j=1}^N y_j\tilde{z}_j\right| > \rho\sqrt{\sum_{j=1}^N y_j^2}\right) \leq 1 - 2\Phi(\rho)\,.$$

To derive a similar large deviation result for the sum of random vectors used in Proposition 4, we consider the following generalization:

$$\mathrm{P}\left(\left\|\sum_{j=1}^N \boldsymbol{Y}_j\tilde{z}_j\right\|_{\mathcal{K},\boldsymbol{V}} > \rho\sqrt{\sum_{j=1}^N\|\boldsymbol{Y}_j\|^2_{\mathcal{K},\boldsymbol{V}}}\right) \leq \phi(\rho)\,,$$

where $\phi(\rho)$ is a non-trivial probability bound that depends on the choice of cone, $\mathcal{K}$, and possibly the dimension and the reference vector, $\boldsymbol{V}$.

An important component of the analysis is the relation among different norms. We denote by $\langle\ ,\ \rangle$ the inner product on a vector space, $\Re^m$, or the space of m by m symmetric matrices. The inner product induces a norm $\|\boldsymbol{X}\| \triangleq \sqrt{\langle \boldsymbol{X}, \boldsymbol{X}\rangle}$. For a vector space, the natural inner product is the Euclidian inner product, $\langle \boldsymbol{x},\ \boldsymbol{y}\rangle = \boldsymbol{x}'\boldsymbol{y}$, and the induced norm is the Euclidian norm $\|\boldsymbol{x}\|_2$. For the space of symmetric matrices, the natural inner product is the trace product or $\langle \boldsymbol{X},\ \boldsymbol{Y}\rangle = \text{trace}(\boldsymbol{X}\boldsymbol{Y})$ and the corresponding induced norm is the Frobenius norm, $\|\boldsymbol{Y}\|_F$.

We analyze the relation of the inner product norm $\sqrt{\langle \boldsymbol{X}, \boldsymbol{X}\rangle}$ with the norm $\|\boldsymbol{X}\|_{\mathcal{K},\boldsymbol{V}}$ for the conic optimization problems we consider. Since $\|\boldsymbol{X}\|_{\mathcal{K},\boldsymbol{V}}$ and the inner product norm $\|\boldsymbol{X}\|$ are valid norms in a finite dimensional space, there exist finite $\alpha_1,\ \alpha_2 > 0$ such that

$$\frac{1}{\alpha_1}\|\boldsymbol{X}\|_{\mathcal{K},\boldsymbol{V}} \leq \|\boldsymbol{X}\| \leq \alpha_2\|\boldsymbol{X}\|_{\mathcal{K},\boldsymbol{V}}\,,$$

for all $\boldsymbol{X}$ in the relevant space. Hence, we define the parameter

$$\alpha_{\mathcal{K},\boldsymbol{V}} = \underbrace{\left(\max_{\|\boldsymbol{X}\|=1}\|\boldsymbol{X}\|_{\mathcal{K},\boldsymbol{V}}\right)}_{=\alpha_1}\underbrace{\left(\max_{\|\boldsymbol{X}\|_{\mathcal{K},\boldsymbol{V}}=1}\|\boldsymbol{X}\|\right)}_{=\alpha_2} \quad (15)$$

which measures the disparity between the norm $\|\cdot\|_{\mathcal{K},\boldsymbol{V}}$ and the inner product norm $\|\cdot\|$.

Parameter $\alpha_{\mathcal{K},\boldsymbol{V}}$ of Common Cones

(a) Second-order cone:
Let $\boldsymbol{e}_{n+1}$ be the reference vector, then

$$\|(\boldsymbol{y}, y_{n+1})\|_{\mathcal{L}^{n+1},\boldsymbol{y}} = \|\boldsymbol{y}\|_2 + |y_{n+1}|\,.$$

Therefore,

$$\begin{aligned}\frac{1}{\sqrt{2}}\|(\boldsymbol{y},\ y_{n+1})\|_{\mathcal{L}^{n+1},\boldsymbol{e}_{n+1}} &\leq \|(\boldsymbol{y},\ y_{n+1})\|_2\\ &\leq \|(\boldsymbol{y},\ y_{n+1})\|_{\mathcal{L}^{n+1},\boldsymbol{e}_{n+1}}\end{aligned}$$

Approximations to Robust Conic Optimization Problems, Table 1
Probability bounds of $P(f(X,(1,\tilde{z}))\prec 0)$ for $\tilde{z}\sim\mathcal{N}(0,I)$.

Type	Probability bound of infeasibility
$\mathcal{L}^{m+1}$	$\sqrt{\frac{e}{2}}\Omega\exp(-\frac{\Omega^2}{4})$
S_+^m	$\sqrt{\frac{e}{m}}\Omega\exp(-\frac{\Omega^2}{2m})$

and hence,

$$\alpha_{\mathcal{L}^{n+1},V}=\sqrt{2}\,.$$

(b) Cone of symmetric positive definite matrix:
Let $\boldsymbol{I}$ be the reference matrix, then for any $m\times m$ symmetric matrix $\boldsymbol{Y}$

$$\|\boldsymbol{Y}\|_{S_+^m,\boldsymbol{I}}=\|\boldsymbol{Y}\|_2\,.$$

Let $\lambda_j,\ j=1,\ldots,m$ be the eigenvalues of the matrix $\boldsymbol{Y}$. Since $\|\boldsymbol{Y}\|_F=\sqrt{\mathrm{trace}(\boldsymbol{Y}^2)}=\sqrt{\sum_j\lambda_j^2}$ and $\|\boldsymbol{Y}\|_2=\max_j|\lambda_j|$, we have

$$\|\boldsymbol{Y}\|_2\le\|\boldsymbol{A}\|_F\le\sqrt{m}\|\boldsymbol{Y}\|_2\,.$$

Hence,

$$\alpha_{S_+^m,\boldsymbol{I}}=\sqrt{m}\,.$$

Theorem 2 *Given an inner product norm $\|\cdot\|$ and under the assumption that $\tilde{z}_j$ are normally and independently distributed with mean zero and variance one, i. e., $\tilde{\boldsymbol{z}}\sim\mathcal{N}(0,\boldsymbol{I})$, then*

$$P\left(\left\|\sum_{j=1}^N\boldsymbol{Y}_j\tilde{z}_j\right\|_{\mathcal{K},V}>\rho\sqrt{\sum_{j\in N}\|\boldsymbol{Y}_j\|_{\mathcal{K},V}^2}\,,\right)\le\frac{\sqrt{e}\rho}{\alpha_{\mathcal{K},V}}\exp\left(-\frac{\rho^2}{2\alpha_{\mathcal{K},V}^2}\right),\quad(16)$$

for all $\rho>\alpha_{\mathcal{K},V}$.

In order to have the smallest budget of uncertainty, ρ, it is reasonable to select $\boldsymbol{V}$ that minimizes $\alpha_{\mathcal{K},V}$, i. e.,

$$\alpha_{\mathcal{K}}=\min_{V\in\mathrm{int}(\mathcal{K})}\alpha_{\mathcal{K},V}\,.$$

For general conic optimization, we have shown that the probability bound depends on the the choice of $\boldsymbol{V}\in\mathrm{int}(\mathcal{K})$. A cone, $\mathcal{K}\subseteq\Re^n$ is *homogenous* if for any pair of points $\boldsymbol{A},\boldsymbol{B}\in\mathrm{int}(\mathcal{K})$ there exists an invertible linear map $\boldsymbol{M}:\Re^n\to\Re^n$ such that $\boldsymbol{M}(\boldsymbol{A})=\boldsymbol{B}$ and $\boldsymbol{M}(\mathcal{K})=\mathcal{K}$. It turns out that for *homogenous cones*, of which semidefinite and second-order cones are special cases, the probability bound does not depend on $\boldsymbol{V}\in\mathrm{int}(\mathcal{K})$.

Theorem 3 *Suppose the cone $\mathcal{K}$ is homogenous. For any $\boldsymbol{V}\in int(\mathcal{K})$, the probability bound of conic infeasibility satisfies*

$$P(\boldsymbol{Y}((1,\boldsymbol{z}))\notin\mathcal{K})\le\frac{\sqrt{e}\rho}{\alpha_{\mathcal{K}}}\exp\left(-\frac{\rho^2}{2\alpha_{\mathcal{K}}^2}\right).$$

For the second-order cone, $\alpha_{\mathcal{L}^{n+1}}=\sqrt{2}$ and for the symmetric positive semidefinite cone, $\alpha_{S_+^m}=\sqrt{m}$.

While different $\boldsymbol{V}$ lead to the same probability bounds, some choices of $\boldsymbol{V}$ may lead to better objectives. The following theorem suggests an iterative improvement strategy.

Theorem 4 *For any $\boldsymbol{V}\in int(\mathcal{K})$, if $\boldsymbol{X}$, $\boldsymbol{t}$ and $y>0$ are feasible in (13), then they are also feasible in the same problem in which $\boldsymbol{V}$ is replaced by*

$$\boldsymbol{W}=\boldsymbol{Y}_0/(\rho y)\,.$$

While we focus on the primitive uncertainty vector $\tilde{\boldsymbol{z}}$ being normally distributed, using the large deviation bounds of Nemirovski [12], we can also apply the same framework to other distributions. The interested reader may refer to Bertsimas and Sim [5].

References

1. Ben-Tal A, Nemirovski A (1998) Robust convex optimization. Math Oper Res 23:769–805
2. Ben-Tal A, Nemirovski A (2000) Robust solutions of Linear Programming problems contaminated with uncertain data. Math Program 88:411–424
3. Ben-Tal A, Nemirovski A (2001) Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications. MPR-SIAM Series on Optimization. SIAM, Philadelphia
4. Bertsimas D, Sim M (2004) Price of robustness. Oper Res 52:35–53
5. Bertsimas D, Sim M (2006) Tractable Approximations to Robust Conic Optimization Problems. Math Program 107(1):5–36

6. Chen W, Sim M (2007) Goal Driven Optimization. Working Paper, NUS Business School
7. Chen W, Sim M, Sun J, Teo C-P (2007) From CVaR to Uncertainty Set: Implications in Joint Chance Constrained Optimization. Working Paper, NUS Business School
8. Chen X, Sim M, Sun P (2006) A robust optimization perspective on stochastic programming. to appear in: Oper Res 55(6)
9. El Ghaoui L, Oustry F, Lebret H (1998) Robust Solutions to Uncertain Semidefinite Programs. SIAM J Optim 9(1):33–52
10. Lin X, Janak SL, Floudas CA (2004) A New Robust Optimization Approach for Scheduling under Uncertainty: I Bounded Uncertainty. Comput Chem Eng 28:1069–1085
11. Janak SL, Lin X, Floudas CA (2007) A New Robust Optimization Approach for Scheduling under Uncertainty: II Uncertainty with Known Probability Distribution. Comput Chem Eng 31:171–195
12. Nemirovski A (2003) On tractable approximations of ramdomly perturbed convex constraints. Proceedings of the 42nd IEEE Conference on Decision and Control, Maui, Hawaii, USA, December, pp 2419–2422
13. Pardalos PM, Wolkowicz H (1998) Topics in Semidefinite and Interior-Point Methods. Fields Institute Communications Series vol 18, American Mathematical Society

Archimedes and the Foundations of Industrial Engineering

PETROS XANTHOPOULOS
Industrial and Systems Engineering, University of Florida, Gainesville, USA

MSC2000: 01A20

Article Outline

Biographical Sketch

Archimedes (287–212 B.C.) was a famous Greek mathematician, engineer and philosopher. Born in the city of Syracuse on the Island of Sicily to an astronomer and mathematician named Phidias, Archimedes spent the first years of his life in his home city and went to Alexandria in Egypt to study mathematics. He soon became friends with Konon of Samos and Eratosthenes. After spending a considerable amount of time in Alexandria, he returned to Syracuse, where he remained for the rest of his life conducting mathematical research. He had a good relationships with king Hieron of Syracuse and his son Gelon. We know that he assisted king Hieron numerous times either with his inventions during the Second Punic War or by solving problems like the well-known case (the one that Archimedes jumped out of his bathtub crying out eureka) with the crown of king Hieron during peacetime.

In this article we will concentrate on the work of Archimedes, which is closely related to what we call today industrial engineering (including the mathematical theory of optimization, operations research, theory of algorithms, etc.). In particular, we will present Archimedes' definition of convex sets, his method of exhaustion for computing finite integrals, his contribution to recursive algorithms, and his approach to solving real-life operations research problems during the Second Punic War.

Archimedes' Work

One very important concept for optimization is the definition of convex sets. The first such definition was given by Euclid in his books *Elements*, but Archimedes elaborated this definition and gave us his definition, which was used until the first decades of the 20th century. In his work *On the sphere and the cylinder* he gives the following definition of the convex arc:

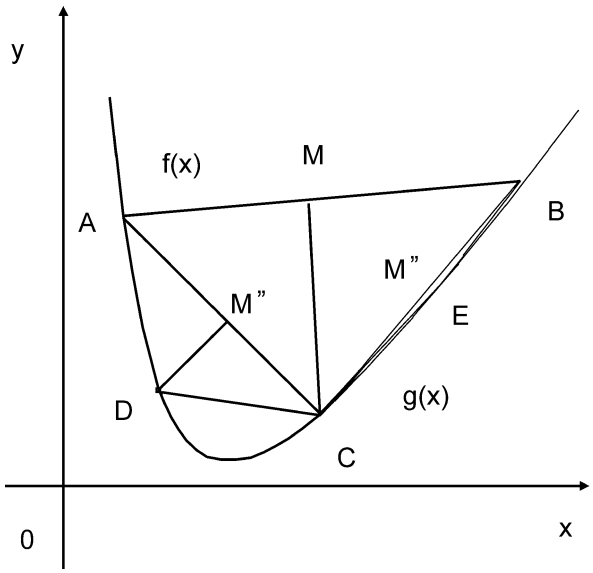
Definition 1 I call convex in one and the same directions the surfaces for which the straight line joining two arbitrary points lies on the same side of the surface.

On his work *On the equilibrium of planes* he gives a definition of the convex set using the center-of-gravity concept:

Definition 2 In any figure whose perimeter is convex the center of gravity must be within the figure.

It is worth mentioning that Archimedes' definitions of convex arcs and convex sets were those used until 1913, when E. Steinitz introduced the modern definitions of convexity.

Archimedes had invented a geometrical method called the method of exhaustion (or method of infinitesimals) in order to be able to compute areas under convex curves. This was one of the first geometrical methods devised to compute what we call today definite



Archimedes and the Foundations of Industrial Engineering, Figure 1
Illustration of Archimedes' exhaustion method

integrals. In modern notation Archimedes was able to compute

$$\int_a^b [f(x) - g(x)]dx, \tag{1}$$

where $f(x)$ is a line segment and $g(x)$ a convex function (usually parabola). An illustration of this method can be found in Fig. 1.

Suppose that we want to compute the area over a curve and below the line segment AB. Archimedes considered the triangle $\widehat{ABC}$, where C is the point below the midpoint M of the line segment AB(MC is the middle vertical of AB). If we iteratively repeat this process, we can see that the next two parabolic triangles have an area that is $\frac{1}{4}$ of the initial triangle. Therefore, the area of the curve was the infinite sum of $1 + \frac{1}{4} + \frac{1}{8} + \ldots$, where 1 corresponds to the area of the initial triangle $\widehat{ABC}$. In this way Archimedes was able to geometrically approximate the area of a convex parabolic curve.

According to [7] Archimedes was the first (in around 220 B.C.) to use a double recursive algorithm to solve the problem of the sand reckoner (Psammitis). In this book he tries to come up with of a number that is much larger than the number of grains of sand in the world and therefore prove that the number of grains of sand in the world is not infinite. For this he fixes a number α and defines the number $p_k(x)$ as follows (using a double recursion scheme):

$$\begin{aligned} p_0(x) &= 1, \\ p_{k+1}(0) &= p_k(\alpha), \\ p_{n+1}(x+1) &= a p_{k+1}(x)\ . \end{aligned} \tag{2}$$

Therefore, $p_k(x) = \alpha^{xk}$. Then he considers $p_\alpha(\alpha)$ for $\alpha = 10^8$, which was the largest number known at that time, and he comes up with the number $10^{10}17$, which was the largest number used in mathematics until 1933.

Apart from Archimedes' exceptional skills in theoretical research, he also became famous for his ability to deal with everyday life problems. Although operations research was developed during World War II, when mathematicians were looking for ways to make better decisions in utilizing certain materials subject to some constraint, some consider Archimedes the father of operations research as he helped his home city defend itself against the Romans during the Second Punic War.

Before King Hieron died, he asked Archimedes to organize the complete defense of Syracuse against Roman general Marcelus. Archimedes is said to have invented many mechanical war machines like the claw of Archimedes, a new version of catapult, an array of mirrors that was able to burn enemy ships, etc.

Archimedes was also responsible for organizing the defense of Syracuse and the redecoration of Fort Euryalus [6]. Due to Archimedes' clever defense plans, Syracuse managed to survive the Roman siege for 2 years.

Conclusion

Archimedes was a perfect example of a scientist who managed to combine theoretical research with practical problem solving. He managed to distinguish between the two by referring to his mechanical inventions as *parergon*. This shows that Archimedes was capable of performing both basic and applied research, but he regarded basic research as more important. In this sense he can be considered the father of the modern industrial engineer who utilizes theoretical methods to solve problems that arise in everyday life.

References

1. Brunschwig J, Lloyd GER (eds) (2000) Greek thought: a guide to classical knowledge. Belknap, Cambridge, Massachusetts
2. Dijksterhuis EJ (1987) Archimedes. Princeton University Press, Princeton, NJ
3. Gow M (2005) Archimedes: Mathematical Genius of the Ancient World. Great Minds of Science series. Enslow, Berkeley Heights, NJ
4. Heath TL (1897) The works of Archimedes. Cambridge University Press, Cambridge
5. Heath TL (ed) (2002) The Works of Archimedes. Dover, New York
6. Lawrence AW (1946) Archimedes and the Design of Euryalus Fort. J Hell Stud – JSTOR
7. Odifreddi P. Recursive Functions. In: Edward Zalta N (ed) The Stanford Encyclopedia of Philosophy (Summer 2005 edn). http://plato.stanford.edu/archives/sum2005/entries/recursive-functions. Accessed 21 Mar 2008
8. Simms DL (1995) Archimedes the Engineer. History of Technology, vol 17. Continuum International, London, pp 45–111
9. Stein S (1999) Archimedes: What Did He Do Besides Cry Eureka? Mathematical Association of America, Washington, DC

Asset Liability Management Decision Support System

KOSMIDOU KYRIAKI[1], ZOPOUNIDIS CONSTANTIN[2]

[1] Department of International European Economic Studies, Athens University Economics and Business, Athens, Greek

[2] Department of Production Engineering and Management, Technical University of Crete, Chania, Greece

MSC2000: 90C29, 65K99

Article Outline

Introduction

Asset Liability Management (ALM) is an important dimension of risk management, where the exposure to various risks is minimized while maintaining the appropriate combination of asset and liability, in order to satisfy the goals of the firm or the financial institution (Kosmidou and Zopounidis [18]).

Up to the 1960's, liability management was aimless. In their majority, the banking institutions considered liabilities as exogenous factors contributing to the limitation of asset management. Indeed, for a long period the greater part of capital resources originated from savings deposits and deposits with agreed maturity.

Nevertheless, the financial system has radically changed. Competition among the banks for obtaining capital has become intense. Liability management is the main component of each bank strategy in order to ensure the cheapest possible financing. At the same time, the importance of decisions regarding the amount of capital adequacy is enforced. Indeed, the adequacy of the bank as far as equity, contributes to the elimination of bankruptcy risk, a situation in which the bank cannot satisfy its debts towards clients who make deposits or others who take out loans. Moreover, the capital adequacy of banks is influenced by the changes of stock prices in relation to the amount of the capital stock portfolio. Finally, the existence of a minimum amount of equity is an obligation of commercial banks to the Central Bank for supervisory reasons. It is worth mentioning that based on the last published data (31/12/2001) the Bank of Greece assigns the coefficient for the Tier 1 capital at 8%, while the corresponding European average is equal to 6%. This results in the configuration of the capital adequacy of the Greek banking system at higher levels than the European average rate. The high capital adequacy index denotes large margins of profitability amelioration, which reduces the risk of a systematic crisis.

Asset management in a contemporary bank cannot be distinct from liability management. The simultaneous management of assets and liabilities, in order to maximize the profits and minimize the risk, demands the analysis of a series of issues.

Firstly, there is the substantive issue of strategic planning and expansion. That is, the evaluation of the total size of deposits that the bank wishes to attract and the total number of loans that it wishes to provide.

Secondly, there is the issue of determination of the "best temporal structure" of the asset liability management, in order to maximize the profits and to ensure

the robustness of the bank. Deposits cannot all be liquidated in the same way. From the point of view of assets, the loans and various placements to securities constitute commitments of the bank's funds with a different duration time. The coordination of the temporal structure of the asset liability management is of major importance in order to avoid the problems of temporary liquidity reduction, which might be very injurious.

Thirdly, there is the issue of risk management of assets and liabilities. The main focus is placed on the assets, where the evaluation of the quality of the loans portfolio (credit risk) and the securities portfolio (market risk) is more easily measurable.

Fourthly, there is the issue of configuration of an integrated invoice, which refers to the entire range of bank operations. It refers mainly to the determination of interest rates for the total of loans and deposits as well as for the various commissions which the bank charges for specific mediating operations. It is obvious that in a bank market which operates in a competitive environment, there is no issue of pricing. This is true even in the case where all interest rates and commissions are set by monetary authorities, as was the situation in Greece before the liberalization of the banking system.

In reality, bank markets have the basic characteristics of monopolistic competition. Thus, the issue of planning a system of discrete pricing and product diversification is of major importance. The problem of discrete pricing, as far as the assets are concerned, is connected to the issue of risk management. It is a common fact that the banks determine the borrowing interest rate on the basis of the interest rates which increase in proportion to the risk as they assess it in each case. The product diversification policy includes all the loan and deposit products and is based on thorough research which ensures the best possible knowledge of market conditions.

Lastly, the management of operating cost and technology constitutes an important issue. The collaboration of a well-selected and fully skilled personnel, as well as contemporary computerization systems and other technological applications, constitutes an important element in creating a low-cost bank. This results in the acquisition of a significant competitive advantage against other banks, which could finally be expressed through a more aggressive policy of attracting loans and deposits with low loan interest rates and high deposit interest rates. The result of this policy is the increase of the market stake. However, the ability of a bank to absorb the input of the best strategic technological innovations depends on the human resources management.

The present research focuses on the study of bank asset liability management. Many are the reasons that lead us to study bank asset liability management, as an application of ALM. Firstly, bank asset/liability management has always been of concern to bank managers, but in the last years and especially today its importance has grown more and more. The development of information technology has led to such an increasing public awareness that the bank's performance, its politics and its management are closely monitored by the press and the bank's competitors, shareholders and customers and thereby highly affect the bank's public standing.

The increasing competition in the national and international banking markets, the changeover towards the monetary union and the new technological innovations herald major changes in the banking environment and challenge all banks to make timely preparations in order to enter into the new competitive monetary and financial environment.

All the above drove banks to seek out greater efficiency in the management of their assets and liabilities. Thus, the central problem of ALM revolves around the bank's balance sheet and the main question that arises is: What should be the composition of a bank's assets and liabilities on average given the corresponding returns and costs, in order to achieve certain goals, such as maximization of the bank's gross revenues?

It is well known that finding an appropriate balance between profitability, risk and liquidity considerations is one of the main problems in ALM. The optimal balance between these factors cannot be found without considering important interactions that exist between the structure of a bank's liabilities and capital and the composition of its assets.

Bank asset/liability management is defined as the simultaneous planning of all asset and liability positions on the bank's balance sheet under consideration of the different banking and bank management objectives and legal, managerial and market constraints. Banks are looking to maximize profit and minimize risk.

Taking into consideration all the above, the purpose of this paper is to develop a goal programming system

into a stochastic environment, focusing, mainly, on the change of the interest rate risk. This system provides the possibility to the administrative board and the managers of the bank to proceed to various scenarios related to their future economic process, aiming mainly to the management of the risks, emerged from the changes of the market parameters.

The rest of the paper is organized as follows. The next section includes a brief overview of bank ALM techniques. Section "Model" outlines the methodology used and describes the development of the ALM decision support system. Finally, the conclusions of the paper as well as future research perspectives are discussed in the last section.

Background

Looking to the past, we find the first mathematical models in the field of bank management. Asset and liability management models can be deterministic or stochastic (Kosmidou and Zopounidis [17]).

Deterministic models use linear programming, assume particular realizations for random events, and are computationally tractable for large problems. The deterministic linear programming model of Chambers and Charnes [6] is the pioneer in ALM. Chambers and Charnes were concerned with formulating, exploring and interpreting the use and construction which may be derived from a mathematical programming model which expresses more realistically than past efforts the actual conditions of current operations. Their model corresponds to the problem of determining an optimal portfolio for an individual bank over several time periods in accordance with requirements laid down by bank examiners which are interpreted as defining limits within which the level of risk associated with the return on the portfolio is an acceptable one.

Cohen and Hammer [9], Robertson [31], Lifson and Blackman [23], Fielitz and Loeffler [14] have realized successful applications of Chambers and Charnes' model. Even though these models have differed in their treatment of disaggregation, uncertainty and dynamic considerations, they all have in common the fact that they are specified to optimize a single objective profit function subject to the relevant linear constraints.

Eatman and Sealey [12] developed a multiobjective linear programming model for commercial bank balance sheet management considering profitability and solvency objectives subject to policy and managerial constraints.

Giokas and Vassiloglou [15] developed a goal-programming model for bank asset and liability management. They supported the idea that apart from attempting to maximize revenues, management tries to minimize risks involved in the allocation of the bank's capital, as well as to fulfill other goals of the bank, such as retaining its market share, increasing the size of its deposits and loans, etc. Conventional linear programming is unable to deal with this kind of problem, as it can only handle a single goal in the objective function. Goal programming is the most widely used approach that solves large-scale multi-criteria decision making problems.

Apart from the deterministic models, several stochastic models have been proposed since the 1970s. These models, including the use of chance-constrained programming [7,8,29], dynamic programming [13,25,26,32], sequential decision theory [3,35] and stochastic linear programming under uncertainty [2,10,11,16], presented computational difficulties. The stochastic models, in their majority, originate from the portfolio selection theory of Markowitz [24] and they are known as static mean-variance methods. Pyle [30] and Brodt [4] adapted Markowitz's theory and presented an efficient dynamic balance sheet management plan that considers only the risk of the portfolio and not other possible uncertainties or maximizes profits for a given amount of risk over a multi-period planning horizon respectively.

Wolf [35] proposed the sequential decision theoretic approach that employs sequential decision analysis to find an optimal solution through the use of implicit enumeration.

An alternative approach in considering stochastic models, is the stochastic linear programming with simple recourse. Kusy and Ziemba [19] employed a multiperiod stochastic linear program with simple recourse to model the management of assets and liabilities in banking while maintaining computational feasibility. Their results indicate that the proposed ALM model is theoretically and operationally superior to a corresponding deterministic linear programming model and that the computational effort required for its implementation is comparable to that of the deterministic

model. Another application of the multistage stochastic programming is the Russell-Yasuda Kasai model [5], which aims at maximizing the long term wealth of the firm while producing high income returns.

Mulvey and Vladimirou [27] used dynamic generalized network programs for financial planning problems under uncertainty and they developed a model in the framework of multi-scenario generalized network that captures essential features of various discrete time financial decision problems.

Finally, Mulvey and Ziemba [28] present a more detailed overview of various asset and liability modeling techniques, including models for individuals and financial institutions such as banks and insurance companies.

Moreover, over the years, many models have been developed in the area of financial analysis and financial planning techniques. Kvanli [20], Lee and Lerro [22], Lee and Chesser [21], Baston [1], Sharma et al. [34], among others have applied goal programming to investment planning. Giokas and Vassiloglou [15], Seshadri et al. [33] presented bank models using goal programming. These studies focus on the areas of banking and financial institutions and they use data from the bank financial statements.

Model

Kosmidou and Zopounidis [18] developed an asset liability management (ALM) methodology into a stochastic environment of interest rates in order to select the best direction strategies to the banking financial planning. The ALM model was developed through goal programming in terms of a one-year time horizon. The model used balance sheet and income statement information for the previous year of the year t to produce a future course of ALM strategy for the year $t+1$. As far as model variables are concerned, we used variables familiar to management and facilitated the specification of the constraints and goals. For example, goals concerning measurements such as liquidity, return and risk have to be expressed in terms of utilized variables.

More precisely, the asset liability management model that was developed can be expressed as follows:

$$\min z = \sum_{P} p_k (d_k^- + d_k^+) \tag{1}$$

subject to constraints:

$$\mathrm{K}\Phi_{X'} \le X' \le \mathrm{A}\Phi_{X'} \tag{2}$$

$$\mathrm{K}\Phi_{Y'} \le Y' \le \mathrm{A}\Phi_{Y'} \tag{3}$$

$$\sum_{i=1}^{n} X_i = \sum_{j=1}^{m} Y_j \quad \forall i = 1, \ldots, n, \quad \forall j = 1, \ldots, m \tag{4}$$

$$\sum_{j \in \Pi_{Y''}} Y_j - a \sum_{i \in \mathrm{E}_{X''}} X_i = 0 \tag{5}$$

$$\sum_{j \in \Pi_1} Y_j - \sum_{i \in \mathrm{E}} w_i X_i - d_s^+ + d_s^- = k_1 \tag{6}$$

$$\sum_{i \in \mathrm{E}_x} X_i - k_2 \sum_{j \in \Pi_k} Y_j + d_l^- - d_l^+ = 0 \tag{7}$$

$$\sum_{i=1}^{n} R_i^X X_i - \sum_{j=1}^{m} R_j^Y Y_j - d_r^+ + d_r^- = k_3 \tag{8}$$

$$\sum_{i \in \mathrm{E}_p} X_i + d_p^- - d_p^+ = l_p\,, \quad \forall p \tag{9}$$

$$\sum_{j \in \Pi_p} Y_j + d_p^- - d_p^+ = l_p\,, \quad \forall p \tag{10}$$

$$X_i \ge 0, Y_j \ge 0, d_k^+ \ge 0, d_k^- \ge 0\,, \quad \text{for all } i = 1, \ldots, n, j = 1, \ldots, m, k \in P \tag{11}$$

where

- $\boldsymbol{X_i}$: the element i of asset, $\forall i = 1, \ldots, n$, n is the number of asset variables
- $\boldsymbol{Y_j}$: the element j of liability, $\forall j = 1, \ldots, m$, m is the number of liability variables
- $\mathrm{K}\Phi_{X'}$ ($\mathrm{K}\Phi_{Y'}$): is the low bound of specific asset accounts X′ (liability Y′)
- $\mathrm{A}\Phi_{X'}$ ($\mathrm{A}\Phi_{Y'}$): is the upper bound of specific asset accounts X′ (liability Y′)
- $\mathrm{E}_{X''}$: specific categories of asset accounts
- $\Pi_{Y''}$: specific categories of liability accounts
- α: the desirable value of specific asset and liability data
- Π_1: the liability set, which includes the equity
- **E:** the set of assets
- $\boldsymbol{w_i}$: the degree of riskness of the asset data
- $\boldsymbol{k_1}$: the solvency ratio, as it is defined from the European Central Bank.

k_2: the liquidity ratio, as it is defined from the bank policy
E_χ: the set of asset data, which includes the loans
Π_κ: the set of liability data, which includes the deposits
R_i^X: the expected return of the asset i, $\forall i = 1, \ldots, n$
R_j^Y: the expected return of the liability j, $\forall j = 1, \ldots, m$
k_3: the expected value for the goal of asset and liability return
P: the goal imposed from the bank
L_p: the desirable value goal for the goal constraint p defined by the bank
d_k^+: the over-achievement of the goal k, $\forall k \in P$
d_k^-: the under-achievement of the goal k, $\forall k \in P$
p_k: the priority degree (weight) of the goal k

Certain constraints are imposed by the banking regulation on particular categories of accounts. Specific categories of asset accounts (X') and liability accounts (Y') are detected and the minimum and maximum allowed limit for these categories are defined based on the strategy and policy that the bank intends to follow (constraints 2–3).

The structural constraints (4–5) include those that contribute to the structure of the balance sheet and especially to the performance of the equation Assets = Liabilities + Net Capital.

The bank management should determine specific goals, such as the desirable structure of each financial institution's assets and liabilities for the units of surplus and deficit, balancing the low cost and the high return. The structure of assets and liabilities is significant, since it affects swiftly the income and profits of the bank.

Referring to the goals of the model, the solvency goal (6) is used as a risk measure and is defined as the ratio of the bank's equity capital to its total weighted assets. The weighting of the assets reflects their respective risk, greater weights corresponding to a higher degree of risk. This hierarchy takes place according to the determination of several degrees of significance for the variables of assets and liabilities. That is, the variables with the largest degrees of significance correspond to categories of the balance sheet accounts with the highest risk stages.

Moreover, a basic policy of the commercial banks is the management of their liquidity and specifically the measurement of their needs that is relative to the progress of deposits and loans. The liquidity goal (7) is defined as the ratio of liquid assets to current liabilities and indicates the liquidity risk, that indicates the possibility of the bank to respond to its current liabilities with a security margin, which allows the probable reduction of the value of some current data.

Furthermore, the bank aims at the maximization of its efficiency that is the accomplishment of the largest possible profit from the best placement of its funds. Its aim is the maximization of its profitability and therefore precise and consistent decisions should be taken into account during the bank management. These decisions will guarantee the combined effect of all the variables that are included on the calculation of the profits. This decision taking gives emphasis to several selected variables that are related to the bank management, such as to the management of the difference between the asset return and the liability cost, the expenses, the liquidity management and the capital management. The goal (8) determines the total expected return based on the expected returns for all the assets R^X and liabilities R^Y.

Beside the goals of solvency, liquidity and return of assets and liabilities, the bank could determine other goals that concern specific categories of assets and liabilities, in proportion to the demands and preferences of the bank managers. These goals are the deposit goal, the loan goal and the goal of asset and liability return.

The drawing of capital, especially from the deposits constitutes a major part of commercial bank management. All sorts of deposits constitute the major source of capital for the commercial banks, in order to proceed to the financing of the economy, through the financing of firms. Thus, it is given special significance to the deposits goal.

The goal of asset and liability return defines the goal for the overall expected return of the selected asset-liability strategy over the year of the analysis.

Finally, there are goals reflecting that variables such as cash, cheques receivables, deposits to the Bank of Greece and fixed assets, should remain at the levels of previous years. More analytically, it is known that the fixed assets are the permanent assets, which have a natural existence, such as buildings, machines, locations and equipment, etc. Intangible assets are the fixed assets, which have no natural existence but constitute rights and benefits. They have significant economic value, which sometimes is larger than the value of the

tangible fixed assets. These data have stable character and are used productively by the bank for the regular operation and performance of its objectives. Since the fixed assets, tangible or intangible, are presented at the balance sheet at their book value that is the initial value of cost minus the depreciation till today, it is assumed that their value does not change during the development of the present methodology.

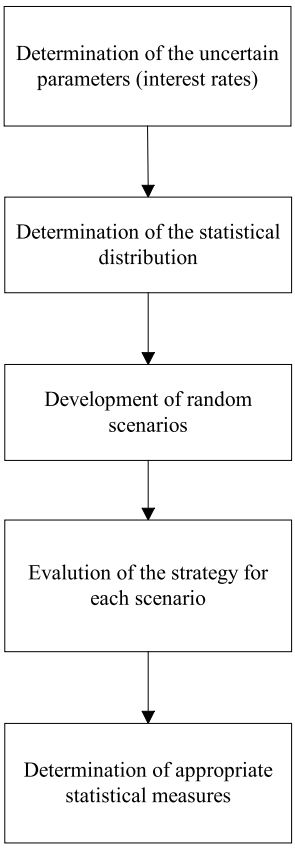
At this point, Kosmidou and Zopounidis [18] took into account that the banks should manage the interest rate risk, the operating risk, the credit risk, the market risk, the foreign exchange risk, the liquidity risk and the country risk.

More specifically, the interest rate risk indicates the effect of the changes to the net profit margin between the deposit and borrowing values, which are evolved as a consequence of the deviations to the dominant interest rates of assets and liabilities. When the interest rates diminish, the banks accomplish high profits since they can refresh their liabilities to lower borrowing values. The reverse stands to high borrowing values. It is obvious, that the changes of the inflation have a relevant impact on the above sorts of risk.

Considering the interest rate risk as the basic uncertainty parameter to the determination of a bank asset liability management strategy, the crucial question that arises concerns the determination of the way through which this factor of uncertainty affects the profitability of the pre-specified strategy. The estimation of the expected return of the pre-specified strategy and of its variance can render a satisfactory response to the above question.

The use of Monte Carlo techniques constitutes a particular widespread approach for the estimation of the above information (expected return – variance of bank asset liability management strategies). Monte Carlo simulation consists in the development of various random scenarios for the uncertain variable (interest rates) and the estimation of the essential statistical measures (expected return and variance), which describe the effect of the interest rate risk to the selected strategy. The general procedure of implementation of Monte Carlo simulation based on the above is presented in Fig. 1.

During the first stage of the procedure the various categories of the interest rate risks are identified. The risk and the return of the various data of bank asset and liability are determined from the different forms of interest rates. For example, the investments of a bank to government or corporate bonds are determined from the interest rates that prevail in the bond market, which are affected so by the general economic environment as by the rules of demand and supply. Similarly, the deposits and loans of the bank are determined from the corresponding interest rates of deposits and loans, which are assigned by the bank according to the conditions that prevail to the bank market. At this stage, the categories of the interest rates, which constitute crucial uncertain variables for the analysis, are detected. The determined interest rates categories depend on the type of the bank. For example, for a decisive commercial bank, the deposit and loan interest rates have a role, whereas for an investment bank more emphasis is given



Asset Liability Management Decision Support System, Figure 1
General Monte Carlo simulation procedure for the evaluation of the asset liability management strategies

to the interest rates and the returns of various investment products (repos, bonds, interest-bearing notes, etc.).

After the determination of the various categories of interest rates, which determine the total interest rate risk, at the second stage of the analysis the statistical distribution that follows each of the pre-specified categories should be determined.

Having determined the statistical distribution that describes the uncertain variables of the analysis (interest rates), a series of random independent scenarios is developed, through a random number generator. Generally, the largest the number of scenarios that are developed, the more reliable conclusions can be derived. However, the computational effort increases significantly, since for each scenario the optimal asset liability strategy should be determined and moreover its evaluation for each other scenario should take place. Thus, the determination of the number volume N of simulations (scenarios), which will take place should be determined, taking into account both the reliability of the results and the available computational resources.

For each scenario s_i ($i = 1, 2, \ldots, N$) over the interest rates the optimal asset liability management strategy Υ_i is determined through the solution of the goal programming problem. It is obvious that this strategy is not expected to be optimal for each of the other scenarios s_j ($j \neq i$). Therefore the results obtained from the implementation of the strategy Y_i under the rest N–1 possible scenarios s_j should be evaluated. The evaluation of the results can be implemented from various directions. The most usual is the one that uses the return. Representing as r_{ij} the outcome (return) of the strategy Υ_i under the scenario s_j, the expected return $\bar{r}_i$ of the strategy can be easily determined based on all the other N–1 scenarios s_j ($j \neq i$), as follows:

$$\bar{r}_i = \frac{1}{N-1} \sum_{j=1, j\neq i}^{N} r_{ij} \tag{12}$$

At the same time, the variance σ_i^2 of the expected return can be determined as a risk measure of the strategy Y_i, as follows:

$$\sigma_i^2 = \frac{1}{N-1} \sum_{j=1, j\neq i}^{N} \left(r_{ij} - \bar{r}_i\right)^2 \tag{13}$$

These two statistical measures (average and variance) contribute to the extraction of useful conclusions concerning the expected efficiency of the asset liability management strategy, as well as the risks that it carries. Moreover, these two basic statistical measures can be used for the expansion of the analysis of the determination of other useful statistical information, such as the determination of the confidence interval for the expected return, the quantiles, etc.

Conclusions

The banking business has recently become more sophisticated due to technological expansion, economic development, creation of financial institutions and increased competition. Moreover, the mergers and acquisitions that have taken place the last years create large groups of banking institutions. The success of a bank depends mainly on the quality of its asset and liability management, since the latter deals with the efficient management of sources and uses of bank funds concentrating on profitability, liquidity, capital adequacy and risk factors.

It is obvious that in the last two decades modern finance has developed into a complex mathematically challenging field. Various and complicated risks exist in financial markets. For banks, interest rate risk is at the core of their business and managing it successfully is crucial to whether or not they remain profitable. Therefore, it has been essential the creation of the department of financial risk management within the banks. Asset liability management is associated with the changes of the interest rate risk. Although several models exist regarding asset liability management, most of them are focused on the general aspects and methodologies of this field and do not refer extensively to the hedging of bank interest rate risk through asset liability management. Thus, the main purpose of the present paper was to describe the development of a bank ALM decision support system, which gives the possibility to the decision maker to proceed to various scenarios of the economic process of the bank in order to monitor its financial situation and to determine the optimal strategic implementation of the composition of assets and liabilities. Moreover, we believe that the development of a bank asset liability management model that takes

into account the exogenous factors and the economic parameters of the market as well as the uncertainty of variations of the financial risks become essential.

Finally, despite the approaches described in this paper, little academic work has been done so far to develop a model for the management of assets and liabilities in the European banking industry. Based on the above we conclude that the quality of asset liability management in the European banking system has become significant as a resource of competitive advantage. Therefore, the development of new technological approaches in bank asset liability management in Europe is worth further research.

References

1. Baston RG (1989) Financial planning using goal programming. Long Range Plan 22(17):112–120
2. Booth GG (1972) Programming Bank Portfolios under Uncertainty: An Extension. J Bank Res 2:28–40
3. Bradley SP, Crane DB (1972) A Dynamic Model for Bond Portfolio Management. Manage Sci 19:139–151
4. Brodt AI (1978) Dynamic Balance Sheet Management Model for a Canadian Chartered Bank. J Bank Financ 2(3): 221–241
5. Carino DR, Kent T, Muyers DH, Stacy C, Sylvanus M, Turner AL, Watanabe K, Ziemba WT (1994) The Russell-Yasuda Kasai Model: An Asset/Liability Model for a Japanese Insurance Company Using Multistage Stochastic Programming. Interfaces 24:29–49
6. Chambers D, Charnes A (1961) Inter-Temporal Analysis and Optimization of Bank Portfolios. Manage Sci 7:393–410
7. Charnes A, Littlechild SC (1968) Intertemporal Bank Asset Choice with Stochastic Dependence. Systems Research Memorandum no.188, The Technological Institute, Nortwestern University, Evanston, Illinois
8. Charnes A, Thore S (1966) Planning for Liquidity in Financial Institution: The Chance Constrained Method. J Finance 21(4):649–674
9. Cohen KJ, Hammer FS (1967) Linear Programming and Optimal Bank Asset Management Decisions. J Financ 22: 42–61
10. Cohen KJ, Thore S (1970) Programming Bank Portfolios under Uncertainty. J Bank Res 2:28–40
11. Crane B (1971) A Stochastic Programming Model for Commercial Bank Bond Portfolio Management. J Finance Quant Anal 6:955–976
12. Eatman L, Sealey W (1979) A Multi-objective Linear Programming Model for Commercial bank Balance Sheet Management. J Bank Res 9:227–236
13. Eppen GD, Fama EF (1971) Three Asset Cash Balance and Dynamic Portfolio Problems. Manage Sci 17:311–319
14. Fielitz D, Loeffler A (1979) A Linear Programming Model for Commercial Bank Liquidity Management. Finance Manage 8(3):44–50
15. Giokas D, Vassiloglou M (1991) A Goal Programming Model for Bank Assets and Liabilities. Eur J Oper Res 50:48–60
16. Kallberg JG, White RW, Ziemba WT (1982) Short Term Financial Planning under Uncertainty. Manage Sci 28:670–682
17. Kosmidou K, Zopounidis C (2001) Bank Asset Liability Management Techniques: An Overview. In: Zopounidis C, Pardalos PM, Baourakis G (eds) Fuzzy Set Systems in Management and Economy. World Scientific Publishers, pp 255–268
18. Kosmidou K, Zopounidis C (2004) Combining Goal Programming Model with Simulation Analysis for Bank Asset Liability Management. Inf Syst Oper Res J 42(3):175–187
19. Kusy IM, Ziemba TW (1986) A Bank Asset and Liability Management model. Oper Res 34(3):356–376
20. Kvanli AH (1980) Financial planning using goal programming-OMEGA. Int J Manag Sci 8(2):207–218
21. Lee SM, Chesser DL (1980) Goal programming for portfolio selection. J Portf Manag 6:22–26
22. Lee SM, Lerro AJ (1973) Optimizing the portfolio selection for mutual funds. J Financ 28:1086–1101
23. Lifson KA, Blackman BR (1973) Simulation and Optimization Models for Asset Deployment and Funds Sources Balancing Profit Liquidity and Growth. J Bank Res 4(3):239–255
24. Markowitz HM (1959) Portfolio Selection. Efficient Diversification of Investments. Wiley, New York
25. Merton RC (1969) Lifetime portfolio selection under certainty: the continuous time case. Rev Eco Stat 3:373–413
26. Merton RC (1990) Continuous-Time Finance. Blackwell Publishers, Merton, UK
27. Mulvey JM, Vladimirou H (1989) Stochastic Network Optimization Models of Investment Planning. Annal Oper Res 20:187–217
28. Mulvey JM, Ziemba WT (1998) Asset and liability management systems for long-term investors: discussion of the issues. In: Ziemba W, Mulvey J (eds) Worldwide Asset and Liability Modelling. Cambridge University Press, Mulvey, UK, pp 3–38
29. Pogue GA, Bussard RN (1972) A Linear Programming Model for Short Term Financial Planning under Uncertainty. Sloan Manag Rev 13:69–98
30. Pyle DH (1971) On the Theory of Financial Intermediation. J Financ 26:737–746
31. Robertson M (1972) A Bank Asset Management Model. In: Eilon S, Fowkes TR (eds) Applications of Management Science in Banking and Finance. Gower Press, Epping, Essex, pp 149–158
32. Samuelson P (1969) Lifetime portfolio selection by dynamic stochastic programming. Rev Eco Stat (August), 239–246

33. Seshadri S, Khanna A, Harche F, Wyle R (1999) A method for strategic asset-liability management with an application to the federal home loan bank of New York. Oper Res 47(3):345–360
34. Sharma JK, Sharma DK, Adeyeye JO (1995) Optimal portfolio selection: A goal programming approach. Indian J Financ Res 7(2):67–76
35. Wolf CR (1969) A Model for Selecting Commercial Bank Government Security Portfolios. Rev Econ Stat 1:40–52

Assignment and Matching

AM

Dimitris Alevras
IBM Corporation, West Chester, USA

MSC2000: 90C35, 90C27, 90C10, 90C05

Article Outline

Keywords

Optimization; Integer programming; Graph theory; Marriage problem

Matching problems comprise an important set of problems that link the areas of graph theory and combinatorial optimization. The maximum cardinality matching problem (see below) is one of the first integer programming problems that was solved in polynomial time. Matchings are of great importance in graph theory (see [9]) as well as in combinatorial optimization (see e. g. [15]).

The matching problem and its variations arise in cases when we want to find an 'optimal' pairing of the members of two (not necessarily disjoint) sets. In particular, if we are given two sets of 'objects' and a 'weight' for each pair of objects, we want to match the objects into pairs in such a way that the total weight is maximal. In graph theory, the problem is defined on a graph $G = (V, E)$ where V is the node set of the graph, corresponding to the union of the two sets of objects, and E is the edge set of the graph corresponding to the possible pairs. A pair is possible if there exists an edge between the corresponding nodes. A *matching* M is a subset of the edges E with the property that each node in V is incident to at most one edge in M. If each node in V is met by exactly one edge in M, then M is called a *perfect matching*. There exist several versions of the matching problem, depending on whether the graph G is bipartite or not (i. e., the two sets of objects are disjoint or not), and on whether we want to find the maximum size (cardinality) or the maximum weight of the matching. The book [1] gives several applications of the matching problem.

Maximum Cardinality Bipartite Matching Problem

The graph G is *bipartite* if the node set V can be partitioned into two disjoint sets V_1 and V_2 such that no edge in E connects nodes from the same set. Finding a maximum cardinality matching on a bipartite graph can be solved by several efficient algorithms with a worst-case bound of $O(\sqrt{n}m)$, where n is the number of nodes and m the number of edges of the graph. See [1] for details.

Weighted Bipartite Matching Problem

This problem is known as the *assignment* or the *marriage* problem. In the traditional definition it is required that the sets V_1 and V_2 are of equal size, but even if not, one can add 'dummy' nodes to the smaller set to satisfy this condition. This problem can be formulated as a zero-one linear programming problem as follows:

$$\begin{cases} \min & \displaystyle\sum_{(u,v)\in E} f(u,v)x_{uv} \\ \text{s.t.} & \displaystyle\sum_{(u,v)\in E} x_{uv} = 1 \quad \text{for all } u \in V_1, \\ & \displaystyle\sum_{(u,v)\in E} x_{uv} = 1 \quad \text{for all } v \in V_2, \\ & x_{uv} \in \{0,1\} \quad \text{for all } u \in V_1,\ v \in V_2. \end{cases}$$

The *assignment problem* has the property that if solved as a linear programming problem in nonnegative x_{uv} it yields an integer solution, i. e., the zero-one integrality condition in the formulation is not necessary. This is

so because the constraint matrix of the equations is *totally unimodular*, i. e., the determinant of every square submatrix of it is 0 or ± 1. This means that if the right-hand sides of the equations are integer numbers, as is the case in the assignment problem, then the solution will be integer.

Linear programming algorithms are not as efficient as specialized algorithms for solving the assignment problem. The assignment problem is a special case of the minimum cost flow problem, and adaptations of algorithms for that problem that take into account the special structure of the assignment problem yield the most efficient algorithms. Probably the best known algorithm is the so called *Hungarian algorithm*, see [8], which is a primal-dual algorithm for the minimum cost flow problem. See [1] for details and other algorithms.

Variations of the bipartite matching include among others the order preserving assignment problem and the stable marriage problem. In the *order preserving assignment problem* the assignment must be such that a prespecified order among the objects of one of the node partitions is preserved. Although the linear programming formulation of this problem is more complicated than that of the assignment problem, the problem itself is easier to solve than the assignment problem and can be solved in $O(m)$ time where m is the number of edges in the graph; see [2,12]. In the *stable marriage problem* each object of one partition has a ranking (or preference) for each of the objects of the other partition, and the assignment must be such that there is no nonmatched pair of objects that its members prefer each other to the ones they are matched against. This problem can be solved in $O(n^2)$ time using a greedy algorithm (n is the number of nodes in one partition). See [1].

Weighted Matching Problem

The weighted matching problem can be formulated as a 0–1 programming problem as follows:

$$\begin{cases} \max & \displaystyle\sum_{(u,v)\in E} f(u,v)x_{uv} \\ \text{s.t.} & \displaystyle\sum_{(u,v)\in E} x_{uv} \le 1 \quad \text{for all } u \in V, \\ & x_{uv} \in \{0,1\} \quad \text{for all } (u,v) \in E. \end{cases}$$

Unlike the case of the assignment problem, relaxing the integrality constraints yields, in general, a fractional solution.

Maximum Cardinality Matching Problem

J. Edmonds showed in [5] that one more set of inequalities—the *odd-set constraints*—is needed in order to get a linear programming formulation of the matching problem. The odd-set or *blossom* inequalities are

$$\sum_{(u,v)\in E(U)} x_{uv} \le \left\lfloor \frac{|U|}{2} \right\rfloor, \quad \forall \text{odd } U \subseteq V,\ |U| \ge 3,$$

where $E(U)$ is the set of all edges in E with both end nodes in U. An odd set is a set of odd cardinality. See also [11].

Solving the matching problem on nonbipartite graphs is considerably more difficult than on bipartite ones. This is so because the path augmenting algorithms used in the case of bipartite matchings, may fail when a structure called blossom is encountered. Edmonds provided an $O(n^4)$ algorithm that would find an integer solution to the linear programming relaxation of the formulation (including the odd-set constraints) for any objective function, proving this way the completeness of the formulation. Several implementations that improved the performance of the algorithm have been proposed (see [1,10], among others) as well as data structures for the efficient implementation of such algorithms (see [3]). M. Grötschel and O. Holland [6] gave a cutting plane algorithm for the weighted matching problem, where they used an efficient separation algorithm to identify violated *blossom inequalities*, based on the algorithm of M.W. Padberg and M.R. Rao [14] for the *b-matching problem*.

The b-matching problem is an important generalization of the matching problem. In the b-matching problem each node $v \in V$ is met by no more than b_v edges; thus, in this context, the previous definition of matching corresponds to an 1-matching. A *perfect b-matching* is one in which each node $v \in V$ is met by exactly b_v edges. If it is permitted to chose an edge more than one times then the problem becomes a general integer program instead of a 0–1 program. The b-matching problem can be reduced to 1-matching problem on an appropriately constructed graph. Although this procedure is not polynomial in gen-

eral—and thus, Edmonds' algorithm can not be readily applied—the *b*-matching problem is polynomially solvable; see [14] and [7]. A linear inequality description for the integer *b*-matching problem is given in [15]. See also [11]. The perfect 0–1 2-matching problem is a relaxation of the *traveling salesman problem* (TSP). Solving the 0–1 2-matching problem yields a heuristic solution to the TSP which is an *NP*-hard problem; see [13].

See also

- ▶ Assignment Methods in Clustering
- ▶ Bi-Objective Assignment Problem
- ▶ Communication Network Assignment Problem
- ▶ Frequency Assignment Problem
- ▶ Maximum Partition Matching
- ▶ Quadratic Assignment Problem

References

1. Ahuja R, Magnanti T, Orlin J (1994) Network flows. Wiley, New York
2. Alevras D (1997) Order preserving assignments without contiguity. Discret Math 163:1–11
3. Ball MO, Derigs U (1983) An analysis of alternate strategies for implementing matching algorithms. Networks 13:517–549
4. Edmonds J (1965) Maximum matching and a polyhedron with (0, 1) vertices. J Res Nat Bureau Standards (B) 69B:125–130
5. Edmonds J (1965) Paths, trees, and flowers. Canad J Math 17:449–467
6. Grötschel M, Holland O (1985) Solving matching problems with linear programming. Math Program 33:243–259
7. Grötschel M, Lovasz L, Schrijver A (1988) Geometric algorithms and combinatorial optimization. Springer, Berlin
8. Kuhn HW (1955) The Hungarian method for the assignment problem. Naval Res Logist Quart 2:83–97
9. Lovasz L, Plummer M (1986) Matching theory. North-Holland, Amsterdam
10. Micali S, Vazirani VV (1980) An $O(\sqrt{(|v|)}|E|)$ algorithm for finding maximum matching in general graphs. IEEE Symp Found Computer Sci, pp 17–27
11. Nemhauser GL, Wolsey L (1988) Integer and combinatorial optimization. Wiley, New York
12. Padberg M, Alevras D (1994) Order-preserving assignments. Naval Res Logist 41:395–421
13. Padberg MW, Grötschel M (1985) Polyhedral theory. The Traveling Salesman Problem. Wiley, New York, pp 251–305
14. Padberg M, Rao MR (1982) Odd minimum cut-sets and b-matchings. Math Oper Res 7:67–80
15. Pulleyblank WR (1989) Polyhedral combinatorics. Optimization, vol 1 of Handbook Oper Res and Management Sci. North-Holland, Amsterdam, pp 371–446

Assignment Methods in Clustering

L. J. HUBERT[1], P. ARABIE[2]
[1] University Illinois, Champaign, USA
[2] Rutgers University, Newark, USA

MSC2000: 62H30, 90C27

Article Outline

Keywords

Combinatorial optimization; Quadratic assignment; Clustering

The use of *assignment methods* in the formulation of various optimization problems encountered in *clustering* and *classification*, can be introduced through the well-known *quadratic assignment* (QA) model (see [5] for a comprehensive discussion of most of the topics presented in this entry). In its most basic form the QA optimization task can be stated using two $n \times n$ matrices, say $\mathbf{P} = \{ p_{ij} \}$, and $\mathbf{Q} = \{ q_{ij} \}$, and the identification of a one-to-one function (or a permutation), $\rho(\cdot)$, on the first n integers, to optimize (either by minimizing or maximizing) the cross-product index

$$\Gamma(\rho) = \sum_{i,j} p_{\rho(i)\rho(j)} q_{ij}. \tag{1}$$

Typically, the main diagonal entries in $\mathbf{P}$ and $\mathbf{Q}$ are considered irrelevant and can be set equal to zero. For arbitrary matrices $\mathbf{P}$ and $\mathbf{Q}$, the cross product index in (1)

may be rewritten as

$$\sum_{i,j}\left(\frac{p_{\rho(i)\rho(j)}+p_{\rho(j)\rho(i)}}{2}\right)\left(\frac{q_{ij}+q_{ji}}{2}\right)$$
$$+\sum_{i,j}\left(\frac{p_{\rho(i)\rho(j)}-p_{\rho(j)\rho(i)}}{2}\right)\left(\frac{q_{ij}-q_{ji}}{2}\right),$$

indicating that the optimization of (1) jointly involves the *symmetric* ($[\mathbf{P}+\mathbf{P}']/2$ versus $[\mathbf{Q}+\mathbf{Q}']/2$) and *skew-symmetric* ($[\mathbf{P}-\mathbf{P}']/2$ versus $[\mathbf{Q}-\mathbf{Q}']/2$) components of both **P** and **Q**. Because of this separation of **P** and **Q** into symmetric and skew-symmetric components, it is possible in the context of the clustering/classification tasks to be discussed below, to assume that both **P** and **Q** are symmetric or that both are skew-symmetric.

In applications to clustering, the matrix **P** usually contains numerical proximity information between distinct pairs of the n objects from some given set $S = \{O_1, \ldots, O_n\}$ that is of substantive interest. If **P** is symmetric, p_{ij} ($= p_{ji}$) denotes the degree to which objects O_i and O_j are similar (and keyed as what is referred to as a *dissimilarity* [or as a *similarity*] measure if smaller [or larger] values reflect greater object similarity). If **P** is skew-symmetric, p_{ij} ($= -p_{ji}$) is an index of dominance (or *flow*) between objects O_i and O_j, with the sign reflecting the directionality of *dominance* and the absolute value indicating the degree. The (target) matrix **Q**, as developed in detail in the next section, will typically be fixed, with the specific pattern of entries characterizing the type of structure to be identified for the set S, e. g., a single object *cluster*, a *partition*, or a *partition hierarchy*. An optimal permutation, say, $\rho^*(\cdot)$, based on the cross-product index in (1) for a specific target matrix **Q** will identify the (salient) combinatorial structure sought.

The QA *optimization* task as formulated through (1) has an enormous literature that will not be reviewed here (for an up-to-date and comprehensive source on QA, see [11]). For current purposes, one might consider the optimization of (1) through a simple object interchange heuristic that would begin with some permutation (possibly chosen at random), and then implement local interchanges until no improvement in the index can be made. By repeatedly initializing such a process randomly, a distribution over a set of *local optima* can be achieved. At least within the context of clustering/classification, such a distribution may be highly relevant diagnostically for explaining whatever structure is inherent in the data matrix **P**, and possibly of even greater interest than the identification of just a single optimal permutation. In a related framework, there are considerable applications for the QA model in a confirmatory context where the distribution of $\Gamma(\rho)$ is constructed over all $n!$ possible permutations considered equally-likely, and the index value associated with some identified permutation is compared to this distribution. Most *nonparametric statistical methods* popular in the literature can be rephrased through the device of defining the matrices **P** and **Q** appropriately (see [5] for a comprehensive development of these special cases as well as approximation methods based on closed-form expressions for the first three moments of $\Gamma(\rho)$). A few of these applications will be briefly noted below.

Weighting Schemes for the Fixed (Target) Matrix Q

Single Cluster Statistics

To identify a *single* salient cluster of fixed size K (that can be varied by the user), consider **Q** to have the partitioned form

$$\mathbf{Q}=\begin{pmatrix}\mathbf{Q}_{11} & \mathbf{Q}_{12}\\ \mathbf{Q}_{21} & \mathbf{Q}_{22}\end{pmatrix},$$

where within each submatrix of the size indicated, the (off-diagonal) entries are constant:

$$\mathbf{Q}_{11}=\begin{pmatrix}0 & \cdots & q_{11}\\ \vdots & \ddots & \vdots\\ q_{11} & \cdots & 0\end{pmatrix}_{K\times K}$$

$$\mathbf{Q}_{12}=\begin{pmatrix}\vdots & \\ \cdots\ q_{12} & \cdots\\ \vdots & \end{pmatrix}_{K\times(n-K)}$$

$$\mathbf{Q}_{21}=\begin{pmatrix}\vdots & \\ \cdots\ q_{21} & \cdots\\ \vdots & \end{pmatrix}_{(n-K)\times K}$$

$$\mathbf{Q}_{22}=\begin{pmatrix}0 & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & 0\end{pmatrix}_{(n-K)\times(n-K)}$$

Depending on how the values for q_{11}, q_{12}, and q_{21} are defined, different indices can be generated that measure

the salience of the subset constructed by any permutation $\rho(\cdot)$, i. e., for the identified cluster $S_\rho \equiv \{O_{\rho(1)}, \ldots, O_{\rho(K)}\}$.

For symmetric **P**:

A) letting

$$q_{11} = \frac{1}{K(K-1)}, \quad q_{12} = q_{21} = 0,$$

the index $\Gamma(\rho)$ is the average proximity within the subset S_ρ and defines a measure of *cluster 'compactness'*;

B) letting

$$q_{11} = 0, \quad q_{12} = q_{21} = \frac{1}{2K(n-K)},$$

$\Gamma(\rho)$ is the average proximity between the subset S_ρ and its complement, and defines a measure of *cluster 'isolation'* for either S_ρ or $S - S_\rho$; alternatively, it can be considered a measure of *'separation'* between S_ρ or $S - S_\rho$;

C) by contrasting A) and B) as

$$q_{11} = \frac{1}{K(K-1)},$$
$$q_{12} = q_{21} = -\frac{1}{2K(n-k)},$$

$\Gamma(\rho)$ characterizes the salience of the subset S_ρ by a trade-off between compactness and isolation. The optimization of $\Gamma(\rho)$ based on these latter weights identifies a cluster that would be both relatively compact and isolated, whereas the emphasis in A) and B) are on clusters that may be either compact or isolated but not necessarily both.

For skew-symmetric **P**:

D) letting

$$q_{11} = 0, \quad q_{12} = \frac{1}{2K(n-K)}, \quad q_{21} = -q_{12},$$

the index $\Gamma(\rho)$ is the average dominance (or flow) from the subset S_ρ to its complement, minus the average dominance (or flow) from the complement to the subset. Thus, its optimization (e. g., maximization) identifies a subset of S whose members tend to dominate those in its complement (or where aggregate outflow exceeds aggregate inflow).

In a confirmatory comparison context, the single-cluster statistic $\Gamma(\rho)$ can be used to generate a number of nonparametric test statistics for comparing the difference between two independent groups. For example, suppose observations are available on n objects, $x_1, \ldots, x_n$, where the first K belong to group I and the last $n - K$ to group II. If the (now asymmetric) proximity matrix is defined as $\mathbf{P} = \{p_{ij}\}$, where $p_{ij} = 1$ if $x_j < x_i$ and $= 0$ if $x_j \geq x_i$ then the weighting scheme in B) gives (a simple linear transform of) the well-known *Mann–Whitney statistic* for comparing two-independent groups, i. e., if two observations are drawn at random from groups I and II, then $\Gamma(\rho_o)$, for ρ_o the identity permutation, is the probability that the group I observation is the larger. The distribution of $\Gamma(\rho)$ over all $n!$ permutations generates the null distribution against which the observed index $\Gamma(\rho_o)$ can be compared. Because of the structure of **Q**, this null distribution is based on all $n!/(K!(n - K)!)$ distinct subsets considered equally-likely to be formed from the collection of size n. (See [3, Chap. 7], for a more complete discussion of the two-independent sample problem in this type of nonparametric framework.)

Although single-cluster statistics that depend on the comparison of mean proximities may be the most obvious to consider, a number of possible alternatives can be constructed by varying the definition for the weight matrices in **Q**. For example, for symmetric **P**, if $\mathbf{Q}_{11}$ is (re)defined to have the form

$$\begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 1 & 0 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 \end{pmatrix},$$

with entries of all ones immediately above and below the main diagonal, and $q_{12} = q_{21} = 0$, the salience of S_ρ is now based on (twice) the sum of adjacent proximities along a path of *length* K considered in the object order $O_{\rho(1)} \leftrightarrow \cdots \leftrightarrow O_{\rho(K)}$. Or, if $\mathbf{Q}_{11}$ is (re)defined to have the form

$$\begin{pmatrix} 0 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 0 & 0 & \cdots & 0 & 0 \end{pmatrix},$$

and $q_{12} = q_{21} = 0$, the salience of S_ρ is now based on (twice) the sum of proximities between $O_{\rho(1)}$ and the

remaining objects $O_{\rho(2)}, \ldots, O_{\rho(K)}$ (this is called a *'star' cluster* of size K with object $O_{\rho(1)}$ as its center; see [10, Sect. 4.5.2] for a further discussion of clustering based on stars).

Partition Statistics

To identify a salient partition of S into M subsets, $S_1, \ldots, S_M$, of fixed sizes $n_1, \ldots, n_M$, respectively, consider $\mathbf{Q}$ to have the partitioned form

$$\mathbf{Q} = \begin{pmatrix} \mathbf{Q}_{11} & \mathbf{Q}_{12} & \cdots & \mathbf{Q}_{1M} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{Q}_{M1} & \mathbf{Q}_{M2} & \cdots & \mathbf{Q}_{MM} \end{pmatrix},$$

where the (off-diagonal) entries in each submatrix $\mathbf{Q}_{mm'}$ of size $n_m \times n_{m'}$, are all equal to a constant $q_{mm'}$, $1 \leq m, m' \leq M$. Again, depending on how these latter values are defined, a variety of different indices can be generated that now measure the salience of the partition generated by a permutation $\rho(\cdot)$. For a symmetric $\mathbf{P}$, three of the most popular alternatives are noted below that differ only in how the weights q_{mm}, $1 \leq m \leq M$, are defined, and which all assume $q_{mm'} = 0$ for $m \neq m'$:

a) $q_{mm} = 1$: each subset in a partition contributes in direct proportion to the number of object pairs it contains;
b) $q_{mm} = 1/(n_m(n_m - 1))$: each subset contributes equally irrespective of the number of objects (or object pairs) it contains;
c) $q_{mm'} = 1/n_m$: each subset contributes in direct proportion to the number of objects it contains.

In a confirmatory comparison context, the partition statistic $\Gamma(\rho)$ with weighting option c) can be used to construct a test-statistic equivalent to the common F-ratio in a *one-way analysis of variance* for assessing whether mean differences exist over K independent groups. Explicitly, suppose observations are available on n objects, $x_1, \ldots, x_n$, with the first n_1 belonging to group 1, the second n_2 belonging to group 2, and so on. If proximity is defined as $\mathbf{P} = \{p_{ij}\}$, where $p_{ij} = (x_i - x_j)^2$, then the weights in c) produce $\Gamma(\rho_o)$, for ρ_o the identity permutation, equal to twice the within group sum of squares. The distribution of $\Gamma(\rho)$ over all $n!$ permutations generates a distribution over all $n!/(n_1! \ldots n_M!)$ equally-likely ways the n observations can be grouped into subsets of sizes $n_1, \ldots, n_M$, and against which the observed index $\Gamma(\rho_o)$ can be compared. (See [9] for a more thorough discussion of thus evaluating a priori classifications.)

For a skew-symmetric $\mathbf{P}$, the partitioning of S would now be into M ordered subsets, $S_1 \prec \ldots \prec S_M$ of fixed sizes $n_1, \ldots, n_M$, with the most natural weights being $q_{mm} = 0$ for $1 \leq m \leq M$, $q_{mm'} = +1$ if $m < m'$, and $= -1$ if $m > m'$. Maximizing $\Gamma(\rho)$ is this case would be a search for an *ordered partition* in which objects in S_m tend to dominate those in $S_{m'}$ if $m < m'$, i. e., there are generally positive dominance values from a lower-placed subset to one that is higher.

There are several special cases of interest for the partition statistic:

i) for symmetric $\mathbf{P}$ and if for convenience it is assumed n is even and $n_m = 2$ for $1 \leq m \leq M$ (so, $n = 2M$), the weights in a) make $\Gamma(\rho)$ the index for a *matching* of the objects in S induced by $\rho(\cdot)$;
ii) if the proximity matrix $\mathbf{P}$ is itself constructed from a partition of S, then the index $\Gamma(\rho)$ can be interpreted as a measure of association for a *contingency table* defined by the n objects cross-classified using $\rho(\cdot)$ and the two partitions underlying $\mathbf{P}$ and $\mathbf{Q}$.

Depending on the choice of weights for $\mathbf{Q}$, and how proximity is defined in $\mathbf{P}$ based on its underlying partition, a number of well-known indices of association can be obtained: *Pearson's chi-square statistic*, *Goodman–Kruskal's* τ_b, and *Rand's index*. For a more complete discussion of these special cases, including the necessary definitions for $\mathbf{P}$, consult [5].

Partition Hierarchy Statistics

One straightforward strategy for extending QA to identify salient partition hierarchies having a specific form, begins with a given collection of T partitions of S, $\mathcal{P}_1, \ldots, \mathcal{P}_T$, that are hierarchically related. Here, $\mathcal{P}_1$ contains all n objects in n separate classes, $\mathcal{P}_T$ contains all n objects in one class, and $\mathcal{P}_{t+1}$ is formed from $\mathcal{P}_t$ for $t \geq 1$ by uniting one or more of the classes in the latter. If $\mathbf{Q} = \{q_{ij}\}$ is defined by $q_{ij} = \min\{t - 1: O_i, O_j \in \text{common object class in } \mathcal{P}_t\}$, then these latter entries satisfy the defining property of being an *ultrametric*, i. e., $q_{ij} \leq \max\{q_{ik}, q_{kj}\}$ for all $O_i, O_j, O_k \in S$ (see [2,10, Chap. 7] for an extensive discussion of ultrametrics). For symmetric $\mathbf{P}$, the optimization of $\Gamma(\rho)$ in (1) would be the search for a salient partition hierarchy having the generic form

defined by $\mathcal{P}_1, \ldots, \mathcal{P}_T$, and which optimizes the cross-product between the proximity information in **P** and the levels at which the object pairs are first placed into common classes in the hierarchy. It might be noted that both single clusters and partitions could be considered special cases of a partition hierarchy when $T = 3$ and the only nontrivial partition is $\mathcal{P}_2$, i. e., to obtain a single cluster, $\mathcal{P}_2$ can be defined by one subset of size K and $n - K$ subsets each of size one; to obtain a single partition, $\mathcal{P}_2$ merely has to be that partition with the desired number of classes and class sizes.

Alternative Assignment Indices

There are a variety of alternatives for replacing the cross-product in the QA index in (1) by a different function between the entries in **P** and **Q**. Depending on how the proximity information in **P** and the target given by **Q** are specified, one might adopt, for example, the sum of absolute differences, $\sum_{i,j} |p_{\rho(i)\rho(j)} - q_{ij}|$, or the sum of dichotomous indicators for equality, $\sum_{i,j} g(p_{\rho(i)\rho(j)}, q_{ij})$, where $g(x, y) = 1$ if $x = y$ and 0 otherwise, or even use *'bottleneck' measures* such as $\min_{i,j} p_{\rho(i)\rho(j)} q_{ij}$ or $\max_{i,j} p_{\rho(i)\rho(j)} q_{ij}$. Somewhat more well-developed in the literature than these possibilities (e. g., see [5, Chap. 5]) are generalizations of (1) that would maintain the basic cross-product structure but which would rely on higher-order functions of the entries in **P** and **Q** before the cross-products were taken. Again, variations would be possible, but two of the more obvious forms of extension are given below that depend solely on the order of the entries within **P** and within **Q**:

- *Three-argument functions*: Given **P** and **Q**, and letting $\text{sign}(x) = +1$ if $x > 0$, $= 0$ if $x = 0$, and $= -1$ if $x < 0$, define

$$\mathcal{A}(\rho) = \sum_{\substack{i \neq j \\ i \neq k}} \text{sign}(p_{\rho(i)\rho(j)} - p_{\rho(i)\rho(k)}) \, \text{sign}(q_{ij} - q_{ik}).$$

  The index $\mathcal{A}(\rho)$ can be interpreted as the difference between two counts, say $\mathcal{A}^+(\rho)$ and $\mathcal{A}^-(\rho)$, where $\mathcal{A}^+(\rho)$ (respectively, $\mathcal{A}^-(\rho)$) is the number of consistencies (inconsistencies) in the ordering of pairs of off-diagonal entries in $\{p_{\rho(i)\rho(j)}\}$ and their counterparts in $\{q_{ij}\}$, where the former pairs share a common (row) object $O_{\rho(i)}$.

- *Four-argument functions*: Define

$$\mathcal{B}(\rho) = \sum_{\substack{i \neq j \\ k \neq l}} \text{sign}(p_{\rho(i)\rho(j)} - p_{\rho(k)\rho(l)}) \, \text{sign}(q_{ij} - q_{kl}).$$

  Again, the index $\mathcal{B}(\rho)$ can be viewed as the difference between $\mathcal{B}^+(\rho)$ and $\mathcal{B}^-(\rho)$, where $\mathcal{B}^+(\rho)$ (respectively, $\mathcal{B}^-(\rho)$) is the number of consistencies (inconsistencies) in the ordering of pairs of off-diagonal entries in $\{p_{\rho(i)\rho(j)}\}$ and their counterparts in $\{q_{ij}\}$. In contrast to $\mathcal{A}(\rho)$, however, no common object need be present in the pairs of off-diagonal entries. The distinction between $\mathcal{A}(\rho)$ and $\mathcal{B}(\rho)$ in measuring the correspondence between **P** and **Q** rests on whether the proximity entries in **P** are strictly comparable only within rows (i. e., to what are called *row conditional proximity data*, e. g., see [1, p. 192]) or whether such comparisons make sense when performed across rows.

To illustrate the interpretation of $\mathcal{A}(\rho)$ and $\mathcal{B}(\rho)$ in the single cluster statistic context, suppose **Q** has the weight structure in A) that generated through (1) the measure of cluster compactness as the average within group proximity in $S_\rho = \{O_{\rho(1)}, \ldots, O_{\rho(K)}\}$. In using this specific target **Q** for $\mathcal{A}(\rho)$, the index is, in words, twice the difference between the number of instances in which a proximity for two objects both within S_ρ is greater than the proximity from one of these two objects to another in $S - S_\rho$, and the number of instances in which it is less. Depending on whether proximity is keyed as a similarity or a dissimilarity, a compact subset would be one for which $\mathcal{A}(\rho)$ is maximized or minimized, respectively. If instead, the weight structure for **Q** given in B) that defined the measure of cluster isolation, the index $\mathcal{A}(\rho)$ would now be twice the difference between the number of instances in which a proximity between two objects that span S_ρ and $S - S_\rho$ is greater than the proximity between two objects within S_ρ or within $S - S_\rho$ (where the latter have one member in common with the two that span S_ρ and $S - S_\rho$), and the number of instances in which it is less. Now, an isolated subset would be identified by maximizing or minimizing $\mathcal{A}(\rho)$ depending on the keying of proximity as a dissimilarity or similarity, respectively. For $\mathcal{B}(\rho)$, and the weight structure in A), the index is, in words, twice the difference between the number of instances in which a proximity for two objects both within S_ρ is greater than the prox-

imity between *any* two objects that span S_ρ and $S - S_\rho$ and the number of instances in which it is less. The index $\mathcal{B}(\rho)$ for the weight matrix in B) would be twice the difference between the number of instances in which a proximity between two objects that span S_ρ and $S - S_\rho$ is greater than the proximity between *any* two objects within S_ρ or within $S - S_\rho$.

In the partition context, a similar interpretation to the use of the single subset compactness measure would be present for $\mathcal{A}(\rho)$ and $\mathcal{B}(\rho)$ and for all of the three weighting options mentioned, but now all aggregated over the M subsets of the partition. In the partition hierarchy framework, the correspondence between $\{p_{\rho(i)\rho(j)}\}$ and $\mathbf{Q}$ is measured by the degree of consistency in the ordering of the object pairs by proximity and the ordering of the object pairs by the levels in which the objects are first placed into a common class.

In addition to replacing the QA index in (1) by the higher order functions adopted in $\mathcal{A}(\rho)$ and $\mathcal{B}(\rho)$ to effect a reliance only on the order properties of the entries within $\mathbf{P}$ and $\mathbf{Q}$, there are several other uses in a clustering/classification context for the definition of three- or four-argument functions. One alternative will be mentioned here that deals with what can be called the *generalized single cluster statistic*. Explicitly, suppose three- and four-argument function of the entries in $\mathbf{P}$ are denoted by $u(\cdot, \cdot, \cdot)$ and $r(\cdot, \cdot, \cdot, \cdot)$, respectively, and those in $\mathbf{Q}$ by $v(\cdot, \cdot, \cdot)$ and $s(\cdot, \cdot, \cdot, \cdot)$, and consider the general cross-product forms of

$$C(\rho) = \sum_{i,j,k} u(\rho(i), \rho(j), \rho(k)) v(i, j, k),$$

$$\mathcal{D}(\rho) = \sum_{i,j,k,l} r(\rho(i), \rho(j), \rho(k), \rho(l)) s(i, j, k, l).$$

It will be assumed here that both $v(\cdot, \cdot, \cdot)$ and $s(\cdot, \cdot, \cdot, \cdot)$ are merely indicator functions for a subset of size K, so $v(i, j, k) = 1$ if $1 \leq i, j, k \leq K$, and $= 0$ otherwise; $s(i, j, k, l) = 1$ if $1 \leq i, j, k, l \leq K$, and $= 0$ otherwise. Thus, the optimization of $C(\rho)$ or $\mathcal{D}(\rho)$ can be viewed as the search for a subset of size K with extreme values for the indices $\sum_{1 \leq i,j,k \leq K} u(\rho(i), \rho(j), \rho(k))$ or $\sum_{1 \leq i,j,k,l \leq K} r(\rho(i), \rho(j), \rho(k), \rho(l))$, and depending on how the functions $u(\cdot, \cdot, \cdot)$ and $r(\cdot, \cdot, \cdot, \cdot)$ are defined, a subset that is very salient with respect to the property that characterizes the latter.

A number of properties that may be desirable to optimize in a subset of size K have been considered (see [4] for a more complete discussion), of which the two listed below are directly relevant to the clustering/classification context:

i) a proximity matrix (with a dissimilarity keying) represents a perfect partition hierarchy if it satisfies the property of being an ultrametric: for all $1 \leq i, j, k \leq n$, $p_{ij} \leq \max\{p_{ik}, p_{kj}\}$, or equivalently, the two largest values among p_{ij}, p_{ik}, and p_{kj} are equal. Thus, if $u(\rho(i), \rho(j), \rho(k))$ equals the absolute difference between the two largest values among $p_{\rho(i)\rho(j)}$, $p_{\rho(i)\rho(k)}$, and $p_{\rho(j)\rho(k)}$, the minimization of $C(\rho)$ seeks a subset of size K that is as close to being an ultrametric as possible (as measured by $C(\rho)$);
ii) a proximity matrix (again, with a dissimilarity keying) represents a perfect *additive tree* where proximities can be reconstructed by minimum path lengths in a tree if they satisfy the four-point property: for all $1 \leq i, j, k, l \leq n$, $p_{ij} + p_{kl} \leq \max\{p_{ik} + p_{jl}, p_{il} + p_{jk}\}$, or equivalently, the largest two sums among $p_{ij} + p_{kl}$, $p_{ik} + p_{jl}$, and $p_{il} + p_{jk}$ are equal. Thus, if $r(\rho(i), \rho(j), \rho(k), \rho(l))$ equals the absolute difference between the two largest values among $p_{\rho(i)\rho(j)} + p_{\rho(k)\rho(l)}$, $p_{\rho(i)\rho(k)} + p_{\rho(j)\rho(l)}$, and $p_{\rho(i)\rho(l)} + p_{\rho(j)\rho(k)}$, the minimization of $\mathcal{D}(\rho)$ seeks a subset of size K that is as close to satisfying the four-point condition as possible (as measured by $\mathcal{D}(\rho)$).

Modifications of the Target Matrix Q

The optimization of an assignment index such as (1) assumes that the target matrix $\mathbf{Q}$ is fixed and given a priori. Based on this invariance, maximizing (1), for example, could be equivalently stated as the minimization of

$$\sum_{i,j} (p_{\rho(i)\rho(j)} - q_{ij})^2. \tag{2}$$

There has been a substantial recent literature (e. g., [6,7,8]) where not only is an optimal permutation, say $\rho^*(\cdot)$, sought that would minimize (2), but in which a specific target matrix $\mathbf{Q}$ is also constructed based on a collection of (linear inequality) constraints that would characterize some type of classificatory structure fitted to $\{p_{\rho(i)\rho(j)}\}$. The constraints imposed on $\mathbf{Q}$ are possibly based on the (sought for) permutation $\rho^*(\cdot)$.

In minimizing (2) but allowing the target matrix $\mathbf{Q}$ to itself be estimated, a typical iterative process would proceed as follows: on the basis of an initial target ma-

trix $\mathbf{Q}^{(0)}$, find a permutation, say $\rho^{(1)}(\cdot)$, to maximize the cross-product in (1). Using $\rho^{(1)}(\cdot)$, fit a target matrix $\mathbf{Q}^{(1)}$ to $\{p_{\rho^{(1)}(i)\rho^{(1)}(j)}\}$ minimizing (2). Continue the process for $\rho^{(t)}$ and $\mathbf{Q}^{(t)}$ for $t > 1$ until convergence. A variety of constraints for $\mathbf{Q}$ have been considered. Among these, there are

i) a sum of matrices each having what are called anti-Robinson forms (i. e., a matrix is *anti-Robinson* if within each row and column, the entries never decrease moving in any direction away from the main diagonal [6]);
ii) a sum of ultrametric matrices (characterized by the ultrametric condition given earlier [7]);
iii) a sum of additive tree matrices (again, as characterized by the four-point condition given earlier [7]);
iv) *unidimensional scales* (i. e., a matrix is a *linear unidimensional scale* if its entries can be given by $\{|x_j - x_i| + c\}$, where the estimated coordinates are $x_1 \leq \cdots \leq x_n$ and c is an estimated constant [8]); and
v) *circular unidimensional scales* (i. e., a matrix is so characterized if it can be represented as $\{\min\{|x_j - x_i|, x_0 - |x_j - x_i|\} + c\}$, where $x_1 \leq \cdots \leq x_n$, x_0 is the circumference of the circular structure, and c is an estimated constant [8]).

See also

- ▶ Assignment and Matching
- ▶ Bi-Objective Assignment Problem
- ▶ Communication Network Assignment Problem
- ▶ Frequency Assignment Problem
- ▶ Maximum Partition Matching
- ▶ Quadratic Assignment Problem

References

1. Carroll JD, Arabie P (1998) Multidimensional scaling. In: Birnbaum MH (ed) Measurement, judgement, and decision making. Handbook Perception and Cognition. Acad Press, New York, pp 179–250
2. De Soete G, Carroll JD (1996) Tree and other network models for representing proximity data. In: Arabie P, Hubert LJ, De Soete G (eds) Clustering and classification. World Sci, Singapore, pp 157–198
3. Gibbons JD (1971) Nonparametric statistical inference. McGraw-Hill, New York
4. Hubert LJ (1980) Analyzing proximity matrices: The assessment of internal variation in combinatorial structure. J Math Psych 21:247–264
5. Hubert LJ (1987) Assignment methods in combinatorial data analysis. M. Dekker, New York
6. Hubert LJ, Arabie P (1994) The analysis of proximity matrices through sums of matrices having (anti-)Robinson forms. British J Math Statist Psych 47:1–40
7. Hubert LJ, Arabie P (1995) Iterative projection strategies for the least-squares fitting of tree structures to proximity data. British J Math Statist Psych 48:281–317
8. Hubert LJ, Arabie P, Meulman J (1997) Linear and circular unidimensional scaling for symmetric proximity matrices. British J Math Statist Psych 50:253–284
9. Mielke PW, Berry KJ, Johnson ES (1976) Multi-response permutation procedures for a priori classifications. Comm Statist A5:1409–1424
10. Mirkin B (1996) Mathematical classification and clustering. Kluwer, Dordrecht
11. Pardalos PM, Wolkowicz H (eds) (1994) Quadratic assignment and related problems. DIMACS, Amer. Math. Soc., Providence, RI

Asymptotic Properties of Random Multidimensional Assignment Problem

PAVLO A. KROKHMAL
Department of Mechanical and Industrial Engineering, The University of Iowa, Iowa City, USA

MSC2000: 90C27, 34E05

Article Outline

Keywords and Phrases

Multidimensional assignment problem; Random assignment problem; Expected optimal value; Asymptotical analysis; Convergence bounds

Introduction

The Multidimensional Assignment Problem (MAP) is a higher dimensional version of the two-dimensional, or Linear Assignment Problem (LAP) [24]. If a classical textbook formulation of the Linear Assignment Problem is to find an optimal assignment of "N jobs to M workers", then, for example, the 3-dimensional Assignment Problem can be interpreted as finding an optimal assignment of "N jobs to M workers in K time slots", etc. In general, the objective of the MAP is to find tuples of elements from given sets, such that the total cost of the tuples is minimized. The MAP was first introduced by Pierskalla [26], and since then has found numerous applications in the areas of data association [4], image recognition [31], multisensor multitarget tracking [18,27], tracking of elementary particles [28], etc. For a discussion of the MAP and its applications see, for example, [7] and references therein.

Without loss of generality, a d-dimensional axial MAP can be written in a form where each dimension has the same number n of elements, i. e.,

$$\min_{x\in\{0,1\}^{n^d}} \left\{ \sum_{\substack{i_k\in\{1,\dots,n\}\\ k\in\{1,\dots,d\}}} c_{i_1\cdots i_d} x_{i_1\cdots i_d} \,\middle|\, \sum_{\substack{i_k\in\{1,\dots,n\}\\ k\in\{1,\dots,d\}\setminus j}} x_{i_1\cdots i_d} = 1, \; i_j = 1,\dots,n,\; j=1,\dots,d \right\} . \tag{1}$$

An instance of the MAP with different numbers of elements in each dimension, $n_1 \geq n_2 \geq \cdots \geq n_d$, is reducible to form (1) by introduction of dummy variables.

Problem (1) admits the following geometric interpretation: given a d-dimensional cubic matrix, find such a permutation of its rows and columns that the sum of the diagonal elements is minimized (which explains the term "axial"). This rendition leads to an alternative formulation of the MAP (1) in terms of permutations $\pi_1,\dots,\pi_{d-1}$ of numbers 1 to n, i. e., one-to-one mappings $\pi_i : \{1,\dots,n\} \mapsto \{1,\dots,n\}$,

$$\min_{\pi_1,\dots,\pi_{d-1}\in\Pi^n} \sum_{i=1}^{n} c_{i,\pi_1(i),\dots,\pi_{d-1}(i)} ,$$

where Π^n is the set of all permutations of the set $\{1,\dots,n\}$. A feasible solution to the MAP (1) can be conveniently described by specifying its cost,

$$z = c_{i_1^{(1)}\cdots i_d^{(1)}} + c_{i_1^{(2)}\cdots i_d^{(2)}} + \cdots + c_{i_1^{(n)}\cdots i_d^{(n)}} , \tag{2}$$

where $\left(i_j^{(1)}, i_j^{(2)}, \dots, i_j^{(n)}\right)$ is a permutation of the set $\{1,2,\dots,n\}$ for every $j = 1,\dots,d$. In contrast to the LAP that represents a $d = 2$ special case of the MAP (1) and is polynomially solvable [7], the MAP with $d \geq 3$ is generally NP-hard, a fact that follows from reduction of the 3-dimensional matching problem (3DM) [8].

Despite its inherent difficulty, several exact and heuristic algorithms [1,6,11,25] have been proposed to this problem. Most of these algorithms rely, at least partly, on repeated local searches in neighborhoods of feasible solutions, which brings about the question of how the number of local minima in a MAP impact these solution algorithms. Intuitively, if the number of local minima is small then one may expect better performance from meta-heuristic algorithms that rely on local neighborhood searches. A solution landscape is considered to be rugged if the number of local minima is exponential with respect to the dimensions of the problem [21]. Evidence in [5] showed that ruggedness of the solution landscape has a direct impact on the effectiveness of the simulated annealing heuristic in solving at least one other hard problem, the quadratic assignment problem. Thus, one of the issues that we address below is estimation of the expected number $\mathsf{E}[M]$ of local minima in random MAPs with respect to different local neighborhoods.

Another problem that we discuss is the behavior of the expected optimal value $Z^*_{d,n}$ of random large-scale MAPs, whose assignment costs are assumed to be independent identically distributed (iid) random variables from a given continuous distribution.

During the last two decades, expected optimal values of random assignment problems have been studied intensively in the context of random LAP. Perhaps, the most widely known result in this area is the conjecture by Mézard and Parisi [17] that the expected optimal value $\mathsf{E}[L_n] := Z^*_{2,n}$ of a LAP of size n with iid uniform or exponential with mean 1 cost coefficients satisfies $\lim_{n\to\infty} \mathsf{E}[L_n] = \frac{\pi^2}{6}$. In fact, this conjecture was preceded by an upper bound on the expected optimal value of the LAP with uniform (0,1) costs: $\limsup_{n\to\infty} L_n \leq 3$ due to

Walkup [32], which was soon improved by Karp [12]: $\limsup_{n\to\infty} L_n \le 2$. A lower bound on the limiting value of L_n was first provided by Lazarus [14]: $\liminf_{n\to\infty} L_n \ge 1 + e^{-1} \approx 1.37$, and then has been improved to 1.44 by Goemans and Kodialam [9] and 1.51 by Olin [20]. Experimental evidence in support of the Mézard-Parisi conjecture was provided by Pardalos and Ramakrishnan [22]. Recently, Aldous [2] has shown that indeed $\lim_{n\to\infty} \mathsf{E}[L_n] = \frac{\pi^2}{6}$, thereby proving the conjecture. Another conjecture due to Parisi [23] stating that the expected optimal value of a random LAP of finite size n with exponentially distributed iid costs is equal to $\mathsf{E}[L_n] = Z^*_{2,n} = \sum_{i=1}^{n} i^{-2}$ has been proven independently in [16] and [19].

Our work contributes to the existing literature on random assignment problems by establishing the limiting value and asymptotic behavior of the expected optimal cost $Z^*_{d,n}$ of random MAP with iid cost coefficients for a broad class of continuous distributions. The presented approach is constructive in the sense that it allows for deriving converging asymptotical lower and upper bounds for $Z^*_{d,n}$, as well as for estimating the rate of convergence for $Z^*_{d,n}$ in special cases.

Expected Optimal Value of Random MAP

Our approach to determining the asymptotic behavior of the expected optimal cost $Z^*_{d,n}$ of an MAP (1) with random cost coefficients involves analysis of the so-called *index tree*, a graph structure that represents the set of feasible solutions of the MAP. First introduced by Pierskalla [26], the index tree graph $\mathcal{G} = (V, E)$ of the MAP (1) has a set of vertices V which is partitioned into n levels[1] and a distinct *root node*. A node at level j of the graph represents an assignment $(i_1, \ldots, i_d)$ with $i_1 = j$ and cost $c_{ji_2\cdots i_d}$, whereby each level contains $\kappa = n^{d-1}$ nodes. The set E of arcs in the index tree graph is constructed in such a way that any feasible solution of the MAP (1) can be represented as a path connecting the root node to a leaf node at level n (such a path is called a *feasible* path); evidently, the index tree contains $n!^{d-1}$ feasible paths, by the number of feasible solutions of the MAP (1).

The index tree representation of MAP aids in construction of lower and upper bounds for the expected optimal cost of MAP (1) with random iid costs via the following lemmata [10].

Lemma 1. *Given the index tree graph $\mathcal{G} = (V, E)$ of $d \ge 3$, $n \ge 3$ MAP whose assignment costs are iid random variables from an absolutely continuous distribution, construct set $\mathcal{A} \subset V$ by randomly selecting α different nodes from each level of the index tree. Then, $\mathcal{A}$ is expected to contain a feasible solution of the MAP if*

$$\alpha = \left\lceil \frac{n^{d-1}}{n!^{\frac{d-1}{n}}} \right\rceil . \tag{3}$$

Lemma 2. *For a $d \ge 3$, $n \ge 3$ MAP whose cost coefficients are iid random variables from an absolutely continuous distribution F with existing first moment, define*

$$\underline{Z}^*_{d,n} := n\mathsf{E}_F\left[X_{(1|\kappa)}\right] \quad \text{and} \quad \overline{Z}^*_{d,n} := n\mathsf{E}_F\left[X_{(\alpha|\kappa)}\right] , \tag{4}$$

where $X_{(i|\kappa)}$ is the ith order statistic of $\kappa = n^{d-1}$ iid random variables with distribution F, and parameter α is determined as in (3). Then, $\underline{Z}^*_{d,n}$ and $\overline{Z}^*_{d,n}$ constitute lower and upper bounds for the expected optimal cost $Z^*_{d,n}$ of the MAP, respectively: $\underline{Z}^*_{d,n} \le Z^*_{d,n} \le \overline{Z}^*_{d,n}$.

Proofs of the lemmas are based on the probabilistic method [3] and can be found in [10]. In particular, the proof of Lemma 2 considers a set $\mathcal{A}_{\min}$ that is constructed by selecting from each level of the index tree α nodes with the smallest costs among the κ nodes at that level. The continuity of distribution F ensures that assignment costs in the MAP (1) are all different almost surely, hence locations of the nodes that comprise the set $\mathcal{A}_{\min}$ are random with respect to the array of nodes in each level of $\mathcal{G}(V, E)$. In the remainder of the paper, we always refer to α and κ as defined above.

By definition, the parameter $\kappa = n^{d-1}$ approaches infinity whenever n or d does; this allows us to denote the corresponding cases by $\kappa \xrightarrow{n} \infty$ and $\kappa \xrightarrow{d} \infty$, respectively. If certain statement holds for both cases of $n \to \infty$ and $d \to \infty$, we indicate this by $\kappa \xrightarrow{n,d} \infty$. The behavior of quantity α (3) when n or d increases is more contrasting. In the case $n \to \infty$ it approaches a finite limiting value,

$$\alpha \to \alpha^* := \lceil e^{d-1} \rceil, \quad \kappa \xrightarrow{n} \infty, \tag{5}$$

[1] In the general case of MAP with n_i elements in dimension $i = 1, \ldots, d$, the index graph would contain n_1 levels.

while in the case of fixed n and unbounded d it increases exponentially:

$$\alpha \sim \kappa^{\gamma_n}, \quad \kappa \xrightarrow{d} \infty, \quad \text{where} \quad \gamma_n = 1 - \frac{\ln n!}{n \ln n}\,, \tag{6}$$

and it is important to observe that $0 < \gamma_n < \frac{1}{2}$ for $n \geq 3$ [13].

The presented lemmata addresses MAPs with $d \geq 3, n \geq 3$. The case $d = 2$ represents, as noted earlier, the Linear Assignment Problem, whose asymptotic behavior is distinctly different from that of MAPs with $d \geq 3$. It can be shown that in the case of $d = 2$ Lemmas 1 and 2 produce only trivial bounds that are rather inefficient in determining the asymptotic behavior of the expected optimal value of the LAP within the presented approach. In the case $n = 2$ the costs of feasible solutions to the MAP (1) have the form

$$z = c_{i_1^{(1)} \cdots i_d^{(1)}} + c_{i_1^{(2)} \cdots i_d^{(2)}}, \quad \text{where} \quad i_j^{(1)}, i_j^{(2)} \in \{1, 2\}, i_j^{(1)} \neq i_j^{(2)}\,,$$

and consequently are iid random variables with distribution F_2, which is the convolution of F with itself: $F_2 = F * F$ [11]. This fact allows for computing the expected optimal value of $n = 2$ MAP exactly, without resorting to bounds (4):

$$Z^*_{d,2} = \mathsf{E}_{F*F}\left[X_{(1|2!^{d-1})}\right]. \tag{7}$$

In the general case $d \geq 3, n \geq 3$ the main challenge is constituted by computation of the upper bound $\overline{Z}^*_{d,n} = n\mathsf{E}_F\left[X_{(\alpha|\kappa)}\right]$, where $X_{(\alpha|\kappa)}$ is the α-th order statistic among κ independent F-distributed random variables. The subsequent analysis relies on representation of $\overline{Z}^*_{d,n}$ in the form

$$\overline{Z}^*_{d,n} = \frac{n\Gamma(\kappa+1)}{\Gamma(\alpha)\Gamma(\kappa-\alpha+1)} \cdot \int_0^1 F^{-1}(u) u^{\alpha-1}(1-u)^{\kappa-\alpha}\,\mathrm{d}u\,, \tag{8}$$

where F^{-1} denotes the inverse of the c.d.f. F of the the distribution of assignment costs in MAP (1). While it is practically impossible to evaluate the integral in (8) exactly in the general case, its asymptotic behavior for large n and d can be determined for a wide range of distributions F. For instance, in the case when distribution F has a finite left endpoint of its support set, the asymptotic behavior of the integral in (8) is obtained by means of the following

Lemma 3. *Let function $h(u)$ have the following asymptotic expansion at $0+$,*

$$h(u) \sim \sum_{s=0}^{\infty} a_s u^{(s+\lambda-\mu)/\mu}, \quad u \to 0+\,, \tag{9}$$

where $\lambda, \mu > 0$. Then for any positive integer m one has

$$\int_0^1 h(u) u^{\alpha-1}(1-u)^{\kappa-\alpha}\,\mathrm{d}u = \sum_{s=0}^{m-1} a_s \phi_s(\kappa) + \mathcal{O}\big(\phi_m(\kappa)\big), \quad \kappa \xrightarrow{n,d} \infty, \tag{10}$$

where $\phi_s(\kappa) = \mathrm{B}\big(\frac{s+\lambda}{\mu} + \alpha - 1,\ \kappa - \alpha + 1\big)$, $s = 0, 1, \ldots$, provided that the integral is absolutely convergent for $\kappa = \alpha = 1$.

Above, $\mathrm{B}(x, y)$ is the Beta function. Using similar results for the cases when the support set of distribution F is unbounded from below, we obtain that the limiting behavior of the expected optimal value $Z^*_{d,n}$ of random MAP is determined by the location of the left endpoint of the support of F [13].

Theorem 1. Expected Optimal Value of Random MAP *Consider a $d \geq 3, n \geq 2$ MAP (1) with cost coefficients that are iid random variables from an absolutely continuous distribution F with existing first moment. If the distribution F satisfies either of the following conditions,*

1. $F^{-1}(u) = F^{-1}(0+) + \mathcal{O}(u^{\beta}), u \to 0+, \beta > 0$
2. $F^{-1}(u) \sim -\nu u^{-\beta_1}\big(\ln \frac{1}{u}\big)^{\beta_2}, \quad u \to 0+, 0 \leq \beta_1 < 1, \beta_2 \geq 0, \beta_1 + \beta_2 > 0, \nu > 0$

where $F^{-1}(0+) = \lim_{u \to 0+} F^{-1}(u)$, the expected optimal value of the MAP satisfies

$$\lim Z^*_{d,n} = \lim nF^{-1}(0+)\,,$$

where both limits are taken at either $n \to \infty$ or $d \to \infty$.

The obtained results can be readily employed to construct upper and lower asymptotical bounds for the expected optimal value of MAP when one of the parameters n or d is large but finite. The following statement follows directly from Lemma 3 and Theorem 1.

Corollary 1. *Consider a $d \geq 3, n \geq 3$ MAP (1) with cost coefficients that are iid random variables from an absolutely continuous distribution with existing first moment. Let $a \in \mathbb{R}$ be the left endpoint of the support set of this distribution, $a = F^{-1}(0+)$, and assume that the inverse $F^{-1}(u)$ of the c.d.f. $F(u)$ of the distribution is such that*

$$F^{-1}(u) \sim a + \sum_{s=1}^{\infty} a_s u^{s/\mu}, \quad u \to 0+, \mu > 0\,. \tag{11}$$

Then, for any integer $m \geq 1$, lower and upper bounds $\underline{Z}^*_{d,n}$, $\overline{Z}^*_{d,n}$ (4) on the expected optimal cost $Z^*_{d,n}$ of the MAP can be asymptotically evaluated as

$$\underline{Z}^*_{d,n} = an + \sum_{s=1}^{m-1} a_s \frac{n\Gamma(\kappa+1)\Gamma\left(\frac{s}{\mu}+1\right)}{\Gamma\left(\kappa+\frac{s}{\mu}+1\right)} + \mathcal{O}\left(n\frac{\Gamma(\kappa+1)\Gamma\left(\frac{m}{\mu}+1\right)}{\Gamma\left(\kappa+\frac{m}{\mu}+1\right)}\right), \kappa \xrightarrow{n,d} \infty\,, \tag{12a}$$

$$\overline{Z}^*_{d,n} = an + \sum_{s=1}^{m-1} a_s \frac{n\Gamma(\kappa+1)\Gamma\left(\frac{s}{\mu}+\alpha\right)}{\Gamma(\alpha)\Gamma\left(\kappa+\frac{s}{\mu}+1\right)} + \mathcal{O}\left(n\frac{\Gamma(\kappa+1)\Gamma\left(\frac{m}{\mu}+\alpha\right)}{\Gamma(\alpha)\Gamma\left(\kappa+\frac{m}{\mu}+1\right)}\right), \kappa \xrightarrow{n,d} \infty\,. \tag{12b}$$

It can be shown that the lower and upper bounds defined by (12a, 12b) are convergent, i. e., $|\overline{Z}^*_{d,n} - \underline{Z}^*_{d,n}| \to 0$, $\kappa \xrightarrow{n,d} \infty$, whereas the corresponding asymptotical bounds for the case of distributions with support unbounded from below may be divergent in the sense that $|\overline{Z}^*_{d,n} - \underline{Z}^*_{d,n}| \nrightarrow 0$ when $\kappa \xrightarrow{n,d} \infty$.

The asymptotical representations (12a, 12b) for the bounds $\underline{Z}^*_{d,n}$ and $\overline{Z}^*_{d,n}$ are simplified when the inverse F^{-1} of the c.d.f. of the distribution has a regular power series expansion in the vicinity of zero. Assume, for example, that function F^{-1} can be written as

$$F^{-1}(u) = a_1 u + \mathcal{O}(u^2), \quad u \to 0+\,. \tag{13}$$

It is then easy to see that for $n \gg 1$ and d fixed the expected optimal value of the MAP is asymptotically bounded as

$$\frac{a_1}{n^{d-2}} + \mathcal{O}\left(\frac{1}{n^{d-1}}\right) \leq Z^*_{d,n} \leq \frac{a_1\lceil e^{d-1}\rceil}{n^{d-2}} + \mathcal{O}\left(\frac{1}{n^{d-1}}\right), \quad n \to \infty\,, \tag{14}$$

which immediately yields the rate of convergence to zero for $Z^*_{d,n}$ as n approaches infinity:

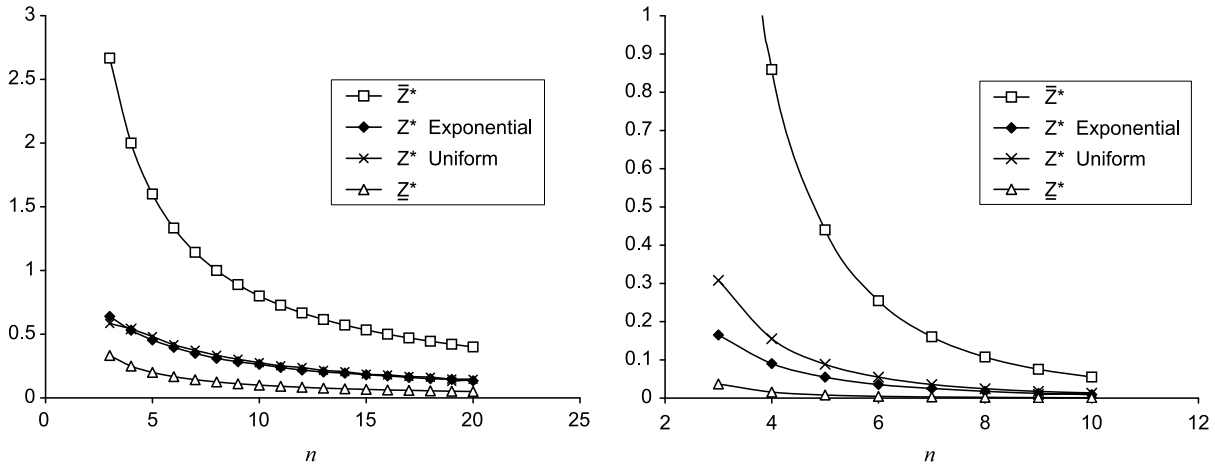
Corollary 2. *Consider a $d \geq 3, n \geq 3$ MAP (1) with cost coefficients that are iid random variables from an absolutely continuous distribution with existing first moment. Let the inverse F^{-1} of the c.d.f. of the distribution satisfy (13). Then, for a fixed d and $n \to \infty$ the expected optimal value $Z^*_{d,n}$ of the MAP converges to zero as $\mathcal{O}\left(n^{-(d-2)}\right)$.*

For example, the expected optimal value of 3-dimensional ($d = 3$) MAP with uniform $U(0,1)$ or exponential distributions converges to zero as $\mathcal{O}(n^{-1})$ when $n \to \infty$.

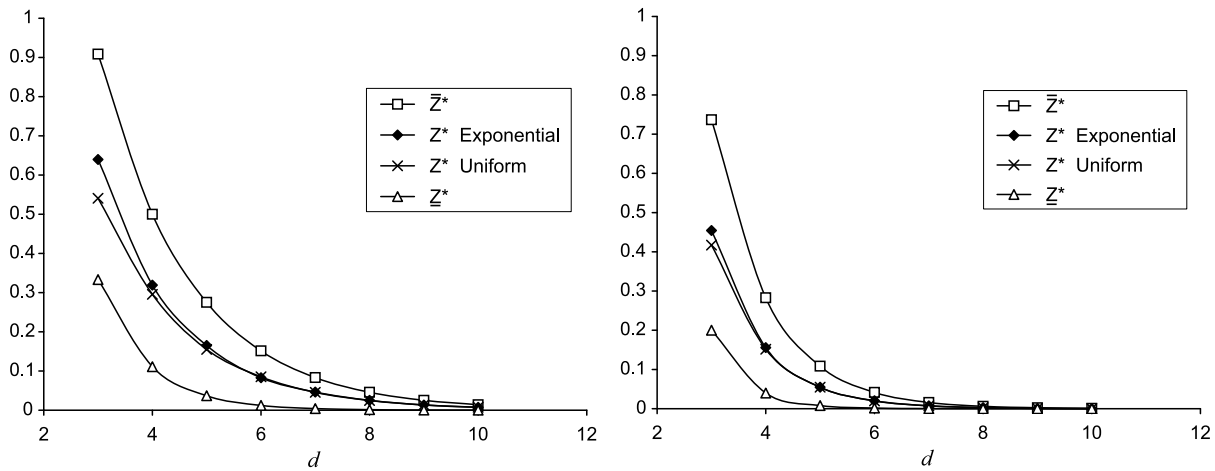
We illustrate the tightness of the developed bounds (12a, 12b) by comparing them to the computed expected optimal values of MAPs with coefficients $c_{i_1 \cdots i_d}$ drawn from the uniform $U(0,1)$ distribution and exponential distribution with mean 1. It is elementary that the inverse functions $F^{-1}(\cdot)$ of the c.d.f.'s for both these distributions are representable in form (13) with $a_1 = 1$.

The numerical experiments involved solving multiple instances of randomly generated MAPs with the number of dimensions d ranging from 3 to 10, and the number n of elements in each dimension running from 3 to 20. The number of instances generated for estimation of the expected optimal value of the MAP with a given distribution of cost coefficients varied from 1000 (for smaller values of d and n) to 50 (for problems with largest n and d).

To solve the problems to optimality, we used a branch-and-bound algorithm that navigated through the index tree representation of the MAP. Figures 1 and 2 display the obtained expected optimal values of MAP with uniform and exponential iid cost coefficients when d is fixed at $d = 3$ or 5 and $n = 3, \ldots, 20$, and when $n = 3$ or 5 and d runs from 3 to 10. This "asymmetry" in reporting of the results is explained by



Asymptotic Properties of Random Multidimensional Assignment Problem, Figure 1
Expected optimal value $Z^*_{d,n}$, lower and upper bounds $\underline{Z}^*_{d,n}$, $\overline{Z}^*_{d,n}$ of an MAP with fixed $d = 3$ (*left*) and $d = 5$ (*right*) for uniform $U(0, 1)$ and exponential (1) distributions



Asymptotic Properties of Random Multidimensional Assignment Problem, Figure 2
Expected optimal value $Z^*_{d,n}$, lower and upper bounds $\underline{Z}^*_{d,n}$, $\overline{Z}^*_{d,n}$ of an MAP with fixed $n = 3$ (*left*) and $n = 5$ (*right*) for uniform $U(0, 1)$ and exponential(1) distributions

the fact that the implemented branch-and-bound algorithm based on index tree is more efficient in solving "shallow" MAPs, i. e., instances that have larger n and smaller d. The solution times varied from several seconds to 20 hours on a 2GHz PC.

The conducted numerical experiments suggest that the constructed lower and upper bounds for the expected optimal cost of random MAPs are quite tight, with the upper bound $\overline{Z}^*_{d,n}$ being tighter for the case of fixed n and large d (see Figs. 1, 2).

Expected Number of Local Minima in Random MAP

Local Minima and p-exchange Neighborhoods in MAP

As it has been mentioned in the Introduction, we consider local minima of a MAP with respect to a local neighborhood, in the sense of [15]. For any $p = 2, \ldots, n$, we define the p-exchange local neighborhood $\mathcal{N}_p(i)$ of the ith feasible solu-

tion $\{i_1^{(1)} \cdots i_d^{(1)}, \ldots, i_1^{(n)} \cdots i_d^{(n)}\}$ of the MAP (1) as the set of solutions obtained from i by permuting p or less elements in one of the dimensions $1, \ldots, d$. More formally, $\mathcal{N}_p(i)$ is the set of n-tuples $\{j_1^{(1)} \cdots j_d^{(1)}, \ldots, j_1^{(n)} \cdots j_d^{(n)}\}$ such that $\{j_k^{(1)}, \ldots, j_k^{(n)}\}$ is a permutation of $\{1, \ldots, n\}$ for all $1 \le k \le d$, and, furthermore, there exists only one $k_0 \in \{1, \ldots, d\}$ such that

$$2 \le \sum_{r=1}^{n} \bar{\delta}_{i_{k_0}^{(r)} j_{k_0}^{(r)}} \le p, \quad \text{while} \quad \sum_{r=1}^{n} \bar{\delta}_{i_k^{(r)} j_k^{(r)}} = 0 \quad \text{for all} \quad k \in \{1, \ldots, d\} \backslash k_0 \,, \tag{15}$$

where $\bar{\delta}_{ij}$ is the negation of the Kroneker delta, $\bar{\delta}_{ij} = 1 - \delta_{ij}$. As an example, consider the following feasible solution to a $d = 3$, $n = 3$ MAP: $\{111, 222, 333\}$. Then, one of its 2-exchange neighbors is $\{111, 322, 233\}$, another one is $\{131, 222, 313\}$; a 3-exchange neighbor is given by $\{311, 122, 233\}$, etc. Evidently, one has $\mathcal{N}_p \subset \mathcal{N}_{p+1}$ for $p = 2, \ldots, n-1$.

Proposition 1. *For any $p = 2, \ldots, n$, the size $|\mathcal{N}_p|$ of the p-exchange local neighborhood of a feasible solution of a MAP (1) is equal to*

$$|\mathcal{N}_p| = d \sum_{k=2}^{p} D(k) \binom{n}{k}, \quad \text{where} \quad D(k) = \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j! \,. \tag{16}$$

The quantity $D(k)$ in (16) is known as the number of *derangements* of a k-element set [29], i. e., the number of permutations $\{1, 2, \ldots, k\} \mapsto \{i^{(1)}, i^{(2)}, \ldots, i^{(k)}\}$ such that $i^{(1)} \ne 1, \ldots, i^{(k)} \ne k$, and can be easily calculated by means of the recurrent relation (see [29])

$$D(k) = kD(k-1) + (-1)^k, \quad D(1) = 0 \,,$$

so that, for example, $D(2) = 1$, $D(3) = 2$, $D(4) = 9$, and so on. Then, according to Proposition 1, the size of a 2-exchange neighborhood is $|\mathcal{N}_2| = d\binom{n}{2}$, the size of a 3-exchange neighborhood is $|\mathcal{N}_3| = d\left[\binom{n}{2} + 2\binom{n}{3}\right]$, etc.

Note also that size of the p-exchange neighborhood is *linear* in the number of dimensions d. Depending on p, $|\mathcal{N}_p|$ is either *polynomial* or *exponential* in the number of elements n per dimension, as follows from the representation

$$D(n) = n! \left(1 - \tfrac{1}{1!} + \tfrac{1}{2!} - \tfrac{1}{3!} + \cdots + \tfrac{(-1)^n}{n!}\right) \approx \frac{n!}{e} \,, \quad n \gg 1 \,.$$

The definition of a local minimum with respect to the p-exchange neighborhood is then straightforward. The kth feasible solution with cost z_k is a p-exchange local minimum iff $z_k \le z_j$ for all $j \in \mathcal{N}_p(k)$. Continuing the example above, the solution $\{111, 222, 333\}$ is a 2-exchange local minimum iff its cost $z_1 = c_{111} + c_{222} + c_{333}$ is less than or equal to costs of all of its 2-exchange neighbors.

The number M_p of local minima of the MAP is obtained by counting the feasible solutions that are local minima with respect to neighborhoods $\mathcal{N}_p$. In a random MAP, where the assignment costs are random variables, M_p becomes a random quantity itself. In this paper we are interested in determining the expected number $\mathsf{E}[M_p]$ of local minima in random MAPs that have iid assignment costs with continuous distribution.

Expected Number of Local Minima in MAP with $n = 2$

As it was noted above, in the special case of random MAP with $n = 2$, $d \ge 3$, the costs of feasible solutions are iid random variables with distribution $F * F$, where F is the distribution of the assignment costs. This special structure of the feasible set allows for a closed-form expression for the expected number of local minima $\mathsf{E}[M]$ (note that in a $n = 2$ MAP the largest local neighborhood is $\mathcal{N}_2$, thus $M = M_2$), as established in [11].

Theorem 2. *In a $n = 2$, $d \ge 3$ MAP with cost coefficients that are iid continuous random variables, the expected number of local minima is given by*

$$\mathsf{E}[M] = \frac{2^{d-1}}{d+1} \,. \tag{17}$$

Equality (17) implies that in a $n = 2$, $d \ge 3$ MAP the number of local minima $\mathsf{E}[M]$ is *exponential* in d, when the cost coefficients are independently drawn from *any* continuous distribution.

Expected Number of Local Minima in a Random MAP with Normally Distributed Costs

Our ability to derive a closed-form expression (17) for the expected number of local minima $\mathsf{E}[M]$ in the previous section has relied on the independence of feasible solution costs (2) in a $n = 2$ MAP. As it is easy to verify directly, in the case $n \geq 3$ the costs of feasible solutions are generally not independent. This complicates analysis significantly if an arbitrary continuous distribution for assignment costs $c_{i_1 \cdots i_d}$ in (1) is assumed. However, as we show below, one can derive upper and lower bounds for $\mathsf{E}[M]$ in the case when the costs coefficients of (1) are independent normally distributed random variables. First, we develop bounds for the number of local minima $\mathsf{E}[M_2]$ defined with respect to 2-exchange neighborhoods $\mathcal{N}_2$ that are most widely used in practice.

2-exchange Local Neighborhoods Noting that in the general case the number N of the feasible solutions to MAP (1) is equal to $N = (n!)^{d-1}$, the expected number of local minima $\mathsf{E}[M_2]$ with respect to local 2-exchange neighborhoods can be written in the form

$$\mathsf{E}[M_2] = \sum_{k=1}^{N} \mathsf{P}\Big[\bigcap_{j \in \mathcal{N}_2(k)} z_k - z_j \leq 0 \Big] , \tag{18}$$

where $\mathcal{N}_2(k)$ is the 2-exchange neighborhood of the kth feasible solution, and z_i is the cost of the ith feasible solution, $i = 1, \ldots, N$. If we allow the n^d cost coefficients $c_{i_1 \cdots i_d}$ of the MAP to be independent standard normal $N(\mu, \sigma^2)$ random variables, then the probability term in (18) can be expressed as

$$\mathsf{P}\Big[\bigcap_{j \in \mathcal{N}_2(k)} z_k - z_j \leq 0 \Big] = F_{\Sigma}(\mathbf{0}) , \tag{19}$$

where F_{Σ} is the c.d.f. of the $|\mathcal{N}_2|$-dimensional random vector

$$\mathbf{Z} = (Z_{121}, \ldots, Z_{12d}, Z_{131}, \ldots, Z_{13d}, \cdots \cdots, Z_{rs1}, \ldots, Z_{rsd}, \cdots, Z_{n-1,n,1}, \ldots, Z_{n-1,n,d}) , \quad r < s . \tag{20}$$

Vector $\mathbf{Z}$ has a normal distribution $N(\mathbf{0}, \Sigma)$ with the covariance matrix Σ defined as

$$\mathrm{Cov}(Z_{rsq}, Z_{ijk}) = \begin{cases} 4\sigma^2, & \text{if} \quad i = r, j = s, q = k, \\ 2\sigma^2, & \text{if} \quad i = r, j = s, q \neq k, \\ \sigma^2, & \text{if} \quad (i = r, j \neq s) \quad \text{or} \quad (i \neq r, j = s) , \\ 0, & \text{if} \quad i \neq r, j \neq s. \end{cases} \tag{21}$$

While the value of $F_{\Sigma}(\mathbf{0})$ in (19) is difficult to compute exactly for large d and n, lower and upper bounds can be constructed using Slepian's inequality [30]. To this end, we introduce covariance matrices $\underline{\Sigma} = (\underline{\sigma}_{ij})$ and $\overline{\Sigma} = (\bar{\sigma}_{ij})$ as

$$\underline{\sigma}_{ij} = \begin{cases} 4\sigma^2, & \text{if} \quad i = j, \\ 2\sigma^2, & \text{if} \quad i \neq j \text{ and} \\ & (i-1) \operatorname{div} d = (j-1) \operatorname{div} d \\ 0, & \text{otherwise} \end{cases} , \tag{22a}$$

$$\bar{\sigma}_{ij} = \begin{cases} 4\sigma^2, & \text{if} \quad i = j, \\ 2\sigma^2, & \text{otherwise} \end{cases} , \tag{22b}$$

so that $\underline{\sigma}_{ij} \leq \sigma_{ij} \leq \overline{\sigma}_{ij}$ holds for all $1 \leq i, j \leq |\mathcal{N}_2|$, with σ_{ij} being the components of the covariance matrix Σ (21). Then, Slepian's inequality claims that

$$F_{\underline{\Sigma}}(\mathbf{0}) \leq F_{\Sigma}(\mathbf{0}) \leq F_{\overline{\Sigma}}(\mathbf{0}) , \tag{23}$$

where $F_{\underline{\Sigma}}(\mathbf{0})$ and $F_{\overline{\Sigma}}(\mathbf{0})$ are c.d.f.'s of random variables $\mathbf{X}_{\underline{\Sigma}} \sim N(\mathbf{0}, \underline{\Sigma})$ and $\mathbf{X}_{\overline{\Sigma}} \sim N(\mathbf{0}, \overline{\Sigma})$ respectively. The structure of matrices $\underline{\Sigma}$ and $\overline{\Sigma}$ allows the corresponding values $F_{\underline{\Sigma}}(\mathbf{0})$ and $F_{\overline{\Sigma}}(\mathbf{0})$ to be computed in a closed form, which leads to the following bounds for the expected number of local minima in random MAP with iid normal coefficients:

Theorem 3. *In a $n \geq 3, d \geq 3$ MAP with iid normal cost coefficients, the expected number of 2-exchange local minima is bounded as*

$$\frac{(n!)^{d-1}}{(d+1)^{n(n-1)/2}} \leq \mathsf{E}[M_2] \leq \frac{2(n!)^{d-1}}{n(n-1)d + 2} . \tag{24}$$

Note that both the lower and upper bounds in (24) coincide with the exact expression (17) for $\mathsf{E}[M_2]$ in the case $n = 2$. Also, from (24) it follows that for fixed $n \geq 3$, the expected number of local minima is *exponential* in the number of dimensions d for a fixed n.

Higher-Order Neighborhoods ($p \geq 3$) The outlined approach is applicable to general p-exchange neighborhoods. For convenience, here we consider the neighborhoods $\mathcal{N}_p^*$ as defined in Sect. "Local Minima and p-exchange Neighborhoods in MAP", i. e., the neighborhoods obtained from a given feasible solution by permuting *exactly* p elements in one of the d dimensions, so that for any feasible solution $i = \{i_1^{(1)} \cdots i_d^{(1)}, \ldots, i_1^{(n)} \cdots i_d^{(n)}\}$ and its p-exchange neighbor $j = \{j_1^{(1)} \cdots j_d^{(1)}, \ldots, j_1^{(n)} \cdots j_d^{(n)}\} \in \mathcal{N}_p^*(i)$ one has (compare to (15))

$$\sum_{r=1}^{n} \bar{\delta}_{i_{k_0}^{(r)} j_{k_0}^{(r)}} = p, k_0 \in \{1, \ldots, d\}, \quad \text{and}$$
$$\sum_{r=1}^{n} \bar{\delta}_{i_k^{(r)} j_k^{(r)}} = 0 \quad \text{for all} \quad k \in \{1, \ldots, d\} \backslash k_0 \, . \tag{25}$$

Then, upper and lower bounds for the expected number of local minima $\mathsf{E}[M_p^*]$ defined with respect to p-exchange neighborhoods $\mathcal{N}_p^*$ can be derived in a similar fashion. Namely, the sought probability

$$\mathsf{P}\Big[\bigcap_{i \in \mathcal{N}_p^*(k)} z_k - z_i \leq 0\Big] = F_{\Sigma_p}(\mathbf{0})$$

can be bounded as $F_{\underline{\Sigma}_p}(\mathbf{0}) \leq F_{\Sigma_p}(\mathbf{0}) \leq F_{\overline{\Sigma}_p}(\mathbf{0})$, where the matrices $\overline{\Sigma}_p, \underline{\Sigma}_p \in \mathbb{R}^{|\mathcal{N}_p^*| \times |\mathcal{N}_p^*|}$ are such that

$$\left(\overline{\Sigma}_p\right)_{ij} = \begin{cases} 2p\sigma^2, & \text{if } i = j, \\ (2p-2)\sigma^2, & \text{if } i \neq j, \end{cases} \tag{26a}$$

$$\left(\underline{\Sigma}_p\right)_{ij} = \begin{cases} 2p\sigma^2, & \text{if } i = j\,, \\ p\sigma^2, & \text{if } i \neq j \text{ and } (i-1) \operatorname{div} \left(dD(p)\right) \\ & \quad = (j-1) \operatorname{div} \left(dD(p)\right), \\ 0, & \text{otherwise}\,. \end{cases} \tag{26b}$$

The corresponding bounds for the expected number of local minima $\mathsf{E}[M_p^*]$ are established by the following theorem [11].

Theorem 4. *In a $n \geq 3, d \geq 3$ MAP with iid normal cost coefficients, the expected number of local minima M_p^* with respect to p-exchange local neighborhoods $\mathcal{N}_p^*$ is bounded as*

$$\frac{n!^{d-1}}{\left[dD(p)+1\right]^{\binom{n}{p}}} \leq \mathsf{E}[M_p^*] \leq n!^{d-1} \int_{-\infty}^{+\infty} \left[\Phi\left(\sqrt{p-1}z\right)\right]^{d\binom{n}{p}D(p)} \mathrm{d}\Phi(z)\,, \tag{27}$$

where $\Phi(z)$ is the c.d.f. of the standard normal $N(0,1)$ distribution. For 3-exchange neighborhoods $\mathcal{N}_3^*$, an improved upper bound holds:

$$\mathsf{E}[M_3^*] \leq \frac{3n!^{d-1}}{n(n-1)(n-2)d+3}\,. \tag{28}$$

It is interesting to note that for a fixed p the ratio of number of local minima to the number of feasible solutions becomes infinitely small as the dimensions of the problem increase (see (17), (24), and (27)).

Conclusions

We have discussed asymptotical analysis of the expected optimal value and the expected number of local minima of the Multidimensional Assignment Problem whose assignment costs are iid random variables drawn from a continuous distribution. It has been demonstrated that for a broad class of distributions, the asymptotical behavior of the expected optimal cost of a random MAP in the case when one of the problem's dimension parameters approaches infinity is determined by the location of the left endpoint of the support set of the distribution. The presented analysis is constructive in the sense that it allows for derivation of lower and upper asymptotical bounds for the expected optimal value of the problem for a prescribed probability distribution.

In addition, we have derived a closed-form expression for the expected number of local minima in a $n = 2$ random MAP with arbitrary distribution of assignment costs. In the case $n \geq 3$, bounds for the expected number of local minima have been derived in the assumption that assignment costs are iid normal random variables. It has been demonstrated that the expected number of local minima is exponential in the number of dimensions d of the problem.

References

1. Aiex RM, Resende MGC, Pardalos PM, Toraldo G (2005) GRASP with Path Relinking for Three-Index Assignment. INFORMS J Comput 17(2):224–247
2. Aldous D (2001) The $\zeta(2)$ limit in the random assignment problem. Random Struct Algorithm 18(4):381–418
3. Alon N, Spencer J (2000) The Probabilistic Method, 2nd edn, Interscience Series in Discrete Mathematics and Optimization. Wiley, New York
4. Andrijich SM, Caccetta L (2001) Solving the multisensor data association problem. Nonlinear Analysis 47:5525–5536
5. Angel E, Zissimopoulos V (2001) On the landscape ruggedness of the quadratic assignment problem. Theor Comput Sci 263:159–172
6. Balas E, Saltzman MJ (1991) An algorithm for the three-index assignment problem. Oper Res 39:150–161
7. Burkard RE (2002) Selected topics on assignment problems. Discret Appl Math 123:257–302
8. Garey MR, Johnson DS (1979) Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, San Francisco
9. Goemans MX, Kodialam M (1993) A lower bound on the expected value of an optimal assignment. Math Oper Res 18:267–274
10. Grundel DA, Oliveira CAS, Pardalos PM (2004) Asymptotic properties of random multidimensional assignment problems. J Optim Theory Appl 122(3):487–500
11. Grundel DA, Krokhmal PA, Oliveira CAS, Pardalos PM (2007) Asymptotic properties of random multidimensional assignment problems. J Comb Optim 13(1):1–18
12. Karp RM (1987) An upper bound on the expected cost of an optimal assignment. In: Discret Algorithm Complexity. Academic Press, Boston, pp 1–4
13. Krokhmal PA, Grundel DA, Pardalos P (2007) Asymptotic Behavior of the Expected Optimal Value of the Multidimensional Assignment Problem. Math Program 109(2–3):525–551
14. Lazarus AJ (1993) Certain expected values in the random assignment problem. Oper Res Lett 14:207–214
15. Lin S, Kernighan BW (1973) An effective heuristic algorithm for the traveling salesman problem. Oper Res 21:498–516
16. Linusson S, Wästlund J (2004) A proof of Parisi's conjecture on the random assignment problem. Probab Theory Relat Fields 128(3):419–440
17. Mézard M, Parisi G (1985) Replicas and optimization. J Phys Lett 46(17):771–778
18. Murphey R, Pardalos P, Pitsoulis L (1998) A greedy randomized adaptive search procedure for the multitarget multisensor tracking problem. In: DIMACS Series, vol 40, American Mathematical Society, pp 277–302
19. Nair C, Prabhakar B, Sharma M (2005) A Proof of the Conjecture due to Parisi for the Finite Random Assignment Problem. Random Struct Algorithms 27(4):413–444
20. Olin B (1992) Asymptotic properties of the random assignment problem. Ph.D thesis, Royal Institute of Technology, Stockholm, Sweden
21. Palmer R (1991) Optimization on rugged landscapes. In: Perelson A, Kauffman S (eds) Molecular Evolution on Rugged Ladscapes: Proteins, RNA, and the Immune System. Addison Wesley, Redwood City, pp 3–25
22. Pardalos PM, Ramakrishnan KG (1993) On the expected optimal value of random assignment problems: Experimental results and open questions. Comput Optim Appl 2:261–271
23. Parisi G (1998) A conjecture on random bipartite matching. Physics e-Print archive, http://xxx.lang.gov/ps/cond-_mat/9801176
24. Papadimitrou CH, Steiglitz K (1998) Combinatorial Optimization: Algorithms and Complexity. Dover, New York
25. Pasiliao EL (2003) Algorithms for Multidimensional Assignment Problems. PhD thesis, University of Florida
26. Pierskalla W (1968) The multidimensional assignment problem. Oper Res 16:422–431
27. Poore AB (1994) Multidimensional assignment formulation of data association problems arising from multitarget and multisensor tracking. Comput Optim Appl 3:27–54
28. Pusztaszeri J, Rensing PE, Liebling TM (1995) Tracking elementary particles near their primary vertex: a combinatorial approach. J Glob Optim 16:422–431
29. Stanley R (1986) Enumerative Combinatorics. Wadsworth and Brooks, Belmont CA
30. Tong YL (1990) The Multivariate Normal Distribution. Springer, Berlin
31. Veenman CJ, Hendriks EA, Reinders MJT (1998) A fast and robust point tracking algorithm. Proc Fifth IEEE Int Conf Image Processing 653–657, Chicago, USA
32. Walkup DW (1979) On the expected value of a random assignment problem. SIAM J Comput 8:440–442

Asynchronous Distributed Optimization Algorithms

Ioannis P. Androulakis
Department of Biomedical Engineering,
Rutgers University, Piscataway, USA

MSC2000: 90C30, 90C30, 90C52, 90C53, 90C55

Article Outline

Keywords

Asynchronous iterative algorithms; Distributed computing; Optimization

Many iterative algorithms, deterministic or stochastic, admit distributed implementations, whereby the work load for performing computational steps, identified as bottlenecks, is distributed among a variety of computational nodes. Extensive literature regarding distributed implementations of optimization algorithms in particular is available, [19]. In recent years, there has been an extremely fruitful interface between mathematical programming algorithms and computer science. This has resulted in major advances in the development of algorithms and implementation of sophisticated optimization algorithms on high performance parallel and distributed computers, [11,12]. Two major issues are important in designing an efficient distributed implementation, namely, *task allocation*, and *communication protocol*. Task allocation relates to the breakdown of the total work load and this can either be static or dynamic depending. Communication patterns and frequency are important since they can induce substantial overhead in cases where workload irregularities occur. Various important implementational details have been presented, among others, in [10]. The straightforward translation of serial to a distributed algorithm would assume some sort of global synchronization mechanism that would guarantee that information among processing nodes is being exchanged once a computational step has been performed. Processors must then synchronize so as to exchange information and proceed all with the same type of information to their next computational step. *Asynchronous algorithms* relax the assumption of a predetermined synchronization protocol, and allow each processing element to compute and communicate following local rates. The primary motivation for developing algorithms was to address situations in which:

- processors do not need to communicate to each other processor at each time instance;
- processors may keep performing computations without having to wait until they receive the messages that have been transmitted to them;
- processors are allowed to remain idle some of the time;
- some processors may be performing computations faster than others.

Such algorithms can alleviate communication overloads and they are not excessively slowed down by either communication delays nor by differences in the time it takes processors to perform one computation, [18]. Another major motivation is clearly to develop robust algorithms for distributed computation on heterogeneous networks of computers. The ideas of asynchronous, also known as *chaotic*, iterative schemes, can be traced by to [9], in which special schemes for solving linear systems of equations were developed. For discussing the basic principles and conditions of asynchronous iterations, the formalism of [8] will be followed. This work presented the first comprehensive treatment of the recent developments in the theory and practice of asynchronous iterations for a variety of problems, including deterministic and stochastic optimization. In essence, most iterative algorithms can be viewed as the search for a fixed point that corresponds to the solution of the original problem. The basic assumptions of the model of asynchronous (chaotic) iterations for determining fixed point of (non)linear mappings are as follows:

1) Let X be a vector space and $x = (x_1, \ldots, x_n) \in X$ are n-tuples describing any vector from this set. It is also assumed that $X = X_1 \times \cdots \times X_n$, with $x_i \in X_i$, $i = 1, \ldots, n$.
2) Let $f: X \to X$ be a function defined by $f(x) = (f_1(x), \ldots, f_n(x))$, $\forall x \in X$.
3) A point $X^\star \in X$ is a *fixed point* of $f(x)$ if $x^\star = f(x^\star)$ or, equivalently, $x_i^\star = f_i(x^\star)$, $i = 1, \ldots, n$.

For the solution of the aforementioned problem, one can define an iterative method as:

$$x_i := f_i(x), \quad i = 1, \ldots, n,$$

with $x_i(t)$ being the values of the ith component at time (iteration) t. In order to comprehend the concept of asynchronous iterations, we assume that there exists a set of times $T = \{0, 1, \ldots\}$ at which one or more (possibly none) components x_i of x are updated by some processor of a distributed computing system. We defined by T^i the set of times at which x_i is updated. Given that no synchronization protocol dictating the information exchange exists, it is quite conceivable that not all processors have access to the same and most recent values of of the corresponding components of x. It will be

therefore assumed that:

$$x_i(t+1) = \begin{cases} f_i(x_1\tau_1^i(t), \ldots, x_n(\tau_n^i(t)), \\ \quad \forall t \in T^i,\ 0 \le \tau_j^i(t) \le t, \\ x_i(t), \quad \forall t \notin T^i. \end{cases}$$

In the aforementioned definition of the iterative process, the difference $t - \tau_j^i(t)$ between the current time t and the time $\tau_j^i(t)$ corresponding to the jth component available at the processor updating $x_i(t)$ can be viewed as some form of communication delay. In studying the convergence behavior of algorithms of this type, two cases have to be considered. The operation can either be *totally asynchronous* or *partially asynchronous*. The concept of totally asynchronous algorithms was first introduced in [9], and subsequently analyzed in, among other, [1,5,15]. [5] proposed a general framework that ensembles a variety of instances. The cornerstone of his approach is based on the *asynchronous convergence theorem*, [8]. It defined a general pattern for proving convergence of the asynchronous counterparts of certain sequential algorithms. The asynchronous convergence theorem can be applied to variety of problems including:

- problems involving maximum norm contraction mappings;
- problems involving monotone mappings;
- the shortest path problem;
- linear and nonlinear network flow problems.

Qualitatively speaking, the fundamental difference between a synchronous and an asynchronous iterative mapping, is similar to the differences between a Jacobi and a Gauss–Seidel iteration. Consider the implementation of both these approaches in the minimization of function $F(x)$. The specifics of the minimization algorithm are irrelevant:

- Jacobi:

$$x_i(t+1) = \arg\min_{x_i} F(x_1(t), \ldots, x_n(t));$$

- Gauss–Seidel:

$$x_i(t+1) = \arg\min_{x_i} F(x_1(t+1), \ldots, x_i(t), \ldots, x_n(t)).$$

The Gauss–Seidel approach corresponds to the instantaneous communication, in a sequential manner, of the information as it being generated. The Jacobi iteration, forces processors to perform iterations utilizing 'outdated' information. The asynchronous iteration is reminiscent to a Jacobi one. A thorough analysis and comparison of these two extremes is presented in [16]. A major class of iterative schemes that can be shown to be convergent when implemented asynchronously, are defined by mappings which can be shown to be *contraction mappings* with respect to a suitably defined *weighted maximum norm*:

$$\|x\|_\infty^\omega = \max_i \frac{|x_i|}{\omega_i},$$
$$x \in \mathbb{R}^n, \quad \omega \in \mathbb{R}_+^n.$$

Let us consider the minimization of an unconstrained quadratic function F:

$$\begin{cases} \min & F(x) = \frac{1}{2}x^\top Ax - b^\top x \\ \text{s.t.} & x \in \mathbb{R}^n, \end{cases}$$

where A is an $n \times n$ positive definite symmetric matrix, and $b \in \mathbf{R}^n$. A gradient iteration of the form

$$x := (I - \gamma A)x + \gamma b$$

will be convergent provided that the maximum row sum of $I - \gamma\, A$ is less than 1, i. e.:

$$\left|1 - \gamma\alpha_{ij}\right| + \sum_{j:j\neq i} \gamma \left|a_{ij}\right| < 1, \quad i = 1, \ldots, n,$$

implying the *diagonal dominance condition*:

$$a_{ij} > \sum_{j:j\neq i} \left|a_{ij}\right|, \quad \forall i.$$

If we consider the general nonlinear unconstrained optimization problem:

$$\begin{cases} \min & g(x) \\ \text{s.t.} & x \in \mathbb{R}^n, \end{cases}$$

where g: $\mathbf{R}^n \to \mathbf{R}$ is a twice-differentiable convex function, with Hessian matrix $\nabla^2 g(x)$ which is positive definite. If one considers a Newton mapping given by:

$$f(x) = x - [\nabla^2 g(x)]^{-1} \nabla g(x)$$

The norm $\| \mathrm{x} \| = \max_i |x_i|$ makes f a contraction mapping in the neighborhood of $x^\star$ (the optimal point). Extensions of the ordinary gradient method

$$f(x) = x - \alpha \nabla g(x)$$

are also discussed in [5]. The shortest path problem is defined in terms of a directed graph consisting of n nodes. We denote by $A(i)$ the set of all nodes j for which there is an outgoing arc (i, j) from node i. The problem is to find a path of minimum length starting at node i and ending at node j. [4] considered the application of the asynchronous convergence theorem to fixed point iterations involving monotone mappings by considering the Bellman–Ford algorithm, [3], applied to the *shortest path problem*. This takes the form:

$$x_i(t+1) = \min_{j \in A(i)} (a_{ij} + x_j(\tau_j^i(t)),$$
$$i = 2, \ldots, n, \quad t \in T^i,$$
$$x_1(t+1) = 0.$$

$A(i)$ is the set of all nodes j for which there exists an arc (i, j). *Linear network flow problems* are discussed in [8] and asynchronous distributed versions of the auction algorithm are discussed. In the general linear network flow problem we are given a set of N nodes and a set of arcs A, each arc (i, j) has associated with it an integer a_{ij}, referred to as the cot coefficient. The problem is to optimally assign flows, f_{ij} to each one of the arcs, and the problem is represented mathematically as follows:

$$\begin{cases} \min & \displaystyle\sum_{(i,j)\in A} a_{ij} f_{ij} \\ \text{s.t.} & \displaystyle\sum_{j:(i,j)\in A} f_{ij} - \sum_{j:(j,i)} f_{ji} = s_i, \ \forall i \in N, \\ & b_{ij} \le f_{ij} \le c_{ij}, \quad \forall (i,j) \in A, \end{cases}$$

where a_{ij}, b_{ij}, c_{ij} and s_i are integers. Extensions of the sequential *auction algorithms* are discussed in [6], in which asynchronism manifests itself in the sense that certain processors may be calculating actions bids which other update object prices. [7] extended the analysis to cover certain classes of *nonlinear network flow problems* in which the costs a_{ij} are functions of the flows f_{ij}:

$$\begin{cases} \min & \displaystyle\sum_{(i,j)\in A} a_{ij}(f_{ij}) \\ \text{s.t.} & \displaystyle\sum_{j:(i,j)\in A} f_{ij} - \sum_{j:(j,i)} f_{ji} = s_i, \ \forall i \in N, \\ & b_{ij} \le f_{ij} \le c_{ij}, \quad \forall (i,j) \in A. \end{cases}$$

Imposing additional reasonable assumptions to the general framework of totally asynchronous iterative algorithms can substantially increase the applicability of the concept. A natural extension is therefore the *partially asynchronous iterative methods*, whereby two major assumptions are be satisfied:

a) each processor performs an update at least once during any time interval of length B;
b) the information used by any processor is outdated by at most B time units.

In other words, the partial asynchronism assumption extends the original model of computation by stating that:

There exists a positive integer B such that:

- For every i and for every $t \ge 0$, at least one of the elements of the set $\{t, \ldots, t + B - 1\}$ belongs to T^i.
- There holds:

$$t - B \le \tau_j^i(t) \le t,$$

for all i and j, and all $t \ge 0$ belonging to T^i.
- There holds $\tau_i^i(t) = t$ for all i and $t \in T^i$.

[17] developed a very elegant framework with important implications on the asynchronous minimization of continuous functions. It was established that, while minimize function $F(x)$, the asynchronous implementation of a gradient-based algorithm:

$$x := x - \gamma \nabla F(X)$$

is convergent if and only if the stepsize γ is small compared to the inverse of the asynchronism measure B. Specifically, let F: $\mathbf{R}^n \to \mathbf{R}$ be a cost function to be minimized subject to no constraints. It will be further assumed that:

1) $F(x) > 0$, $\forall x \in \mathbf{R}^n$;
2) $F(x)$ is *Lipschitz continuous*:

$$\|\nabla F(x) - \nabla F(y)\| \le K_1 \|x - y\|,$$
$$\forall x, y, \in \mathbb{R}^n.$$

The asynchronous gradient algorithm of the synchronous iteration:

$$x := x - \gamma \nabla F(x)$$

is denoted by:

$$x_i(t+1) := x_i(t) - \gamma s_i(t), \quad i = 1, \ldots, n,$$

where γ is a positive stepsize, and $s_i(t)$ is the update direction. It will be assumed that

$$s_i(t) = 0, \quad \forall t \notin T^i.$$

It is important to realize that processor i at time time t has knowledge of a vector $x^i(t)$ that is a, possibly, outdated version of $x(t)$. In other words: $x^i(t) = ((x_1(\tau_1^i(t)), \ldots, x_n(\tau_n^i(t)))$. It is further assumed that when x_i is being updated, the update direction s_i is a *descent direction*: For every i and t:

$$s_i(t)\nabla_i F(x^i(t)) \leq 0$$

there exists positive constants K_2, K_3 such that

$$K_1 \left|\nabla_i F(x^i(t))\right| \leq |s_i(t)| \leq K_3 \left|\nabla_i F(x^i(t))\right|, \quad \forall t \in T^i, \quad \forall i.$$

If all of the above is satisfied, then for the asynchronous gradient iteration it can be shown that: There exists some γ_0, depending on n, B, K_1, K_3, such that if $0 < \gamma < \gamma_0$, then $\lim_{t\to\infty} \lambda\, F(x(t)) = 0$.

It can actually be further shown that the choice

$$\gamma = \frac{1}{K_3 K_1 (1 + B + nB)}$$

can guarantee convergence of the asynchronous algorithm. This results clearly states that one can always, in principle, identify an adequate stepsize for any finite delay.

Furthermore, [14] elaborated on the use of gradient projection algorithm, within the asynchronous iterative framework, for addressing certain classes of constraint nonlinear optimization problems. The constrained optimization problems considered, is that of minimizing a convex function F: $\mathbf{R}^n \to \mathbf{R}$, defined over the space $X = \prod_{i=1}^n X_i$ of lower-dimensional sets $X_i \subset \mathbf{R}^{n_i}$, and $\sum_{i=1}^m n_i = n$. The ith component of the solution vector is now updated by

$$x_i(t+1) = [x_i(t) - \gamma \nabla_i F(x^i(t))]^+$$

where $[\cdot]^+$ denotes the projection on the set X_i. Once again: $x_i(t+1) = x_i(t)$, $t \notin T^i$. Once again, a gradient based algorithm is defined, for which

$$s_i(t) = \begin{cases} \frac{1}{\gamma}\left([x_i(t) - \gamma\nabla_i F(x^i(t))]^+ - x_i(t)\right), \\ \quad t \in T^i, \\ 0 \quad t \notin T^i. \end{cases}$$

It can actually be shown that for, provided that the partial asynchronism assumption holds, one can always define, in principle, a suitable stepsize γ_0 such that for any $0 < \gamma < 0$ the limit point, x^*, of the sequence generated by the partially asynchronous gradient projection iteration minimizes the Lipschitz continuous, convex function F over the set X. Recently, [2], analyzed asynchronous algorithms for minimizing a function when the communication delays among processors are assumed to be stochastic with Markovian character. The approach is also based on a gradient projection algorithm and was used to address a an optimal routing problem.

A major consideration in asynchronous distributed computing is the fact that since no globally controlling mechanism exists makes the use of any termination criterion which is based on local information obsolete. Clearly, when executing asynchronously a distributed iteration of the form $x_i - f_i(x)$ local error estimates can, and will be, misleading in terms of the global state of the system. Recently [13] made several suggestions as to how the standard model can be supplemented with an additional interprocessor communication protocol so as to address the issue of finite termination of asynchronous iterative algorithms.

See also

- ▶ Automatic Differentiation: Parallel Computation
- ▶ Heuristic Search
- ▶ Interval Analysis: Parallel Methods for Global Optimization
- ▶ Load Balancing for Parallel Optimization Techniques
- ▶ Parallel Computing: Complexity Classes
- ▶ Parallel Computing: Models
- ▶ Parallel Heuristic Search
- ▶ Stochastic Network Problems: Massively Parallel Solution

References

1. Baudet GM (1978) Asynchronous iterative methods for multiprocessors. J ACM 25:226–244
2. Beidas BF, Papavassilopoulos GP (1995) Distributed asynchronous algorithms with stochastic delays for constrained optimization problems with conditions of time drift. Parallel Comput 21:1431–1450

3. Bellman R (1957) Dynamic programming. Princeton University Press, Princeton
4. Bertsekas DP (1982) Distributed dynamic programming. IEEE Trans Autom Control AC-27:610–616
5. Bertsekas DP (1983) Distributed asynchronous computation of fixed points. Math Program 27:107–120
6. Bertsekas DP, Eckstein J (1987) Distributed asynchronous relaxation methodfs for linear network flow problems. Proc IFAC:39–56
7. Bertsekas DP, El Baz D (1987) Distributed asynchronous relaxation methods for convex network flow problems. SIAM J Control Optim 25:74–85
8. Bertsekas DP, Tsitsiklis JN (1989) Parallel and distributed computation: Numerical methods. Prentice-Hall, Englewood Cliffs, NJ
9. Chazan D, Miranker W (1968) Chaotic reaxation. Linear Alg Appl 2:199–222
10. Ferreira A, Pardalos PM (eds) (1997) Solving combinatorial optimization problems in parallel. Springer, Berlin
11. Pardalos PM, Phillips AT, Rosen JB (eds) (1992) Topics in parallel computing in mathematical programming. Sci Press, Marrickville, Australia
12. Pardalos PM (ed) (1992) Advances in optimization and parallel computing. North-Holland, Amsterdam
13. Savari SA, Bertsekas DP (1996) Finite termination of asynchronous iterative algorithms. Parallel Comput 22:39–56
14. Tseng P (1991) On the rate of convergence of a partially asynchronous gradient projection algorithm. SIAM J Optim 1:603–619
15. Tsitsiklis JN (1987) On the stability of asynchronous iterative processes. Math Syst Theory 20:137–153
16. Tsitsiklis JN (1989) A comparison of Jacobi and Gauss–Seidel parallel iterations. Appl Math Lett 2:167–170
17. Tsitsiklis JN, Bertsekas DP, Athans M (1986) Distributed asynchronous deterministic and stochastic gradient optimization algorithms. IEEE Trans Autom Control ac-31:803–813
18. Tsitsiklis JN, Bertsekas DP, Athnas M (1986) Distributed asynchronous deterministic and stochastic gradient optimization algorithms. IEEE Trans Autom Control AC-31:803–812
19. Zenios AS (1994) Parallel numerical optimization: Current status and annotated bibliography. ORSA J Comput 1: 20–42

Auction Algorithms

DIMITRI P. BERTSEKAS
Labor. Information and Decision Systems,
Massachusetts Institute Technol., Cambridge, USA

MSC2000: 90C30, 90C35

Article Outline

Keywords

Linear programming; Optimization; Assignment problem; Transshipment problem

The auction algorithm is an intuitive method for solving the classical assignment problem. It outperforms substantially its main competitors for important types of problems, both in theory and in practice, and is also naturally well suited for parallel computation. In this article, we will sketch the basic principles of the algorithm, we will explain its computational properties, and we will discuss its extensions to more general network flow problems. For a detailed presentation, see the survey paper [3] and the textbooks [2,4]. For an extensive computational study, see [8]. The algorithm was first proposed in the 1979 report [1].

In the classical *assignment problem* there are n persons and n objects that we have to match on a one-to-one basis. There is a benefit a_{ij} for matching person i with object j and we want to assign persons to objects so as to maximize the total benefit. Mathematically, we want to find a one-to-one assignment [a set of person-object pairs $(1, j_1), \ldots, (n, j_n)$, such that the objects $j_1, \ldots, j_n$ are all distinct] that maximizes the total benefit $\sum_{i=1}^{n} a_{ij_i}$.

The assignment problem is important in many practical contexts. The most obvious ones are resource allocation problems, such as assigning personnel to jobs, machines to tasks, and the like. There are also situations where the assignment problem appears as a subproblem in various methods for solving more complex problems.

The assignment problem is also of great theoretical importance because, despite its simplicity, it embodies a fundamental linear programming structure. The most important type of linear programming prob-

lems, the linear network flow problem, can be reduced to the assignment problem by means of a simple reformulation. Thus, any method for solving the assignment problem can be generalized to solve the linear network flow problem, and in fact this approach is particularly helpful in understanding the extension of auction algorithms to network flow problems that are more general than assignment.

The classical methods for assignment are based on iterative improvement of some cost function; for example a primal cost (as in primal simplex methods), or a dual cost (as in Hungarian-like methods, dual simplex methods, and relaxation methods). The auction algorithm departs significantly from the cost improvement idea; at any one iteration, it may deteriorate both the primal and the dual cost, although in the end it finds an optimal assignment. It is based on a notion of approximate optimality, called ϵ-*complementary slackness*, and while it implicitly tries to solve a dual problem, it actually attains a dual solution that is not quite optimal.

The Auction Process

To develop an intuitive understanding of the auction algorithm, it is helpful to introduce an economic equilibrium problem that turns out to be equivalent to the assignment problem. Let us consider the possibility of matching the n objects with the n persons through a market mechanism, viewing each person as an economic agent acting in his own best interest. Suppose that object j has a price p_j and that the person who receives the object must pay the price p_j. Then, the (net) value of object j for person i is $a_{ij} - p_j$ and each person i would logically want to be assigned to an object j_i with maximal value, that is, with

$$a_{ij_i} - p_{j_i} = \max_{j=1,\ldots,n} \{a_{ij} - p_j\}. \tag{1}$$

We will say that a person i is 'happy' if this condition holds and we will say that an assignment and a set of prices are at *equilibrium* when all persons are happy.

Equilibrium assignments and prices are naturally of great interest to economists, but there is also a fundamental relation with the assignment problem; it turns out that an equilibrium assignment offers maximum total benefit (and thus solves the assignment problem), while the corresponding set of prices solves an associated dual optimization problem. This is a consequence of the celebrated duality theorem of linear programming.

Let us consider now a natural process for finding an equilibrium assignment. I will call this process the *naive auction algorithm*, because it has a serious flaw, as will be seen shortly. Nonetheless, this flaw will help motivate a more sophisticated and correct algorithm.

The naive auction algorithm proceeds in 'rounds' (or 'iterations') starting with *any* assignment and *any* set of prices. There is an assignment and a set of prices at the beginning of each round, and if all persons are happy with these, the process terminates. Otherwise some person who is not happy is selected. This person, call him i, finds an object j_i which offers maximal value, that is,

$$j_i \in \arg \max_{j=1,\ldots,n} \{a_{ij} - p_j\}, \tag{2}$$

and then:

a) Exchanges objects with the person assigned to j_i at the beginning of the round;
b) Sets the price of the best object j_i to the level at which he is indifferent between j_i and the second best object, that is, he sets p_{j_i} to

$$p_{j_i} + \gamma_i, \tag{3}$$

where

$$\gamma_i = v_i - w_i, \tag{4}$$

v_i is the best object value,

$$v_i = \max_j \{a_{ij} - p_j\}, \tag{5}$$

and w_i is the second best object value

$$w_i = \max_{j \neq j_i} \{a_{ij} - p_j\}, \tag{6}$$

that is, the best value over objects other than j_i. (Note that γ_i is the largest increment by which the best object price p_{j_i} can be increased, with j_i still being the best object for person i.)

This process is repeated in a sequence of rounds until all persons are happy.

We may view this process as an *auction*, where at each round the bidder i raises the price of his or her preferred object by the *bidding increment* γ_i. Note that γ_i

cannot be negative since $v_i \geq w_i$ (compare (5) and (6)), so the object prices tend to increase. Just as in a real auction, bidding increments and price increases spur competition by making the bidder's own preferred object less attractive to other potential bidders.

Does this auction process work? Unfortunately, not always. The difficulty is that the bidding increment γ_i is zero when more than one object offers maximum value for the bidder i (cf. (4) and (6)). As a result, a situation may be created where several persons contest a smaller number of equally desirable objects without raising their prices, thereby creating a never ending cycle.

To break such cycles, we introduce a perturbation mechanism, motivated by real auctions where each bid for an object must raise its price by a minimum positive increment, and bidders must on occasion take risks to win their preferred objects. In particular, let us fix a positive scalar ϵ and say that a person i is 'almost happy' with an assignment and a set of prices if the value of its assigned object j_i is within ϵ of being maximal, that is,

$$a_{ij_i} - p_{j_i} \geq \max_{j=1,\ldots,n} \{a_{ij} - p_j\} - \epsilon. \tag{7}$$

We will say that an assignment and a set of prices are *almost at equilibrium* when all persons are almost happy. The condition (7), introduced first in 1979 in conjunction with the auction algorithm, is known as ϵ-*complementary slackness* and plays a central role in several optimization contexts. For $\epsilon = 0$ it reduces to ordinary complementary slackness (compare (1)).

We now reformulate the previous auction process so that the bidding increment is always at least equal to ϵ. The resulting method, the *auction algorithm*, is the same as the naive auction algorithm, except that the bidding increment γ_i is

$$\gamma_i = v_i - w_i + \epsilon, \tag{8}$$

(rather than $\gamma_i = v_i - w_i$ as in (4)). With this choice, the bidder of a round is almost happy at the end of the round (rather than happy). The particular increment $\gamma_i = v_i - w_i + \epsilon$ used in the auction algorithm is the maximum amount with this property. Smaller increments γ_i would also work as long as $\gamma_i \geq \epsilon$, but using the largest possible increment accelerates the algorithm. This is consistent with experience from real auctions, which tend to terminate faster when the bidding is aggressive.

We can now show that this reformulated auction process terminates in a finite number of rounds, necessarily with an assignment and a set of prices that are almost at equilibrium. To see this, note that once an object receives a bid for the first time, then the person assigned to the object at every subsequent round is almost happy; the reason is that a person is almost happy just after acquiring an object through a bid, and continues to be almost happy as long as he holds the object (since the other object prices cannot decrease in the course of the algorithm). Therefore, the persons that are not almost happy must be assigned to objects that have never received a bid. In particular, once each object receives at least one bid, the algorithm must terminate. Next note that if an object receives a bid in m rounds, its price must exceed its initial price by at least $m\epsilon$. Thus, for sufficiently large m, the object will become 'expensive' enough to be judged 'inferior' to some object that has not received a bid so far. It follows that only for a limited number of rounds can an object receive a bid while some other object still has not yet received any bid. Therefore, there are two possibilities: either

a) the auction terminates in a finite number of rounds, with all persons almost happy, before every object receives a bid; or
b) the auction continues until, after a finite number of rounds, all objects receive at least one bid, at which time the auction terminates. (This argument assumes that any person can bid for any object, but it can be generalized for the case where the set of feasible person-object pairs is limited, as long as at least one feasible assignment exists.)

Optimality Properties at Termination

When the auction algorithm terminates, we have an assignment that is almost at equilibrium, but does this assignment maximize the total benefit? The answer here depends strongly on the size of ϵ. In a real auction, a prudent bidder would not place an excessively high bid for fear that he might win the object at an unnecessarily high price. Consistent with this intuition, we can show that if ϵ is small, then the final assignment will be 'almost optimal'. In particular, we can show that the total benefit of the final assignment is within $n\epsilon$ of being

optimal. To see this, note that an assignment and a set of prices that are almost at equilibrium may be viewed as being at equilibrium for a *slightly different* problem where all benefits a_{ij} are the same as before, except for the n benefits of the assigned pairs which are modified by an amount no more than ϵ.

Suppose now that the benefits a_{ij} are all integer, which is the typical practical case (if a_{ij} are rational numbers, they can be scaled up to integer by multiplication with a suitable common number). Then, the total benefit of any assignment is integer, so if $n\epsilon < 1$, a complete assignment that is within $n\epsilon$ of being optimal must be optimal. It follows, that if

$$\epsilon < \frac{1}{n},$$

and the benefits a_{ij} are all integer, then the assignment obtained upon termination of the auction algorithm is optimal. Let us also note that the final set of prices is within $n\epsilon$ of being an optimal solution of the dual problem

$$\min_{\substack{p_j \\ j=1,\ldots,n}} \left\{ \sum_{j=1}^{n} p_j + \sum_{i=1}^{n} \max_j \{a_{ij} - p_j\} \right\}. \tag{9}$$

This leads to the interpretation of the auction algorithm as a dual algorithm (in fact an approximate coordinate ascent algorithm; see the cited literature).

Computational Aspects: ϵ-Scaling

The auction algorithm exhibits interesting computational behavior, and it is essential to understand this behavior to implement the algorithm efficiently. First note that the amount of work to solve the problem can depend strongly on the value of ϵ and on the maximum absolute object value

$$C = \max_{i,j} \left| a_{ij} \right|.$$

Basically, for many types of problems, the number of bidding rounds up to termination tends to be proportional to C/ϵ. Note also that there is a dependence on the initial prices; if these prices are 'near optimal,' we expect that the number of rounds to solve the problem will be relatively small.

The preceding observations suggest the idea of ϵ-*scaling*, which consists of applying the algorithm several times, starting with a large value of ϵ and successively reducing ϵ up to an ultimate value that is less than some critical value (for example, $1/n$, when the benefits a_{ij} are integer). Each application of the algorithm provides good initial prices for the next application. This is a very common idea in nonlinear programming, encountered for example, in barrier and penalty function methods. An alternative form of scaling, called *cost scaling*, is based on successively representing the benefits a_{ij} with an increasing number of bits, while keeping ϵ at a constant value.

In practice, it is a good idea to at least consider scaling. For sparse assignment problems, that is, problems where the set of feasible assignment pairs is severely restricted, scaling seems almost universally helpful. In theory, scaling leads to auction algorithms with a particularly favorable polynomial complexity (without scaling, the algorithm is pseudopolynomial; see the cited literature).

Parallel and Asynchronous Implementation

Both the bidding and the assignment phases of the auction algorithm are highly parallelizable. In particular, the bidding and the assignment can be carried out for all persons and objects simultaneously. Such an implementation can be termed *synchronous*. There are also *totally asynchronous* implementations of the auction algorithm, which are interesting because they are quite flexible and also tend to result in faster solution in some types of parallel machines. To understand these implementations, it is useful to think of a person as an autonomous decision maker who at unpredictable times obtains information about the prices of the objects. Each person who is not almost happy makes a bid at arbitrary times on the basis of its current object price information (that may be outdated because of communication delays).

See [7] for a careful formulation of the totally asynchronous model, and a proof of its validity, including extensive computational results on a shared memory machine, confirming the advantage of asynchronous over synchronous implementations.

Variations and Extensions

The auction algorithm can be extended to solve a number of variations of the assignment problem, such as the

asymmetric assignment problem where the number of objects is larger than the number of persons and there is a requirement that all persons be assigned to some object. Naturally, the notion of an assignment must now be modified appropriately. To solve this problem, the auction algorithm need only be modified in the choice of initial conditions. It is sufficient to require that all initial prices be zero. A similar algorithm can be used for the case where there is no requirement that all persons be assigned. Other variations handle efficiently the cases where there are several groups of 'identical' persons or objects ([5]).

There have been extensions of the auction algorithm for other types of linear network optimization problems. The general approach for constructing auction algorithms for such problems is to convert them to assignment problems, and then to suitably apply the auction algorithm and streamline the computations. In particular, the classical shortest path problem can be solved correctly by the naive auction algorithm described earlier, once the method is streamlined. Similarly, auction algorithms can be constructed for the max-flow problems, and are very efficient. These algorithms bear a close relation to preflow-push algorithms for the max-flow problem, which were developed independently of auction ideas.

The auction algorithm has been extended to solve linear transportation problems ([5]). The basic idea is to convert the transportation problem into an assignment problem by creating multiple copies of persons (or objects) for each source (or sink respectively), and then to modify the auction algorithm to take advantage of the presence of the multiple copies.

There are extensions of the auction algorithm for linear minimum cost flow (*transshipment*) problems, such as the so called ϵ-relaxation method, and the auction/sequential shortest path algorithm algorithm (see the cited literature for a detailed description). These methods have interesting theoretical properties and like the auction algorithm, are well suited for parallelization (see the survey [6], and the textbook [7]).

Let us finally note that there have been proposals of auction algorithms for convex separable network optimization problems with and without gains (but with a single commodity and without side constraints); see [9].

See also

- ▶ Communication Network Assignment Problem
- ▶ Directed Tree Networks
- ▶ Dynamic Traffic Networks
- ▶ Equilibrium Networks
- ▶ Evacuation Networks
- ▶ Generalized Networks
- ▶ Maximum Flow Problem
- ▶ Minimum Cost Flow Problem
- ▶ Network Design Problems
- ▶ Network Location: Covering Problems
- ▶ Nonconvex Network Flow Problems
- ▶ Piecewise Linear Network Flow Problems
- ▶ Shortest Path Tree Algorithms
- ▶ Steiner Tree Problems
- ▶ Stochastic Network Problems: Massively Parallel Solution
- ▶ Survivable Networks
- ▶ Traffic Network Equilibrium

References

1. Bertsekas DP (1979) A distributed algorithm for the assignment problem. Working Paper MIT Lab Information & Decision Systems
2. Bertsekas DP (1991) Linear network optimization: Algorithms and codes. MIT, Cambridge, MA
3. Bertsekas DP (1992) Auction algorithms for network flow problems: A tutorial introduction. Comput Optim Appl 1: 7–66
4. Bertsekas DP (1998) Network optimization: Continuous and discrete problems. Athena Sci., Belmont, MA
5. Bertsekas DP, Castañon DA (1989) The auction algorithm for transportation problems. Ann Oper Res 20:67–96
6. Bertsekas DP, Castañon DA, Eckstein J, Zenios S (1995) Parallel computing in network optimization. In: Ball MO, Magnanti TL, Monma CL, Nemhauser GL (eds) Handbooks in OR and MS, vol 7, North-Holland, Amsterdam, pp 331–399
7. Bertsekas DP, Tsitsiklis JN (1989) Parallel and distributed computation: Numerical methods. Prentice-Hall, Englewood Cliffs, NJ
8. Castañon DA (1993) Reverse Auction Algorithms for Assignment Problems. In: Johnson DS, McGeoch CC (eds) Algorithms for network flows and matching. Amer Math Soc, Providence, pp 407–429
9. Tseng P, Bertsekas DP (1996) An epsilon-relaxation method for separable convex cost generalized network flow problems. Math Program (to appear), MIT Lab Information & Decision Systems P-2374

Automatic Differentiation: Calculation of the Hessian

LAURENCE DIXON
Numerical Optim. Centre, University Hertfordshire, Hatfield, England

MSC2000: 90C30, 65K05

Article Outline

Keywords

Automatic differentiation; Gradient; Hessian; Doublet; Triplet

The *Hessian* of a scalar function $f(x)$ can be computed automatically in at least two ways. The first is a natural extension of the forward method for calculating gradients. The others extend the reverse method.

The Forward Mode

The concept of forward *automatic differentiation* was described by L.B. Rall [14]. When calculating the gradient vector of a function of n variables, a *doublet* data structure is introduced, consisting of $n + 1$ floating point numbers. To calculate the Hessian matrix, this data structure is extended to a *triplet*.

A triplet is a data structure that, in the simplest form, contains $1 + n + n(n+1)/2$ floating point numbers. If X is a variable that occurs in the evaluation of $f(x)$, then the triplet of X consists of

$$\left(X, \frac{\partial X}{\partial x_i}, \frac{\partial^2 X}{\partial x_i \partial x_j}\right)$$

for $i = 1, \ldots, n$ and $j \le i$.

The doublet consists of the first $n + 1$ elements of the triplet.

At the start of the function evaluation the triplets of the variables x_k must be set and these are simply $(x_k, e_k, 0)$ where e_k is the unit vector with 1 in the kth place, and 0 is the null matrix. If the function evaluation is expanded as a Wengert list [17] consisting of three types of operations,

- addition and subtraction,
- multiplication and division,
- nonlinear scalar functions,

then the arithmetic required to correctly update the triplets is easily deduced.

- If $X_k = X_l + X_m$, $l, m < k$, then to obtain the triplet of X_k, the elements of the triplets of X_l and X_m are simply added together element by element.
- If $X_k = X_l\, X_m$, $l, m < k$, then the background arithmetic is more complex as

$$\frac{\partial X_k}{\partial x_i} = X_l \frac{\partial X_m}{\partial x_i} + X_m \frac{\partial X_l}{\partial x_i}$$

and

$$\frac{\partial^2 X_k}{\partial x_i \partial x_j} = \frac{\partial X_l}{\partial x_j}\frac{\partial X_m}{\partial x_i} + X_l \frac{\partial^2 X_m}{\partial x_i \partial x_j} + \frac{\partial X_m}{\partial x_j}\frac{\partial X_l}{\partial x_i} + X_m \frac{\partial^2 X_l}{\partial x_i \partial x_j}.$$

As all these terms are stored in the triplets of X_l and X_m, given the triplets of X_l and X_m the triplet of X_k can be computed by a standard routine.

- If $X_k = \phi\,(X_m)$, $m < k$, then

$$\frac{\partial X_k}{\partial x_i} = \phi'(X_m)\frac{\partial X_m}{\partial x_i}$$

and

$$\frac{\partial^2 X_k}{\partial x_i \partial x_j} = \phi''(X_m)\frac{\partial X_m}{\partial x_i}\frac{\partial X_m}{\partial x_j} + \phi'(X_m)\frac{\partial^2 X_m}{\partial x_i \partial x_j}.$$

To perform this operation the values of $\phi'\,(X_m)$ and $\phi''\,(X_m)$ must be calculated with $\phi(X_m)$; all the other data is contained in the triplet of X_m.

Illustrative Example 1: Forward Mode

Consider the simple function

$$f(x) = (x_1 x_2 + \sin x_1 + 4)\left(3x_2^2 + 6\right)$$

In this case $n = 2$ and each triplet contains 6 floating point numbers, the value of X, its *gradient*, and the upper half of its Hessian. To evaluate the function, gradient, and Hessian, first expand the function in the Wengert list as shown in column 1 and then evaluate the triplets one by one. The evaluation is performed at the point (0, 1) below.

X_k	triplet(X_k)
$X_1 = x_1$	0, 1, 0, 0, 0, 0
$X_2 = x_2$	1, 0, 1, 0, 0, 0
$X_3 = X_1 X_2$	0, 1, 0, 0, 1, 0
$X_4 = \sin X_1$	0, 1 , 0, 0, 0, 0
$X_5 = X_3 + X_4$	0, 2, 0, 0, 1, 0
$X_6 = X_5 + 4$	4, 2, 0, 0, 1, 0
$X_7 = X_2^2$	1, 0, 2,0,0, 2
$X_8 = 3X_7$	3, 0, 6, 0, 0, 6
$X_9 = X_8 + 6$	9, 0, 6, 0, 0, 6
$X_{10} = X_6 X_9$	36, 18, 24, 0, 21, 24

The last row contains the values of the function, gradient and Hessian. The values for this simple problem can be easily verified by direct differentiation.

In practice forward automatic differentiation may be implemented in many ways, one possibility in many modern computer languages is to introduce the new data type triplet and over-write the meaning of the standard operators and functions so they perform the arithmetic described above. The code for the function evaluation can then be written normally without recourse to the Wengert list. Details of an implementation in Ada are given in [13]. A single run through a function evaluation code then computes the function, gradient and Hessian. If S is the store required to compute $f(x)$ then this method requires $(1 + n + n(n + 1)/2)S$ store. If M is the number of operations required to compute $f(x)$ then $(1 + 3n + 7n^2)M$ is a pessimistic bound on the operations required to compute the function, gradient and Hessian. Additional overheads are incurred to access the data type and the over-written operator subroutines. The efficiency is often improved by treating the triplet as a vector array and using sparse storage techniques. The number of zeros in the triplets of the above simple example illustrates the strength of the sparse form to calculate full Hessians. Maany reports the following results for the CPU time to differentiate the 50-dimensional Helmholz function (for details see [10]).

	Doublets		triplets	
	full	sparse	full	sparse
f	1.36	0.44	60.29	0.44
$f, \nabla f$	9.24	3.42	68.68	3.52
$f, \nabla f, \nabla^2 f$	N/A	N/A	476.36	20.69

The CPU time for calculating f alone within the full triplet package rises dramatically as although the derivative calculations are switched off the full package still allocates the space for the full triplet. Using the sparse package is also especially helpful if n is large and $f(x)$ is a partially separable function, i. e.

$$f(x) = \sum_k f_k(x)$$

where $f_k(x)$ only depends on a small number V_k of the n variables, as then, throughout the calculation of $f_k(x)$, the *sparse triplet* will only contain at most $1 + V_k + V_k(V_k + 1)/2$ nonzeros, and V_k will replace n in all the operation bounds, to give $\sum_k (1 + 3V_k + 7V_k^2)M_k$ operations.

One of the main purposes for calculating the Hessian matrix is to use it in optimization calculations. The truncated Newton method can be written so that it either requires the user to provide f, ∇f, and $\nabla^2 f$ at each outer iteration or f, ∇f at each outer iteration and $(\nabla^2 f)\, p$ at each inner iteration. The first method is ideally suited to be combined with sparse triplet differentiation. The algorithm is described in [9] and results given on functions of up to $n = 3000$ in [8]. The calculation of $(\nabla^2 f)\, p$ can also be undertaken simply by a modification of the triplet method.

In [7] the conclusion was drawn that

> the *sparse doublet* and sparse triplet codes in Ada enable normal code to be written for the func-

tion f and accurate values of ∇f and $\nabla^2 f$ to be obtained reliably by the computer. The major hope for automatic differentiation is therefore achieved.

Implementations are also available in Pascal.SC, C++, and Fortran90. The NOC Optima Library [1] code, *OPFAD*, implements the sparse doublet and triplet methods described above in Fortran90.

The Mixed Method

The advent of reverse automatic differentiation, A. Griewank [10], raised the hope that quicker ways could be found. The bound on the operations needed to compute the Hessian by the full forward triplet method contains the term $1/2n^2M$; by using a mixed method this is not required. The simplest mixed method is to use reverse automatic differentiation to compute the gradient which, [10], only requires $5M$ operations to compute the function and gradient for any value of n. This can be repeated at appropriate steps h along each axis, i. e. at $x + he_i$, $i = 1, \ldots, n$, and simple differences applied to the gradient vectors to calculate the Hessian in less than $5(n + 1)(M + 1)$ operations.

Illustrative Example 2: Reverse Differentiation

To obtain the gradient by *reverse differentiation* we must introduce the adjoint variables X_k^* and reverse back through the list. These rules are discussed in the previous article, but for convenience are repeated. If in the calculation of $f(x)$,

$$X_k = \phi(X_i, X_j), \qquad i, j < k,$$

then in the reverse pass

$$X_i^* = X_i^* + \frac{\partial \phi}{\partial X_i} X_k^*$$

and

$$X_j^* = X_j^* + \frac{\partial \phi}{\partial X_j} X_k^*.$$

For the same example the steps needed to calculate the gradient by reverse differentiation are

X_k^*	X_k^*
$X_{10}^* = 1$	1
$X_9^* = X_{10}^* X_6$	4
$X_6^* = X_{10}^* X_9$	9
$X_8^* = X_9^*$	4
$X_7^* = 3X_8^*$	12
$X_2^* = 2X_2 X_7^*$	24
$X_5^* = X_6^*$	9
$X_4^* = X_5^*$	9
$X_3^* = X_5^*$	9
$X_1^* = X_4^* \cos X_1$	9
$X_2^* = X_2^* + X_1 X_3^*$	24
$X_1^* = X_1^* + X_2 X_3^*$	18

giving the gradient as (18, 24) in agreement with the forward calculation. To perform this calculation the values of X_6 and X_9 were required which had been calculated during the function value calculation. The reverse gradient calculation must, therefore, follow a forward function evaluation calculation and the required data must be stored.

The bound $5M$ on the number of operations required to calculate the gradient is often very pessimistic, especially when the function evaluation uses matrix operations, [15], standard subroutines, [5], or when efficient sparse storage is used, [6]. The store required by this simple approach is simply that needed to calculate the gradient by reverse differentiation. The original reverse method required $O(M)$ store, but Griewank [11] describes how the store required can be reduced to $O(S \log M)$ at the cost of increasing the operation bound to $O(M \log M)$.

The accuracy obtained by calculating the Hessian by simple differences will depend on h but will often be sufficient as accurate Hessians are rarely required in optimization. Many software packages for calculating the gradient by reverse differentiation now exist, including the Optima Library Code *OPRAD* [1].

In 1998 the most widely used code to calculate gradients automatically is probably the *ADIFOR* code, [3], many examples of its use are given in that reference; unfortunately this implements a 'statement level hybrid mode'. In this, each assignment statement

$$Y_i = \Psi(Y_j, j < i, j \in J)$$

is treated in turn and the gradient, $\frac{\partial \Psi}{\partial Y_j}, j \in J$, computed efficiently by RAD but then to obtain the Doublet

$$\frac{\partial Y_i}{\partial x_m} = \sum_j \frac{\partial \Psi}{\partial Y_j} \frac{\partial Y_j}{\partial x_m}$$

many multiplications and additions may be required leading to a high operation count.

Reverse Method

A fully automatic approach could start by obtaining the Wengert list for the function and gradient as calculated by reverse automatic differentiation. This list will contain at most $5M$ steps. Then a forward sparse Doublet pass through this list could be performed that would need less than $(1 + 3n)\ 5M$ operations. The Doublet formed for the same example is illustrated below. In the Wengert list all identical Doublets are merged and composite steps involving more than one operation are split, it will be observed that the last two rows of the Doublet contain the gradient and Hessian, as desired, and that the number of operations, 22, is much less than the bound $5M = 50$. The storage requirement for this approach, when n is large, is considerably greater than that needed by the difference method. An alternative would be to perform a reverse pass through the gradient list. A full discussion is given in [4], who shows the two are identical in arithmetic, storage and operation count. His experience with his Ada implementation showed that the performance was very machine dependent. If the sparse Doublet approach is used with this reverse method on the partially separable function described above then the bound on the operations needed to obtain the Hessian reduces to $\sum_k 5(V_k + 1)(M_k + 1)$, a considerable saving. An early implementation, *PADRE2*, is described in [12]. A more recent code, *ADOL-F*, is described in [16]. Christianson's method is implemented in OPRAD, mentioned above. It should perhaps be mentioned that all the above methods can be hand-coded to solve any important problem without incurring the overheads still associated with most automatic packages, many of the helping hands described in [5] are still not implemented in an automatic package.

Further methods for speeding up the calculation of the Hessian are described in ► Automatic Differentiation: Calculation of Newton Steps.

Illustrative Example 3: Reverse Gradient, Forward Hessian

The variables in the Wengert list of the function and gradient calculation will be denoted by Y.

Y_k	Doublet Y_k
$Y_1 = x_1$	0, 1, 0
$Y_2 = x_2$	1, 0, 1
$Y_3 = Y_1 Y_2$	0, 1, 0
$Y_4 = \sin Y_1$	0, 1, 0
$Y_5 = Y_3 + Y_4$	0, 2, 0
$Y_6 = Y_5 + 4$	4, 2, 0
$Y_7 = Y_2^2$	1, 0, 2
$Y_8 = 3Y_7$	3, 0, 6
$Y_9 = Y_8 + 6$	9, 0, 6
$Y_{10} = Y_6 Y_9$	36, 18, 24
$Y_{11} = 1$	1, 0, 0
$Y_{12} = Y_{11} Y_6$	4, 2, 0
$Y_{13} = Y_{11} Y_9$	9, 0, 6
$Y_{14} = 3Y_{12}$	12, 6, 0
$Y_{15} = Y_2 Y_{14}$	12, 6, 12
$Y_{16} = 2Y_{15}$	24, 12, 24
$Y_{17} = \cos Y_1$	1, 0, 0
$Y_{18} = Y_{17} Y_{13}$	9, 0, 6
$Y_{19} = Y_1 Y_{13}$	0, 9, 0
$Y_{20} = Y_2 Y_{13}$	9, 0, 15
$Y_{21} = Y_{18} + Y_{20}$	18, 0, 21
$Y_{22} = Y_{16} + Y_{19}$	24, 21, 24

See also

- ► Automatic Differentiation: Calculation of Newton Steps
- ► Automatic Differentiation: Geometry of Satellites and Tracking Stations
- ► Automatic Differentiation: Introduction, History and Rounding Error Estimation
- ► Automatic Differentiation: Parallel Computation
- ► Automatic Differentiation: Point and Interval
- ► Automatic Differentiation: Point and Interval Taylor Operators
- ► Automatic Differentiation: Root Problem and Branch Problem
- ► Nonlocal Sensitivity Analysis with Automatic Differentiation

References

1. Bartholomew-Biggs MC Optima library. Numerical Optim. Centre Univ. Hertfordshire, England
2. Berz M, Bischof Ch, Corliss G, Griewank A (eds) (1996) Computational differentiation: techniques, applications, and tools. SIAM, Philadelphia
3. Bischof Ch, Carle A (1996) Users' experience with ADIFOR 2.0. In: Berz M, Bischof Ch, Corliss G, Griewank A (eds) Computational Differentiation: Techniques, Applications, and Tools. SIAM, Philadelphia, pp 385–392
4. Christianson DB (1992) Automatic Hessians by reverse accumulation. IMA J Numer Anal 12:135–150
5. Christianson DB, Davies AJ, Dixon LCW, Zee P Van der, (1997) Giving reverse differentiation a helping hand. Optim Methods Softw 8:53–67
6. Christianson DB, Dixon LCW, Brown S (1996) Sharing storage using dirty vectors. In: Berz M, Bischof Ch, Corliss G, Griewank A (eds) Computational Differentiation: Techniques, Applications, and Tools. SIAM, Philadelphia, pp 107–115
7. Dixon LCW (1993) On automatic differentiation and continuous optimization. In: Spedicato E (ed) Algorithms for continuous optimization: the state of the art. NATO ASI series. Kluwer, Dordrecht, pp 501–512
8. Dixon LCW, Maany Z, Mohsenina M (1989) Experience using truncated Newton for large scale optimization. In: Bromley K (ed) High Speed Computing, vol 1058. SPIE–The Internat. Soc. for Optical Engineering, Bellingham, WA, pp 94–104
9. Dixon LCW, Maany Z, Mohseninia M (1990) Automatic differentiation of large sparse systems. J Econom Dynam Control 14(2)
10. Griewank A (1989) On automatic differentiation. In: Iri M, Tanabe K (eds) Mathematical programming: recent developments and applications. Kluwer, Dordrecht, pp 83–108
11. Griewank A (1992) Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. Optim Methods Softw 1:35–54
12. Kubota K (1991) PADRE2, A FORTRAN precompiler yielding error estimates and second derivatives. In: Griewank A and Corliss GF (eds) Automatic differentiation of algorithms: theory, implementation, and application. SIAM, Philadelphia, pp 251–262
13. Maany ZA (1989) Ada automatic differentiation packages. Techn Report Hatfield Polytechnic NOC TR224
14. Rall LB (1981) Automatic differentiation: techniques and applications. Lecture Notes Computer Sci, vol 120. Springer, Berlin
15. Shiriaev D (1993) Fast automatic differentiation for vector processors and reduction of the spatial complexity in a source translation environment. PhD Thesis Inst Angew Math Univ Karlsruhe
16. Shiriaev D, Griewank A (1996) ADOL–F Automatic differentiation of Fortran codes. In: Berz M, Bischof Ch, Corliss G, Griewank A (eds) Computational Differentiation: Techniques, Applications, and Tools. SIAM, Philadelphia, pp 375–384
17. Wengert RE (1964) A simple automatic derivative evaluation program. Comm ACM 7(8):463–464

Automatic Differentiation: Calculation of Newton Steps

LAURENCE DIXON
Numerical Optim. Centre, University Hertfordshire, Hatfield, UK

MSC2000: 90C30, 65K05

Article Outline

Keywords

Automatic differentiation; Jacobian matrix; Hessian matrix; Sparsity; Newton step

Many algorithms for solving optimization problems require the minimization of a merit function, which may be the original objective function, or the solution to sets of simultaneous nonlinear equations which may involve the constraints in the problem. To obtain second order convergence near the solution algorithms to solve both rely on the calculation of Newton steps.

When solving a set of nonlinear equations

$$s_j(x) = 0, \qquad j = 1, \ldots, n,$$

the *Newton step* d at $x^{(0)}$, $x \in \mathbf{R}^n$, is obtained by solving the linear set of equations

$$\sum_i \frac{\partial s_j}{\partial x_i} d_i = -s_j, \qquad j = 1, \ldots, n,$$

where both the derivatives $\partial s_j / \partial x_i$ and the vector function s_j are evaluated at a point, which we will denote by $x^{(0)}$.

For convenience we introduce the *Jacobian matrix* J and write the equation as

$$Jd = -s$$

When minimizing a function $f(x)$ the Newton equation becomes

$$\sum_i \frac{\partial^2 f}{\partial x_i \partial x_j} d_j = -\frac{\partial f}{\partial x_j}$$

where all the derivatives are calculated at a point, again denoted by $x^{(0)}$.

In terms of the *Hessian*, H, and the gradient, g, this can be written

$$Hd = -g$$

Automatic differentiation can be used to calculate the gradient, Hessian and Jacobian, but it can also be used to calculate the Newton step directly without calculating the matrices. In this article we will first discuss the calculation of the Jacobian, then extend briefly the calculation of the gradient and Hessian, which was the subject of ► Automatic differentiation: Calculation of the Hessian, and finally discuss the direct calculation of the Newton step.

Jacobian Calculations

If the functions s_j were each evaluated as separate entities, requiring M_j operations, then the derivatives could be evaluated by reverse automatic differentiation in 5 M_j operations. For many sets of functions it would, however, be very inefficient to evaluate the set s in this way, as considerable savings could be made by calculating threads of operations common to more than one s_j only once. In such situations the number of operations M required to evaluate the set s may be much less than $\sum_j M_j$. Under these circumstances the decision on how the Jacobian should be evaluated becomes much more complicated.

Before the advent of automatic differentiation the Jacobian was frequently approximated by one-sided differences

$$\frac{\partial s_j}{\partial x_i} = \frac{s_j\left(x^{(0)} + he_i\right) - s_j\left(x^{(0)}\right)}{h}$$

If the vector function s requires M Wengert operations, then the Jacobian would need $(n + 1)$ M operations by this approach. The accuracy of the result depends on a suitable choice of h. If simple forward automatic differentiation using doublets (see ► Automatic differentiation: Calculation of the Hessian) is used, an accurate Jacobian is obtained at a cost of $3nM$ operations. If a Newton step is to be calculated then the Jacobian must be square and so the simple reverse mode, which involves a backward pass through the Wengert list for each subfunction, would be bounded by 5 n M operations.

Most large Jacobians are sparse and M.J.D. Powell, A.R. Curtis, and J.R. Reid [5], introduced the idea of combining columns i that had no common nonzeros. Then, provided the sparsity pattern of J is known, the values in those columns can be reconstructed by a reduced number of differences. If the number of such *PCR groups* required to cover all the columns is c then the operations count is reduced to $(c + 1)$ M. For example, the columns of the following 5×5 sparse Jacobian could be divided into 3 groups

$$\begin{bmatrix} ■ & ▲ & 0 & 0 & ★ \\ ■ & ▲ & 0 & ★ & 0 \\ 0 & ▲ & ■ & ★ & 0 \\ 0 & 0 & ■ & 0 & ★ \\ ■ & ▲ & 0 & 0 & ★ \end{bmatrix}$$

indicated by ■, ▲, and ★.

This same grouping could be used with forward automatic differentiation to produce an accurate Jacobian in at most $3cM$ operations. If the sparse Doublet is used, the full benefit of *sparsity* within the calculation of the s_j is obtained, as well as the benefit due to sparsity in the Jacobian, without the need to determine the column groupings. Results showing the advantage of calculating large ($n = 5000$) Jacobians this way are given in [15] and summarised in [7].

It is possible for the calculation of some s_j to be independent of other s that do contain a common thread. It would obviously be efficient to calculate these s_j by reverse differentiation, requiring $5Mj$ operations. Reverse differentiation will also be appropriate if the common thread has less outputs than inputs. Then sparse reverse doublets, [2], should be used. These are implemented in OPRAD, see ► Automatic differentiation: Calculation of the Hessian.

T.F. Coleman et al. [3,4] demonstrated that calculating some columns using groups in the forward mode and some rows using groups in the reverse mode is considerably more efficient than using either alone. All nonzeros of the Jacobian must be included in a row and/or column computed. Similar results follow if some columns are computed using sparse doublets and some rows using the sparse reverse method. If C is the maximum number of nonzeros in a row within the columns computed forward and R the maximum number of nonzeros in a column within the rows computed in reverse then a crude bound on the number of operations is $(3C + 5R)M$. This bound does not allow for the additional sparsity in the early calculations nor for the fact that for some reverse calculations M_j should replace M. The selection of rows and columns taking account of such considerations is still unresolved.

But the advantages to be obtained can be appreciated by considering the arrow-head Jacobian, where only the diagonal elements and the last row and column contain nonzeros. If the gradient of s_n is computed using sparse reverse doublets this will require at most $5M_n$ operations and if the other gradients are computed using sparse forward doublets, no doublet will contain more than 2 nonzeros, so the operations will be bounded by $6M$. The total operations required in this case is independent of n.

The Extended Matrix

If the calculation of the functions s_j proceeds by a sequence of steps

$$X_k = x_k, \qquad k = 1, \ldots, n,$$
$$X_k = \phi_k(X_l, l \in L, l < k),$$
$$k = n + 1, \ldots, M + n,$$

with

$$s_j = X_{M+j}, \qquad j = 1, \ldots, n,$$

then

$$\frac{\partial X_k}{\partial x_m} = 0, \quad m \neq k, \quad k = 1, \ldots, n,$$
$$\frac{\partial X_k}{\partial x_k} = 1,$$

and

$$\frac{\partial X_k}{\partial x_m} = \sum_l \frac{\partial \phi_k}{\partial X_l} \frac{\partial X_l}{\partial x_m}.$$

If we now denote $\partial X_k / \partial x_m$ by Y_k and $\partial \phi_k / \partial X_l$ by L_{kl}, then this becomes

$$Y_k = \sum_l L_{kl} Y_l$$

i. e. the kth row of the matrix-vector product

$$(I - L)Y,$$

where the elements in the first n rows of L are all zeros, and then

$$\frac{\partial s_j}{\partial x_m} = Y_{M+j}.$$

Obtaining the Jacobian by the forward method may be considered as equivalent to solving

$$(I - L)Y = e_m.$$

Turning now to the reverse method if

$$X_k = \phi_k(X_l),$$

then the adjoint variable X_l^* contains a term

$$X_l^* = X_l^* + \frac{\partial \phi_k}{\partial X_l} X_k^*$$

which is the lth row of the matrix-vector product

$$(I - L^\top)X^*.$$

To obtain the gradient of s_m is therefore equivalent to solving

$$(I - L^\top)X^* = e_{M+m},$$

then

$$\frac{\partial s_m}{\partial x_i} = X_i^*.$$

So both the calculation of the Jacobian by the forward and backward method are equivalent to solving a very sparse set of equations. If the Wengert list is used, each row of L contains at most two nonzeros. It has therefore been suggested that methods for solving linear equations with sparse matrices could be used to calculate J, A. Griewank and S. Reese [14] suggested using the Markowitz rule, while U. Geitner, J. Utke and Griewank [11] applied the *method of Newsam and Ramsdell*.

Hessian Calculations

The calculation of the Hessian, as discussed in ► Automatic differentiation: Calculation of the Hessian, can also be formulated as a sparse matrix calculation. Using the notation of ► Automatic differentiation: Calculation of the Hessian if the calculation of $f(x)$ consists of

$$X_k = \phi_k(X_m, m < k, m \in M_k),$$

then the reverse gradient calculation consists of

$$X_m^* = X_m^* + X_k^* \frac{\partial \phi_k}{\partial X_m}, \quad m \in M_k.$$

If now we denote

$$Y_k = \frac{\partial X_k}{\partial x_i}, \qquad k = 1, \ldots, M,$$

and

$$Y_k = \frac{\partial X_{2M-k+1}^*}{\partial x_i}, \qquad k = M+1, \ldots, 2M,$$

then we obtain

$$Y_k = \frac{\partial \phi_k}{\partial X_m} Y_m, \qquad k = 1, \ldots, M,$$

and

$$Y_{2M+1-m} = Y_{2M+1-m} + Y_{2M+1-k} \frac{\partial \phi_k}{\partial X_m} + X_k^* \frac{\partial^2 \phi_k}{\partial X_m \partial X_j} Y_j.$$

The second derivatives are 1, if ϕ is a multiplication, 0 if ϕ is an addition, and if ϕ is unary only nonzero if $j = m$. If we denote these second order terms by B, the calculation of $H\, e_i$ is equivalent to solving

$$\begin{bmatrix} I - L & 0 \\ B & I - L^S \end{bmatrix} Y = \begin{pmatrix} e_i \\ 0 \end{pmatrix}.$$

Here the superscript S indicates that L has been transposed through both diagonals. The ith column of the Hessian is then the last n values of Y. For the illustrative example

$$f(x) = (x_1 x_2 + \sin x_1 + 4)(3x_2^2 + 6)$$

used in ► Automatic differentiation: Calculation of the Hessian, the off-diagonal nonzeros in the matrix which we will denote by K, are

$$\begin{aligned}
K_{3,1} &= K_{20,18} = X_2, \\
K_{3,2} &= K_{19,18} = X_1, \\
K_{4,1} &= K_{20,17} = \cos X_1, \\
K_{5,3} &= K_{18,16} = 1, \\
K_{5,4} &= K_{17,16} = 1, \\
K_{6,5} &= K_{16,15} = 1, \\
K_{7,2} &= K_{19,14} = 2X_2, \\
K_{8,7} &= K_{14,13} = 3, \\
K_{9,8} &= K_{13,12} = 1, \\
K_{10,6} &= K_{15,11} = X_9, \\
K_{10,9} &= K_{12,11} = X_6, \\
K_{12,6} &= K_{14,9} = X_{10}^*, \\
K_{19,1} &= K_{20,2} = X_3^*, \\
K_{19,2} &= 2X_7^*, \\
K_{20,1} &= -X_4^* \sin X_1,
\end{aligned}$$

L contains 11 nonzeros and B contains 6. The matrix is very sparse and the same sparse matrix techniques could be used to solve this system of equations.

The Newton Step

As the notation is easier we will consider the Jacobian case.

We have shown that if we solve $(I - L)\, Y = e_m$, then column m of the Jacobian J is in the last n terms of Y. If we wish to evaluate $J\, p$ we simply have to solve

$$(I - L)Y = p'$$

where p' has its first n terms equal to p and the remaining terms zero. Then the solution is again in the last n terms of Y. To calculate the Newton step we know $J\, d$ as it must be equal to $-\, s$, but we do not know d. We must therefore add the equations

$$Y_{M+i} = -s_i$$

to the equations, and delete the equations $Y_i = p_i$. For convenience we will partition L, putting the first n columns into A, retaining L for the remainder. So we have to solve

$$\begin{bmatrix} -A & I - L \\ 0 & E \end{bmatrix} \begin{pmatrix} d \\ Y \end{pmatrix} = \begin{pmatrix} 0 \\ -s \end{pmatrix}$$

for d. The matrix E is rectangular and is full of zeros except for the diagonals which are 1. Solving for d gives

$$E(I-L)^{-1}Ad = -s,$$

so

$$J = E(I-L)^{-1}A,$$

which is also the *Schur complement* of the sparse set of equations.

One popular way of solving a sparse set of equations is to form the Schur complement and solve the resulting equations, in this instance this becomes 'form J and solve $J\ d = -\ s$', which would be the normal indirect method. This also justifies the attention given in this article to the efficient calculation of J.

Griewank [12] observed that it may be possible to calculate the Newton step more cheaply than forming J and then solving the Newton equations. Utke [16] demonstrated that a number of ways of solving the sparse set of equations were indeed quicker. His implementation was compatible with ADOL-C and included many rules for eliminating variables. This approach was motivated by noting that if the Jacobian $J = D + a\ b^{\intercal}$, where D is diagonal and a and b vectors, then J is full and so solving $J\ x = -\ s$ is an $O(n^3)$ operation. However introducing one extra variable $z = b^{\intercal}\ x$ enables the *extended matrix* to be solved very cheaply

$$b^{\top}x - z = 0,$$
$$Dx + az = -s$$

gives

$$x = -D^{-1}(az + s),$$
$$z = -b^{\top}D^{-1}(az + s),$$

so

$$z = -(1 + b^{\top}D^{-1}a)^{-1}b^{\top}D^{-1}s,$$

and then x may be determined by substitution, which is an $O(n)$ operation. The challenge to find an automatic process that finds such short cuts is still open.

L.C.W. Dixon [6] noted that the extended matrix is an *echelon form*. An echelon matrix of degree k has ones on the k super-diagonal and zeros above it. If the lower part is sparse and contains NNZ nonzeros then the Schur complement can be computed in $kNNZ$ operations and the Newton step obtained by solving the resulting equations in $O(k^3)$ steps. The straight forward sparse system is an echelon form with $k = n$, so he suggested that by re-arranging rows and columns it might be possible to reduce k. This would reduce the operations needed for both parts of the calculation. Many sorting algorithms have been proposed for reducing the echelon index of sparse matrices. J.S. Duff et al. [9] discuss the performance of methods known as P^4 and P^5. R. Fletcher [10] introduced SPK1. Dixon and Z. Maany [8] introduced another which when applied to the extended matrix of the extended Rosenbrock function reduces the echelon index from n to $n/2$ and gives a diagonal Schur complement. It follows that this method, too, has considerable potential.

All these approaches still require further research.

Truncated Methods

Experience using the *truncated Newton* code has led many researchers to doubt the wisdom of calculating accurate Newton steps. Approximate solutions are often preferred in which the *conjugate gradient* method is applied to $H\ d = -\ g$; this can be implemented by calculating $H\ p$ at each inner iteration. $H\ p$ can be calculated very cheaply by a single forward doublet pass with initial values set at p through list for g obtained by reverse differentiation. The operations required to compute $H\ p$ are therefore bounded by $15M$.

If an iterative method is used to solve $J\ d = -\ s$, the products $J\ p$ and $J^{\intercal}\ v$ can both be obtained cheaply, the first by forward, the second by reverse automatic differentiation.

See also

- ► Automatic Differentiation: Calculation of the Hessian
- ► Automatic Differentiation: Geometry of Satellites and Tracking Stations
- ► Automatic Differentiation: Introduction, History and Rounding Error Estimation
- ► Automatic Differentiation: Parallel Computation
- ► Automatic Differentiation: Point and Interval
- ► Automatic Differentiation: Point and Interval Taylor Operators

- ▶ Automatic Differentiation: Root Problem and Branch Problem
- ▶ Dynamic Programming and Newton's Method in Unconstrained Optimal Control
- ▶ Interval Newton Methods
- ▶ Nondifferentiable Optimization: Newton Method
- ▶ Nonlocal Sensitivity Analysis with Automatic Differentiation
- ▶ Unconstrained Nonlinear Optimization: Newton–Cauchy Framework

References

1. Berz M, Bischof Ch, Corliss G, Griewank A (eds) (1996) Computational differentiation: techniques, applications, and tools. SIAM, Philadelphia
2. Christianson DB, Dixon LCW (1992) Reverse accumulation of Jacobians in optimal control. Techn Report Numer Optim Centre, School Inform Sci Univ Hertfordshire 267
3. Coleman TF, Cai JY (1986) The cyclic coloring problem and estimation of sparse Hessian matrices. SIAM J Alg Discrete Meth 7:221–235
4. Coleman TF, Verma A (1996) Structure and efficient Jacobian calculation. In: Berz M, Bischof Ch, Corliss G, Griewank A (eds) Computational Differentiation: Techniques, Applications, and Tools. SIAM, Philadelphia, pp 149–159
5. Curtis AR, Powell MJD, Reid JK (1974) On the estimation of sparse Jacobian matrices. J Inst Math Appl 13:117–119
6. Dixon LCW (1991) Use of automatic differentiation for calculating Hessians and Newton steps. In: Griewank A, Corliss GF (eds) Automatic Differentiation of Algorithms: Theory, Implementation, and Application. SIAM, Philadelphia, pp 114–125
7. Dixon LCW (1993) On automatic differentiation and continuous optimization. In: Spedicato E (ed) Algorithms for continuous optimisation: the state of the art. NATO ASI series. Kluwer, Dordrecht, pp 501–512
8. Dixon LCW, Maany Z (Feb. 1988) The echelon method for the solution of sparse sets of linear equations. Techn Report Numer Optim Centre, Hatfield Polytechnic NOC TR177
9. Duff IS, Anoli NI, Gould NIM, Reid JK (1987) The practical use of the Hellerman–Ranck P^4 algorithm and the P^5 algorithm of Erisman and others. Techn Report AERE Harwell CSS213
10. Fletcher R, Hall JAJ (1991) Ordering algorithms for irreducible sparse linear systems. Techn Report Dundee Univ NA/131
11. Geitner U, Utke J, Griewank A (1991) Automatic computation of sparse Jacobians by applying the method of Newsam and Ramsdell. In: Griewank A, Corliss GF (eds) Automatic Differentiation of Algorithms: Theory, Implementation, and Application, SIAM, Philadelphia, pp 161–172
12. Griewank A (1991) Direct calculation of Newton steps without accumulating Jacobians. In: Griewank A, Corliss GF (eds) Automatic Differentiation of Algorithms: Theory, Implementation, and Application, SIAM, Philadelphia, pp 126–137
13. Griewank A, Corliss GF (eds) (1991) Automatic differentiation of algorithms: theory, implementation, and application. SIAM, Philadelphia
14. Griewank A, Reese S (1991) On the calculation of Jacobian matrices by the Markowitz rule. In: Griewank A, Corliss GF (eds) Automatic Differentiation of Algorithms: Theory, Implementation, and Application, SIAM, Philadelphia, pp 126–135
15. Parkhurst SC (Dec. 1990) The evaluation of exact numerical Jacobians using automatic differentiation. Techn Report Numer Optim Centre, Hatfield Polytechnic, NOC TR224
16. Utke J (1996) Efficient Newton steps without Jacobians. In: Berz M, Bischof Ch, Corliss G, Griewank A (eds) Computational Differentiation: Techniques, Applications, and Tools, SIAM, Philadelphia, pp 253–264

Automatic Differentiation: Geometry of Satellites and Tracking Stations

DAN KALMAN
American University, Washington, DC, USA

MSC2000: 26A24, 65K99, 85-08

Article Outline

Keywords

Astrodynamics; Automatic differentiation; Satellite orbit; Vector geometry

Satellites are used in a variety of systems for communication and data collection. Familiar examples of these systems include satellite networks for broadcasting video programming, meteorological and geophysical data observation systems, the global positioning system (GPS) for navigation, and military surveillance systems. Strictly speaking, these are systems in which satellites are just one component, and in which there are other primary subsystems that have no direct involvement with satellites. Nevertheless, they will be referred to as *satellite systems* for ease of reference.

Simple geometric models are often incorporated in simulations of satellite system performance. Important operational aspects of these systems, such as the times when satellites can communicate with each other or with installations on the ground (e. g. *tracking stations*), depend on dynamics of satellite and station motion. The geometric models represent these motions, as well as constraints on communication or data collection. For example, the region of space from which an antenna on the ground can receive a signal might be modeled as a cone, with its vertex centered on the antenna and axis extending vertically upward. The antenna can receive a signal from a satellite only when the satellite is within the cone. Taking into account the motions of the satellite and the earth, the geometric model predicts when the satellite and tracking station can communicate.

Elementary optimization problems often arise in these geometric models. It may be of interest to determine the closest approach of two satellites, or when a satellite reaches a maximum elevation as observed from a tracking station, or the extremes of angular velocity and acceleration for a rotating antenna tracking a satellite. Optimization problems like these are formulated in terms of geometric variables, primarily distances and angles, as well as their derivatives with respect to time. The derivatives appear both in the optimization algorithms, as well as in functions to be optimized. One of the previously mentioned examples illustrates this. When a satellite is being tracked from the ground, the antenna often rotates about one or more axes so as to remain pointed at the satellite. The angular velocity and acceleration necessary for this motion are the first and second derivatives of variables expressed as angles in the geometric configuration of the antenna and satellite. Determining the extreme values of these *derivatives* is one of the optimization problems mentioned earlier.

Automatic differentiation is a feature that can be included in a computer programming language to simplify programs that compute derivatives. In the situation described above, satellite system simulations are developed as computer programs that include computed values for the distance and angle variables of interest. With automatic differentiation, the values of derivatives are an automatic by-product of the computation of variable values. As a result, the computer programmer does not have to develop and implement the computer instructions that go into calculating derivative values. As a specific example of this idea, consider again the rotating antenna tracking a satellite. Imagine that the programmer has worked out the proper equations to describe the angular position of the antenna at any time. The simulation also needs to compute values for the angular velocity and acceleration, the first and second derivatives of angular position. However, the programmer does not need to work out the proper equations for these derivatives. As soon as the equations for angular position are included in the computer program, the programming language provides for the calculation of angular velocity and acceleration *automatically*. That is the effect of automatic differentiation. Because the derivatives of geometric variables such as distances and angles can be quite involved, automatic differentiation results in computer programs that are much easier to develop, debug, and maintain.

The preceding comments have provided a brief overview of geometric models for satellite systems, as well as associated optimization problems and the use of automatic differentiation. The discussion will now turn to a more detailed examination of these topics.

Geometric Models

The geometric models for satellite systems are formulated in the context of three-dimensional real space. A conventional rectangular coordinate system is defined by mutually perpendicular x, y, and z axes. The earth is modeled as a sphere or ellipsoid centered at the origin (0, 0, 0), with the north pole on the positive z axis, and the equator in the xy plane. The coordinate axes are considered to retain a constant orientation rel-

ative to the fixed stars, so that the earth rotates about the z axis.

In this setting, tracking station and satellite locations are represented by points moving in space. Each such moving point is specified by a vector valued function $\mathbf{r}(t) = (x(t), y(t), z(t))$ where t represents time. Geometric variables such as angles and distances can be determined using standard vector operations:

$$
\begin{aligned}
&c(x, y, z) = (cx, cy, cz),\\
&(x, y, z) \pm (u, v, w) = (x \pm u, y \pm v, z \pm w),\\
&(x, y, z) \cdot (u, v, w) = xu + yv + zw,\\
&(x, y, z) \times (u, v, w)\\
&\quad = (yw - zv, zu - xw, xv - yu),\\
&\|(x, y, z)\| = \sqrt{x^2 + y^2 + z^2}\\
&\quad = \sqrt{(x, y, z) \cdot (x, y, z)}.
\end{aligned}
$$

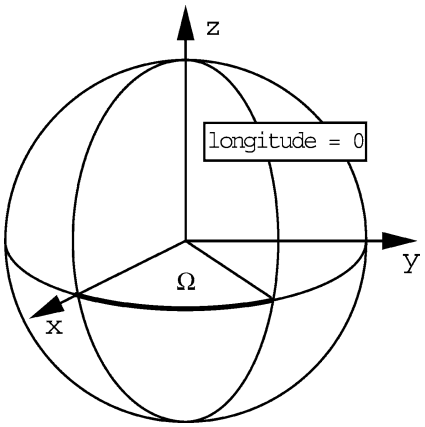
The distance between two points $\mathbf{r}$ and $\mathbf{s}$ is then given by $\| \mathbf{r} - \mathbf{s} \|$. The angle θ defined by rays from point $\mathbf{r}$ through points $\mathbf{p}$ and $\mathbf{q}$ is determined by

$$
\cos\theta = \frac{(\mathbf{p} - \mathbf{r}) \cdot (\mathbf{q} - \mathbf{r})}{\|\mathbf{p} - \mathbf{r}\| \cdot \|\mathbf{q} - \mathbf{r}\|}.
$$

A more complete discussion of vector operations, their properties, and geometric interpretation can be found in any calculus textbook; [9] is one example.

There are a variety of models for the motions of points representing satellites and tracking stations. The familiar conceptions of a uniformly rotating earth circled by satellites that travel in stable closed orbits is only approximately correct. For qualitative simulations of the performance of satellite systems, particularly at preliminary stages of system design, these models may be adequate. More involved models can take into account such effects as the asphericity of the gravitational field of the earth, periodic wobbling of the earth's axis of rotation, or atmospheric drag, to name a few. Modeling the motions of the earth and satellites with high fidelity is a difficult endeavor, and one that has been studied extensively. Good general references for this subject are [1,2,3,10].

For illustrative purposes, a few of the details will be presented for the simplest models, circular orbits



Automatic Differentiation: Geometry of Satellites and Tracking Stations, Figure 1
Earth rotation angle

around a spherical earth, uniformly spinning on a fixed axis. The radius of the earth will be denoted R_e.

As a starting point, the rotation of the earth can be specified by a single function of time, $\Omega(t)$, representing the angular displacement of the prime meridian from a fixed direction, typically the direction specified by the positive x axis (see Fig. 1.). At any time, the positive x axis emerges from the surface of the earth at some point on the equator. Suppose that at a particular time t, the point where the positive x axis emerges happens to be on the prime meridian, located at latitude 0 and longitude 0. Then $\Omega(t) = 0$ for that t. As time progresses, the prime meridian rotates away from the x axis, counter-clockwise as viewed by an observer above the north pole. The function Ω measures the angle of rotation, starting at 0 each time the prime meridian is aligned with the x axis, and increasing toward a maximum of 360° (2 π in radian measure) with each rotation of the earth. With a uniformly spinning earth, Ω increases linearly with t during each rotation.

Once Ω is specified, any terrestrial location given by a latitude ϕ, longitude λ, and altitude a can be transformed into absolute coordinates in space, according to the equations

$$\theta = \lambda + \Omega(t), \tag{1}$$

$$r = R_e + a, \tag{2}$$

$$x = r\cos\theta\cos\phi, \tag{3}$$

$$y = r \sin\theta \cos\phi, \tag{4}$$

$$z = r \sin\phi. \tag{5}$$

Holding latitude, longitude, and altitude constant, these equations express the position in space of a fixed location on the earth for any time, thereby modeling the point's motion. It is also possible to develop models for tracking stations that are moving on the surface of the earth, say on an aircraft or on a ship in the ocean. For example, if it is assumed that the moving craft is traveling at constant speed on a great circle arc or along a line of constant latitude, it is not difficult to express latitude and longitude as functions of time. In this case, the equations above reflect a dependence on t in λ and ϕ, as well as in Ω. A more complicated example would be to model the motion of a missile or rocket launched from the ground. This can be accomplished in a similar way: specify the trajectory in earth relative terms, that is, using latitude, longitude, and altitude, and then compute the absolute spatial coordinates (x, y, z). In each case, the rotation of the earth is accounted for solely by the effect of $\Omega(t)$.

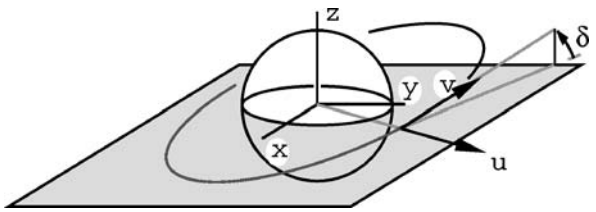
For a satellite in circular orbit, the position at any time is specified by an equation of the following form:

$$\mathbf{r}(t) = r[\cos(\omega t)\mathbf{u} + \sin(\omega t)\mathbf{v}].$$

In this equation, $\omega\, t$ is understood as an angle in radian measure for the sin and cos operations; r, ω, $\mathbf{u}$, and $\mathbf{v}$ are constants. The first, r is the length of the orbit circle's radius. It is equal to the sum of the earth's radius R_e and the satellite's altitude. The constant ω is the angular speed of the satellite. The satellite completes an orbit every $2\pi/\omega$ units of time, thus giving the *orbital period*. Both $\mathbf{u}$ and $\mathbf{v}$ are unit vectors: $\mathbf{u}$ is parallel to the initial position of the satellite; $\mathbf{v}$ is parallel to the initial velocity. See Fig. 2.

Mathematically, the equation above describes some sort of orbit no matter how the constants are selected. But not all of these are accurate descriptions of a free falling satellite in circular orbit. For one thing, $\mathbf{u}$ and $\mathbf{v}$ must be perpendicular to produce a circular orbit. In addition, there is a physical relationship linking r and ω. Assuming that the circular orbit follows Newton's laws of motion and gravitation, r and ω satisfy

$$\omega = Kr^{-\frac{3}{2}} \tag{6}$$



Automatic Differentiation: Geometry of Satellites and Tracking Stations, Figure 2
Circular orbit

where K is a physical constant that depends on both Newton's universal gravitational constant and the mass of the earth. Its numerical value also depends on the units of measurement used for time and distance. For units of hours and kilometers, the value of K is $2.27285 \cdot 10^6$. As this relationship shows, for a given altitude (and hence a given value of r), there is a unique angular speed at which a satellite will maintain a circular orbit. Equivalently, the altitude of a circular orbit determines the constant speed of the satellite, as well as the period of the satellite.

Generally, constants are chosen for a circular orbit based on some geometric description. Here is a typical approach. Assume that the initial position of the satellite is directly above the equator, with latitude 0, a given longitude, and a given altitude. In other words, assume that the initial position is in the plane of the equator, and so has a z coordinate of 0. (This is the situation depicted in Fig. 2.) Moreover, the initial heading of the satellite can be specified in terms of the angle it makes with the xy plane (which is the plane of the equator). Call that angle δ. From these assumptions we can determine values for the constants r, ω, $\mathbf{u}$, and $\mathbf{v}$ in the equation for $\mathbf{r}(t)$. Now the altitude for the orbit is constant, so the initial altitude determines r, as well as ω via equation (6). The initial latitude, longitude, and altitude also provide enough information to determine absolute coordinates (x, y, z) for the initial satellite position using equations (1)–(5). Accordingly, the unit vector $\mathbf{u}$ is given by

$$\mathbf{u} = \frac{(x, y, z)}{\|(x, y, z)\|}.$$

As already observed, the z coordinate of $\mathbf{u}$ will be 0. Finally, the unit vector $\mathbf{v}$ is determined from the initial position and heading. It is known that $\mathbf{v}$ make an angle

of δ with the xy plane, and hence makes an angle of $\pi/2 - \delta$ with the z axis. This observation can be expressed as the equation

$$\mathbf{v} \cdot (0, 0, 1) = \sin \delta.$$

It is also known that $\mathbf{v}$ must be perpendicular to $\mathbf{u}$, so

$$\mathbf{v} \cdot \mathbf{u} = 0.$$

Finally, since $\mathbf{v}$ is a unit vector,

$$\mathbf{v} \cdot \mathbf{v} = 1.$$

If $\mathbf{u} = (u_1, u_2, 0)$, then these three equations lead to $\mathbf{v} = (\pm u_2 \cos \delta, \mp u_1 \cos \delta, \sin \delta)$. The ambiguous sign can be resolved by assuming that the direction of orbit is either in agreement with or contrary to the direction of the earth's rotation. Assuming that the orbit is in the same direction as the earth's rotation, $\mathbf{v} = (-u_2 \cos \delta, u_1 \cos \delta, \sin \delta)$. The alternative possibility, that the satellite orbit opposes the rotation of the earth, is generally not practically feasible, so is rarely encountered.

The preceding paragraphs are intended to provide some insight about the mathematics used to describe the movement of satellites and terrestrial observers. Although the models presented here are the simplest ones available, they appear in the same general framework as much more sophisticated models. In particular, in any of these models, it is necessary to be able to compute instantaneous positions for satellites and terrestrial observers at any time during a simulation. Moreover, the use of vector algebra and geometry to set up the simple models is representative of the methods used in more complicated cases.

Sample Optimization Problems

Computer simulations of satellite system performance provide one tool for comparing alternative designs and making cost/benefit trade-offs in the design process. Optimization problems contribute both directly and indirectly. In many cases, system performance is characterized in terms of extreme values of variables: what is the maximum number of users that can be accommodated by a communications system? At a given latitude, what is the longest period of time during which at most three satellites can be detected from some point on the ground? In these examples, the optimization problems are directly connected with the goals of the simulation.

Optimization problems also arise indirectly as part of the logistics of the simulation software. This is particularly the case when a simulation involves events that trigger some kind of system response. Examples of such events include the passage of a satellite into or out of sunlight, reaching a critical level of some resource such as power or data storage, or the initiation or termination of radio contact with a tracking station. The detection of these events typically involves either root location or optimization. These processes are closely related: the root of an equation can usually be characterized as an extreme value of a variable within a suitable domain; conversely, optimization algorithms often generate candidate solutions by solving equations.

In many of these event identification problems, the independent variable is time. The objective functions ultimately depend on the geometric models for satellite and tracking station motion, and so can be formulated in terms of explicit functions of time. In contrast, some of the optimization problems that concern direct estimation of system performance seek to optimize that performance by varying design parameters. A typical approach to this kind of problem is to treat performance measures as functions of the parameters, where the values of the functions are determined through simulation. Both kinds of optimization are illustrated in the following examples.

Minimum Range

As a very simple example of an optimization problem, it is sometimes of interest to determine the closest approach of two orbiting bodies. Assume that a model has been developed, with $\mathbf{r}(t)$ and $\mathbf{s}(t)$ representing the positions at time t for the two bodies. The distance between them is then expressed as $\| \mathbf{r}(t) - \mathbf{s}(t) \|$. This is the objective function to be minimized. Observe that it is simply expressed as a composition of vector operations and the motion models for the two bodies.

A variation of this problem occurs when several satellites are required to stay in radio communication. In that case, an antenna on one satellite (at position A, say) may need to detect signals from two others (at positions B and C). In this setting, the measure of $\angle BAC$ is of interest. If the angle is wide, the antenna requires a correspondingly wide field of view. As the satellites proceed in their orbits, what is the maximum value of

the angle? Equivalently, what is the minimum value of the cosine of the angle? As before, the objective function in this minimization problem is easily expressed by applying vector operations to the position models for the satellites. If $\mathbf{a}(t)$, $\mathbf{b}(t)$, and $\mathbf{c}(t)$ are the position functions for the three satellites, then

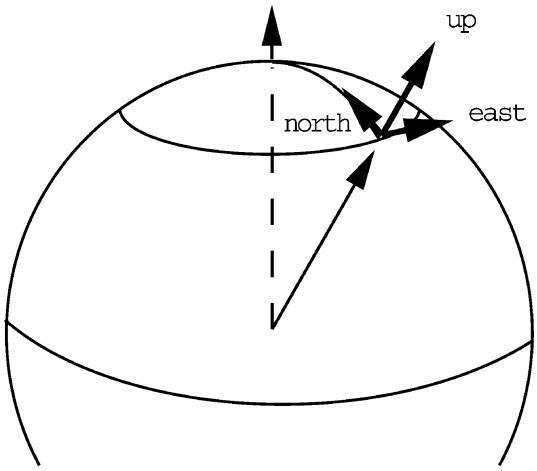
$$\cos \angle BAC = \frac{(\mathbf{b}-\mathbf{a})\cdot(\mathbf{c}-\mathbf{a})}{\|\mathbf{b}-\mathbf{a}\|\cdot\|\mathbf{c}-\mathbf{a}\|}.$$

This is a good example of combining vector operations with the models for satellite motion to derive the objective function in an optimization problem. The next example is similar in style, but mathematically more involved.

Direction Angles and Their Derivatives

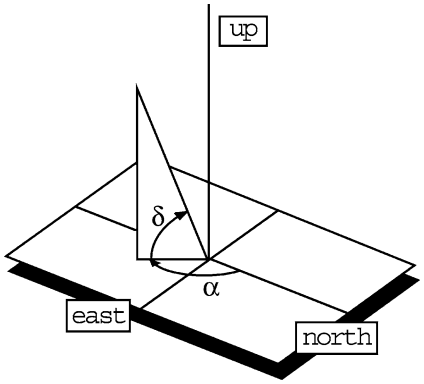
A common aspect of satellite system simulation is the representation of sensors of various kinds. The images that satellites beam to earth of weather systems and geophysical features are captured by sensors. Sensors are also used to locate prominent astronomical features such as the sun, the earth, and in some cases bright stars, in order to evaluate and control the satellite's attitude. Even the antenna used for communication is a kind of sensor. It is frequently convenient to define a coordinate system that is attached to a sensor, that is, define three mutually perpendicular axes which intersect at the sensor location, and which can be used as an alternate means to assign coordinates to points in space. Such a coordinate system is then used to describe the vectors from the sensor to other objects, and to model sensor sensitivity to signals arriving from various directions. With several different coordinate systems in use, it is necessary to transform information described relative to one system into a form that makes sense in the context of another system. This process also often involves what are called *direction angles*.

As a concrete example, consider an antenna at a fixed location on the earth, tracking a satellite in orbit. The coordinate system attached to the tracking antenna is the natural map coordinate system at that point on the earth: the local x and y axes point east and north, respectively, and the z axis points straight up (Fig. 3). The direction from the station to the satellite is expressed in terms of two angles: the elevation δ of the satellite above the local xy plane, and the compass angle α measured clockwise from north. (See Fig. 4.) To illustrate,



Automatic Differentiation: Geometry of Satellites and Tracking Stations, Figure 3
Local map coordinates

here is the meaning of an elevation of 30 degrees and a compass angle of 270 degrees. Begin by looking due north. Turn clockwise through 270 degrees, maintaining a line of sight that is parallel to the local xy plane. At that point you are looking due west. Now raise the line of sight until it makes a 30 degree angle with the local xy plane. This direction of view, with elevation 30 and compass angle 270 degrees, might thus be described as 30 degrees above a ray 270 degrees clockwise from due north. The elevation and compass angle are examples of direction angles. Looked at another way, if a spherical coordinate system is imposed on the local rectan-



Automatic Differentiation: Geometry of Satellites and Tracking Stations, Figure 4
Compass and elevation angles

gular system at the antenna, then every point in space is described by a distance and two angles. The angles are direction angles. Direction angles can be defined in a similar way for any local coordinate system attached to a sensor.

How are direction angles computed? In general terms, the basic idea is to define the local coordinate system in terms of moving vectors, and then to use vector operations to define the instantaneous value of direction angles. Here is a formulation for the earth based antenna. First, the local z axis points straight up. That means the vector from the center of the earth to the location of the antenna on the surface is parallel to the z axis. Given the latitude, longitude, and altitude of the antenna, its absolute position $\mathbf{r}(t) = (x, y, z)$ is computed using equations (1)–(5), as discussed earlier. The parallel unit vector is then given by $\mathbf{r}/\parallel \mathbf{r} \parallel$. To distinguish this from the global z axis, we denote it as the vector **up**. The vector pointing due east must be perpendicular to the up direction. It also must be parallel to the equatorial plane, and hence perpendicular to the global z axis. Using properties of vector cross products, a unit vector pointing east can therefore be expressed as

$$\mathbf{east} = \frac{(0,0,1) \times \mathbf{up}}{\|(0,0,1) \times \mathbf{up}\|}.$$

Finally, the third perpendicular vector is given by the cross product of the other two: **north** = **up** × **east**. Note that these vectors are defined as functions of time. At each value of t the earth motion model gives an instantaneous value for $\mathbf{r}(t)$, and that, in turn, determines the vectors **up**, **east**, and **north**.

Next, suppose that a satellite is included in the model, with instantaneous position $\mathbf{s}(t)$. The **view** vector from the antenna to the satellite is given by $\mathbf{v}(t) = [\mathbf{s}(t) - \mathbf{r}(t)]/\parallel \mathbf{s}(t) - \mathbf{r}(t) \parallel$. The goal is to calculate the direction angles α and δ for $\mathbf{v}$. Since δ measures the angle between $\mathbf{v}$ and the plane of **east** and **north**, the complimentary angle can be measured between $\mathbf{v}$ and **up**. This leads to the equation

$$\sin\delta = \mathbf{up} \cdot \mathbf{v}.$$

The angle α is found from

$$v_n = \mathbf{v} \cdot \mathbf{north}$$
$$v_e = \mathbf{v} \cdot \mathbf{east}$$

according to the equations

$$\cos\alpha = \frac{v_n}{\sqrt{v_n^2 + v_e^2}}$$
$$\sin\alpha = \frac{v_e}{\sqrt{v_n^2 + v_e^2}}.$$

These follow from the fact that the projection of $\mathbf{v}$ into the local xy plane is given by v_e **east** + v_n **north**.

In this example, direction angles play a role in several optimization problems. First, it may be of interest to predict the maximum value of δ as a satellite passes over the tracking station. This maximum value of elevation is an indication of how close the satellite comes to passing directly overhead, and may be used to determine whether communication will be possible between satellite and tracking station.

Additional optimization problems concern the derivatives of α and δ. In many designs, an antenna can turn about horizontal and vertical axes to point the center of the field of view in a particular direction. In order to stay pointed at a passing satellite, the antenna must be rotated on its axes so as to match the motion of the satellite, and α and δ specify exactly how far the antenna must be rotated about each axis at each time. However, there are mechanical limits on how fast the antenna can turn and accelerate. For this reason, during the time that the satellite is in view, the maximum values of the first and second derivatives of α and δ are of interest. If the first derivatives exceed the antenna's maximum turning speed, or if the second derivatives exceed the antenna's maximum acceleration, the antenna will not be able to remain pointed at the satellite.

Design Parameter Optimization

The preceding examples all involve simple kinds of optimization problems with objective functions depending only on time. There are also many situations in which system performance variables are optimized over some domain of design parameters. As one example of this, consider a system with a single satellite traveling in a circular orbit. Assume that the initial point of the orbit falls on the equator, with angle δ between the initial heading and the xy plane, as in Fig. 2. In this example, the object is to choose an optimal value of δ. The optimization problem includes several tracking stations on the ground that are capable of communicating with the

satellite. As it orbits, there may be times when the satellite cannot communicate with any of the tracking stations. At other times, one or more stations may be accessible. Over the simulation period, the total amount of time during which at least one tracking station is accessible will depend on the value of δ. It is this total amount of access time (denoted A) that is to be maximized.

In this problem, the objective function A is not given as a mathematical expression involving the variable δ. An appropriate simulation can be created to compute A for any particular δ of interest. This can then be used in conjunction with an optimization algorithm, with the simulation executed each time it is necessary to calculate $A(\delta)$.

The preceding example is a simple one, and the execution time required to compute $A(\delta)$ is small. For more complicated situations, each execution of the simulation can require a significant amount of time. In these cases, it may be more practical to use some sort of interpolation scheme. The idea would be to run the simulation for some values of the parameter(s), and to interpolate between these values as needed during the optimization process.

In some situations, there is a resource allocation problem that can add yet another level of complexity to optimizing system performance. For example, if there are several satellites that must compete for connection time with the various tracking stations, just determining how to assign the tracking stations to the satellites is not a simple matter. In this situation, there may be one kind of optimization problem performed during the simulation to make the resource allocations, and then a secondary optimization that considers the effect of changing system design parameters. An example of this kind of problem is described in detail in [6].

The preceding examples have been provided to illustrate the kinds of optimization problems that arise in simulations of satellite systems. Although there has been very little discussion of methods to solve these optimization problems, it should be clear that standard methods apply, especially in the cases for which the independent variable is time. In that context, the ability to compute derivatives relative to time for the objective function is of interest. In addition, it sometimes occurs that the objective function is, itself, defined as a derivative of some geometric variable, providing another motivation for computing derivatives. The next topic of discussion concerns the use of automatic differentiation for computing the desired derivatives.

Automatic Differentiation

Automatic differentiation refers to a family of techniques for automatically computing derivatives as a byproduct of function evaluation. A survey of different approaches and applications can be found in [5] and in-depth treatment appears in [4]. For the present discussion, attention will be restricted to what is called the *forward mode of automatic differentiation*, and in particular, the approach described in [8]. In this approach, to provide automatic calculation of the first m derivatives of real valued expressions of a single variable x, an algebraic system is defined consisting of real $m+1$ tuples, to which are extended the familiar binary operations and elementary functions generally defined on real variables. For concreteness, m will be assumed to be 3 below, but the discussion can be generalized to other values in an obvious way.

With $m = 3$, the objects manipulated by the automatic differentiation system are 4-tuples. The idea is that each 4-tuple represents the value of a function and its first 3 derivatives, and that the operations on tuples preserve this interpretation. Thus, if $a = (a_0, a_1, a_2, a_3)$ consists of the value of $f(t)$, $f'(t)$, $f''(t)$, and $f'''(t)$ at some t, and if $b = (b_0, b_1, b_2, b_3)$ is similarly defined for function g, then the product ab that is defined for the automatic differentiation system will consist of the value at t of fg and its first 3 derivatives. Similarly, the extension of the squareroot function to 4-tuples is so contrived that $\sqrt{a}$ will consist of the value of $\sqrt{f(t)}$ and its first 3 derivatives.

In the preceding remarks, the functions f and g are assumed to be real valued, but similar ideas work for vector valued functions. The principle difference is this: when $f(t)$ is a vector, then so are its derivatives, and the a_i referred to above are then vectors rather than scalars. In addition, for vector valued functions, there are different operations than for scalar valued functions. For example, vector functions may be combined with a dot product, as opposed to the conventional product of real scalars, and while the squareroot operation is not defined for vector valued functions, the norm operation $\| f(t) \|$ is.

In an automatic differentiation system built along these lines, there must be some functions that are evaluated directly to produce 4-tuples. For example, the constant function with value c can be evaluated directly to produce the tuple $(c, 0, 0, 0)$, and the identity function $I(t) = t$ can be evaluated directly to produce $(t, 1, 0, 0)$. For geometric satellite system simulations, it is also convenient to provide direct evaluation of tuples for the motion models. For example, let $\mathbf{r}(t)$ be the position vector for a tracking station, as developed in equations (1)–(5). It is a simple matter to work out appropriate formulas for the first three derivatives of $\mathbf{r}(t)$, each of which is also a vector. This is included in the automatic differentiation system so that when a particular value of t is given, the motion model computes the 4-tuple $(\mathbf{r}(t), \mathbf{r}'(t), \mathbf{r}''(t), \mathbf{r}'''(t)$. A similar arrangement is made for every moving object represented in the simulation, including satellites, tracking stations, ships, aircraft, and so on.

Here is a simple example of how automatic differentiation is used. In the earlier discussion of optimization problems, there appeared the following equation:

$$\cos \angle BAC = \frac{(\mathbf{b} - \mathbf{a}) \cdot (\mathbf{c} - \mathbf{a})}{\|\mathbf{b} - \mathbf{a}\| \cdot \|\mathbf{c} - \mathbf{a}\|}.$$

Using automatic differentiation, $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ would be 4-tuples, each consisting of four vectors. These are produced by the motion models for three satellites, as the values of position and its first three derivatives at a specific time. The operations used in the equation, vector difference, dot product, and norm, as well as scalar multiplication and division, are all special modified operations that work directly on 4-tuples. The end result is also a 4-tuple, consisting of the cosine of angle BAC, as well as the first three derivatives of that function, all at the specified value of t. As a result, the programmer can obtain computed values for the derivatives of the function without explicitly coding equations for these derivatives. More generally, after defining appropriate 4-tuples for all of the motion models, the programmer automatically obtains derivatives for any function that is defined by operating on the motion models, just by defining the operations. No explicit representation of the derivatives of the operations is needed. Some details of how the system works follow.

Scalar Functions and Operations

Consider first operations which apply to scalars. There are two basic types: binary operations $(+, -, \times, \div)$ and elementary functions (squareroot, exponential and logarithm, trigonometric functions, etc.). These operations must be defined for the 4-tuples of the automatic differentiation system in such a way that derivatives are correctly propagated.

The definition for multiplication will illustrate the general approach for binary operations. Suppose that (a, b, c, d) and (u, v, w, x) are two 4-tuples of scalars. They represent values of functions and their derivatives, say, $(a, b, c, d) = (f(t), f'(t), f''(t), f'''(t))$ and $(u, v, w, x) = (g(t), g'(t), g'''(t), g'''(t))$. The product is supposed to give $((fg)(t), (fg)'(t), (fg)''(t), (fg)''(t))$. Each of these derivatives can be computed using the derivatives of f and g.

$$\begin{aligned}
(fg)(t) &= f(t)g(t),\\
(fg)'(t) &= f'(t)g(t) + f(t)g'(t),\\
(fg)''(t) &= f''(t)g(t) + 2f'(t)g'(t) + f(t)g''(t),\\
(fg)'''(t)) &= f'''(t)g(t) + 3f''(t)g'(t)\\
&\qquad + 3f'(t)g''(t) + f(t)g'''(t)).
\end{aligned}$$

On the right side of each equation, now substitute the entries of (a, b, c, d) and (u, v, w, x).

$$\begin{aligned}
(fg)(t) &= au,\\
(fg)'(t) &= av + bu,\\
(fg)''(t) &= aw + 2bv + cu,\\
(fg)'''(t)) &= ax + 3bw + 3cv + du.
\end{aligned}$$

This shows that 4-tuples must be multiplied according to the rule

$$\begin{aligned}
&(a, b, c, d)(u, v, w, x)\\
&= (au, av + bu, aw + 2bv + cu,\\
&\qquad ax + 3bw + 3cv + du).
\end{aligned}$$

For addition, subtraction, and division a similar approach can be used. All that is required is that successive derivatives of the combination of f and g be expressed in terms of the derivatives of f and g separately. Replacing these derivatives with the appropriate components of (a, b, c, d) and (u, v, w, x) produces the desired formula for operating on 4-tuples.

To define the operation on a 4-tuple of an elementary function, a similar approach will work. Consider defining how a function h should apply to a 4-tuple $(a, b, c, d) = (f(t), f'(t), f''(t), f'''(t))$. This time, the desired end result should contain derivatives for the composite function $h \circ f$, and so should have the form $((h \circ f)(t), (h \circ f)'(t), (h \circ f)''(t), (h \circ f)'''(t))$ The derivative of $h \circ f$ is given by $h'(f(t))\, f'(t)$, which becomes $h'(a)\, b$ after substitution. Similar computations produce expressions for the second and third derivatives:

$$\begin{aligned}&(h \circ f)''(t)\\ &= h''(f(t))f'(t)^2 + h'(f(t))f''(t)\\ &= h''(a)b^2 + h'(a)c\end{aligned}$$

and

$$\begin{aligned}&(h \circ f)'''(t)\\ &= h'''(f(t))f'(t)^3 + 3h''(f(t))f'(t)f''(t)\\ &\quad + h'(f(t))f'''(t)\\ &= h'''(a)b^3 + 3h''(a)bc + h'(a)d.\end{aligned}$$

These results lead to

$$\begin{aligned}&h(a, b, c, d)\\ &= (h(a), h'(a)b, h''(a)b^2 + h'(a)c,\\ &\qquad h'''(a)b^3 + 3h''(a)bc + h'(a)d).\end{aligned}$$

As an example of how this is applied, let $h(t) = e^t$. Then $h(a) = h'(a) = h''(a) = h'''(a) = e^a$ so

$$\begin{aligned}&e^{(a,b,c,d)}\\ &= (e^a, e^a b, e^a b^2 + e^a c, e^a b^3 + 3e^a bc + e^a d)\\ &= e^a(1, b, b^2 + c, b^3 + 3bc + d).\end{aligned}$$

Other functions are a little more complicated, but the overall approach is generally correct.

The preceding discussion indicates how operations on 4-tuples would be built into an automatic differentiation system. However, the user of such a system would simply apply the operations. So, if an appropriate definition has been provided for $\Omega(t)$ as discussed earlier, along with the derivatives, the program would compute a 4-tuple for Ω and its derivatives at a particular time. Say that is represented in the program by the variable W. If the program later includes the call $\sin(W)$, the result would be a 4-tuple with values for $\sin(\Omega(t))$, and the first three derivatives.

Vector Functions and Operations

The approach for vector functions is basically the same as for scalar functions. The only modification that is needed is to recognize that the components of 4-tuples are now vectors. Because the rules for computing derivatives of vector operations are so similar to those for scalar operations, there is little difference in the appearance of the definitions. For example, here is the definition for the dot product of two 4-tuples, whose components are vectors:

$$\begin{aligned}&(a, b, c, d) \cdot (u, v, w, x)\\ &= (a \cdot u, a \cdot v + b \cdot u, a \cdot w + 2b \cdot v + c \cdot u,\\ &\qquad a \cdot x + 3b \cdot w + 3c \cdot v + d \cdot u).\end{aligned}$$

The formulation for vector cross product is virtually identical, as is the product of a scalar 4-tuple with a vector 4-tuple. For the vector norm, simply define

$$\|(a, b, c, d)\| = \sqrt{(a, b, c, d) \cdot (a, b, c, d)}.$$

Since both dot product of vector 4-tuples and square-root of scalar 4-tuples have already been defined in the automatic differentiation system, this equation will propagate derivatives correctly.

With a full complement of scalar and vector operations provided by the automatic differentiation system, all of the geometric variables discussed in previous examples can be included in a computer program, with derivatives generated automatically. As a particular case, reconsider the discussion earlier of computing elevation δ and compass angle α for a satellite as viewed from a tracking station. Assuming that r and s have been defined as 4-tuples for the vector positions of that station and satellite, the following fragment of pseudocode would carry out the computations described earlier:

up	= r/norm(r)
east	= cross(pole, up)
east	= east/norm(east)
north	= cross(up, east)
v	= (s—r)/norm(s—r)
vn	= dot(v, north)
ve	= dot(v, east)
vu	= dot(v, up)
delta	= asin(vu)
alpha	= atan2(ve, vn)

Executed in an automatic differentiation system, this code produces not just the instantaneous values of the angles α and δ, but their first three derivatives, as well. The programmer does not need to derive and code explicit equations for these derivatives, a huge savings in this problem. And all of the derivative information is useful. Recall that the first and second derivatives are of interest for their physical interpretations as angular velocities and accelerations. The third derivatives are used in finding the maximum values of the second derivatives (accelerations).

Implementation Methods

One of the simplest ways to implement automatic differentiation is to use a language like C^{++} that supports the definition of abstract data types and operator overloading. Then the automatic differentiation system would be implemented as a series of data types and operations, and included as part of the code for a simulation. A discussion of one such implementation can be found in [7].

Another approach is to develop a preprocessor that automatically augments code with the steps needed to compute derivatives. With such a system, the programmer develops code in a conventional language such as FORTRAN, with some additional features that control the application of automatic differentiation. Next, this code is operated on by the preprocessor, producing a modified program. That is then compiled and executed in the usual way. Examples of this approach can be found in [5].

Summary

Geometric models are very useful in representing the motions of satellites and terrestrial objects in simulations of satellite systems. These models are defined in terms of vector operations, which permit the convenient formulation of equations for geometric constructs such as distances and angles arising in the satellite system configuration. Equations which specify instantaneous positions in space of moving objects are a fundamental component of the geometric modeling framework.

Optimization problems occur in this framework in two guises. First, there are problems in which the objective functions are directly defined as features of the geometric setting. An example of this would be to find the minimum distance between two satellites. Second, measures of system performance are derived via simulation as a function of design parameters, and these measures are optimized by varying the parameters. An example of this kind of problem would be to seek a particular orbit geometry in order to maximize the total amount of time a satellite has available to communicate with a network of tracking stations.

Automatic differentiation is a feature of an environment for implementing simulations as computer programs. In an automatic differentiation system, the equations which define values of variables automatically produce the values of the derivatives, as well. In the geometric models of satellite systems, derivatives of some variables are of intrinsic interest as velocities and accelerations. Derivatives are also useful in solving optimization problems.

Automatic differentiation can be provided by replacing single operands with tuples, representing the operands and their derivatives. For some tuples, the derivatives must be explicitly provided. This is the case for the motion models. For tuples representing combinations of the motion models, the derivatives are generated automatically. These combinations can be defined using any of the supported operations provided by the automatic differentiation system, typically including the operations of scalar and vector arithmetic, as well as scalar functions such as exponential, logarithmic, and trigonometric functions. Languages which support abstract data types and operator overloading are a convenient setting for implementing an automatic differentiation system.

See also

- ► Automatic Differentiation: Calculation of the Hessian
- ► Automatic Differentiation: Calculation of Newton Steps
- ► Automatic Differentiation: Introduction, History and Rounding Error Estimation
- ► Automatic Differentiation: Parallel Computation
- ► Automatic Differentiation: Point and Interval
- ► Automatic Differentiation: Point and Interval Taylor Operators

► Automatic Differentiation: Root Problem and Branch Problem
► Nonlocal Sensitivity Analysis with Automatic Differentiation

References

1. Bate RR, Mueller DD, White JE (1971) Fundamentals of astrodynamics. Dover, Mineola, NY
2. Battin RH (1987) An introduction to the mathematics and methods of astrodynamics. AIAA Education Ser. Amer. Inst. Aeronautics and Astronautics, Reston, VA
3. Escobal PR (1965) Methods of orbit determination. R.E. Krieger, Huntington, NY
4. Griewank A (2000) Evaluating derivatives: Principles and techniques of algorithmic differentiation. SIAM, Philadelphia
5. Griewank A, Corliss GF (eds) (1995) Automatic differentiation of algorithms: Theory, implementation, and application. SIAM, Philadelphia
6. Kalman D (1999) Marriages made in the heavens: A practical application of existence. Math Magazine 72(2):94–103
7. Kalman D, Lindell R (1995) Automatic differentiation in astrodynamical modeling. In: Griewank A and Corliss GF (eds) Automatic differentiation of algorithms: Theory, implementation, and application. SIAM, Philadelphia, pp 228–241
8. Rall LB (Dec. 1986) The arithmetic of differentiation. Math Magazine 59(5):275–282
9. Thomas GB Jr, Finney RL (1996) Calculus and analytic geometry, 9th edn. Addison-Wesley, Reading, MA
10. Wertz JR (ed) (1978) Spacecraft attitude determination and control. Reidel, London

Automatic Differentiation: Introduction, History and Rounding Error Estimation

MASAO IRI, KOICHI KUBOTA
Chuo University, Tokyo, Japan

MSC2000: 65D25, 26A24

Article Outline

Keywords

Differentiation; System analysis; Error analysis

Introduction

Most numerical algorithms for analyzing or optimizing the performance of a nonlinear system require the partial derivatives of functions that describe a mathematical model of the system. The *automatic differentiation* (abbreviated as AD in the following), or its synonym, *computational differentiation*, is an efficient method for computing the numerical values of the derivatives. AD combines advantages of numerical computation and those of symbolic computation [2,4].

Given a vector-valued function $\mathbf{f}$: $\mathbf{R}^n \to \mathbf{R}^m$:

$$\mathbf{y} = \mathbf{f}(\mathbf{x}) \equiv \begin{pmatrix} f_1(x_1, \ldots, x_n) \\ \vdots \\ f_m(x_1, \ldots, x_n) \end{pmatrix} \tag{1}$$

of n variables represented by a big program with hundreds or thousands of program statements, one often had encountered (before the advent of AD) some difficulties in computing the partial derivatives $\partial f_i/\partial x_j$ with conventional methods (as will be shown below). Now, one can successfully differentiate them with AD, deriving from the program for $\mathbf{f}$ another program that efficiently computes the numerical values of the partial derivatives.

AD is entirely different from the well-known numerical approximation with quotients of finite differences, or *numerical differentiation*. The quotients of finite differences, such as $(f(x + h) - f(x))/h$ and $(f(x + h) - f(x - h))/2h$, approximate the derivative $f'(x)$, where truncation errors are of $O(h)$ and $O(h^2)$, respectively, but there is an insurmountable difficulty to compute better and better approximation. For, although an appropriately small value of h is chosen, it may fail to compute the values of the function when $x \pm h$ is out of the domain of f, and, furthermore, the effect of rounding errors in computing the values of the functions is of problem.

AD is also different from symbolic differentiation with a symbolic manipulator. The symbolic differentiation derives the expressions of the partial derivatives rather than the values. The mathematical model of a large scale system may be described in thousands of program statements so that it becomes very difficult to handle whole of them with an existing symbolic manipulator. (There are a few manipulators combined with AD, which can handle such large scale programs. They should be AD regarded as a symbolic manipulator.)

Example 1 Program 1 computes an output value y_1 as a composite function f_1 for given input values $x_1 = 2$, $x_2 = 3$, $x_3 = 4$:

$$y_1 = f_1(x_1, x_2, x_3) = \frac{x_1(x_2 - x_3)}{\exp(x_1(x_2 - x_3)) + 1} . \qquad (2)$$

```
IF (x2.le.x3)
  THEN y1 = x1(x2 − x3)
  ELSE y1 = x1(x2 + x3)
ENDIF
y1 = y1/(exp(y1) + 1).
```

Automatic Differentiation: Introduction, History and Rounding Error Estimation, Program 1
Example

The execution of this program is traced by a sequence of assignment statements (Program 2).

```
y1 ← x1(x2 − x3),
y1 ← y1/(exp(y1) + 1).
```

Automatic Differentiation: Introduction, History and Rounding Error Estimation, Program 2
Program 1 expanded to straight line program for the specified input values

A set of unary or binary arithmetic operators (+, −, ∗, /) and elementary transcendental functions (exp, log, sin, cos, …) that may be used in the programs will be called *basic operations*. (Some special operations such as those generating 'constant' and 'input' are also to be counted among basic operations.) Program 2 can be expanded into a sequence of assignment statements each of whose right side has only one basic operation (Program 3), where $z_1, \ldots, z_s$ are temporary variables ($s = 2$ for this example).

```
1 | z1 ← x2 − x3,
2 | z1 ← x1 ∗ z1,
3 | z2 ← exp(z1),
4 | z2 ← z2 + 1,
5 | z1 ← z1/z2.
```

Automatic Differentiation: Introduction, History and Rounding Error Estimation, Program 3
Expanded history of execution with each line having only one basic operation

Moreover, it is useful to rewrite Program 3 into a sequence of *single assignment* statements, in which each variable appears at most once in the left sides (Program 4), hence, '←' can be replaced by ' = '.

```
1 | v1 ← x2 − x3,
2 | v2 ← x1 ∗ v1,
3 | v3 ← exp(v2),
4 | v4 ← v3 + 1.,
5 | v5 ← v2/v4,
```

Automatic Differentiation: Introduction, History and Rounding Error Estimation, Program 4
Computational process

The sequence is called a *computational process*, where the additional variables $v_1, \ldots, v_5$ are called *intermediate variables* that keep the intermediate results. A graph called a *computational graph*, $G = (V, A)$, may be used to represent the process (see Fig. 1).

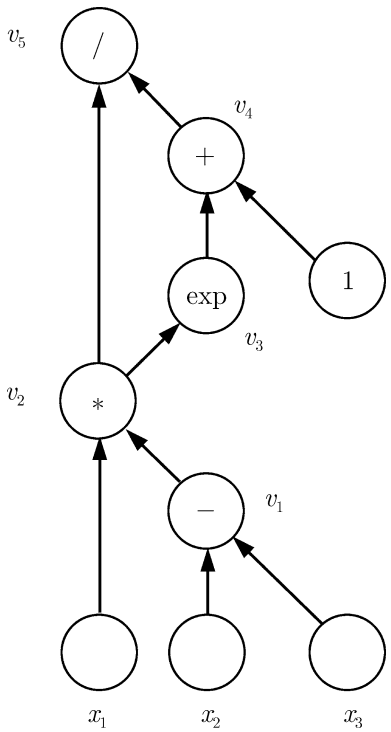
Algorithms

There are two modes for AD algorithm, *forward mode* and *reverse mode*. The forward mode is to compute $\partial y_i/\partial x_j$ ($i = 1, \ldots, m$) for a fixed j, whereas the reverse mode is to compute $\partial y_i/\partial x_j$ ($j = 1, \ldots, n$) for a fixed i.

The forward mode corresponds to tracing an expanded program such as Program 3 in the natural order. Assume that execution of the kth assignment in the program is represented as

$$z_c \leftarrow \psi_k(z_a, z_b) . \qquad (3)$$

When the values of both $\partial z_a/\partial x_j$ and $\partial z_b/\partial x_j$ are known, $\partial z_c/\partial x_j$ can be computed by applying the chain rule of



Automatic Differentiation: Introduction, History and Rounding Error Estimation, Figure 1
Computational graph

differentiation to (3):

$$\frac{\partial z_c}{\partial x_j} \leftarrow \frac{\partial \psi_k}{\partial z_a}\frac{\partial z_a}{\partial x_j} + \frac{\partial \psi_k}{\partial z_b}\frac{\partial z_b}{\partial x_j}\,. \tag{4}$$

$\partial\psi_k/\partial z_a$ and $\partial\psi_k/\partial z_b$ are called *elementary partial derivatives*, and are computed by Table 1 for various ψ_k.

Introducing new variables $\overline{z}_1, \ldots, \overline{z}_s$, $\overline{x}_1, \ldots, \overline{x}_n$ corresponding to $\partial z_1/\partial x_j$, …, $\partial z_s/\partial x_j$, $\partial x_1/\partial x_j$, …, $\partial x_n/\partial x_j$, respectively, and initializing $\overline{x}_k \leftarrow 0$ $(1 \le k \le n, k \ne j)$ and $\overline{x}_j \leftarrow 1$, we may express (4) as

$$\overline{z}_c \leftarrow \frac{\partial \psi_k}{\partial z_a}\overline{z}_a + \frac{\partial \psi_k}{\partial z_b}\overline{z}_b\,. \tag{5}$$

Thus, we can write down the whole program for the forward mode as shown in Program 5.

The reverse mode corresponds to tracing a computational process such as Program 4 backwards. The kth *computational step*, i. e., execution of the kth assignment in the program, can be written in general as

$$v_k = \psi_k(u_{k1}, u_{k2})|_{u_{k1}=v_{\alpha_k}, u_{k2}=v_{\beta_k}}\,, \tag{6}$$

Automatic Differentiation: Introduction, History and Rounding Error Estimation, Table 1
Elementary partial derivatives

$z_c = \psi(z_a, z_b)$	$\frac{\vartheta\psi}{\vartheta z_a}$	$\frac{\vartheta\psi}{\vartheta z_b}$
$z_c = z_a \pm z_b$	1	± 1
$z_c = z_a \cdot z_b$	z_b	z_a
$z_c = z_a/z_b$	$1/z_b$	$-z_a/z_b^2$ $(= -z_c/z_b)$
$z_c = \sqrt{z_a}$	$\frac{1}{2}/\sqrt{z_a} (= \frac{1}{2}/z_c)$	–
$z_c = \log(z_a)$	$1/z_a$	–
$z_c = \exp(z_a)$	$\exp(z_a) (= z_c)$	–
$z_c = \cos(z_a)$	$-\sin(z_a)$	–
$z_c = \sin(z_a)$	$\cos(z_a)$	–
⋮	⋮	⋮

	Initialization
	$\overline{x}_j \leftarrow 1$,
	$\overline{x} \leftarrow 0$ $(1 \le k \le n,\ k \ne j)$,
	Forward algorithm:
1	$z_1 \leftarrow x_2 - x_3$,
1′	$\overline{z}_1 \leftarrow 1 * \overline{x}_2 - 1 * \overline{x}_3$,
2′	$\overline{z}_1 \leftarrow z_1 * \overline{x}_1 + x_1 * \overline{z}_1$,
2	$z_1 \leftarrow x_1 * z_1$,
3	$z_2 \leftarrow \exp(z_1)$,
3′	$\overline{z}_2 \leftarrow z_2 * \overline{z}_1$,
4	$z_2 \leftarrow z_2 + 1$,
4′	$\overline{z}_2 \leftarrow 1 * \overline{z}_2$,
5	$z_1 \leftarrow z_1/z_2$,
5′	$\overline{z}_1 \leftarrow (1/z_2) * \overline{z}_1 - (z_1/z_2) * \overline{z}_2$

Automatic Differentiation: Introduction, History and Rounding Error Estimation, Program 5
Forward mode program for differentation

where $u_{k,1}$ and $u_{k,2}$ are formal parameters, v_{α_k} and v_{β_k} are real parameters representing some of $x_1, \ldots, x_n, v_1, \ldots, v_{k-1}$. If ψ_k is unary, $u_{k,2}$ and v_{β_k} are omitted. Let r be the total number of computational steps. In Program 4, we have $r = 5$ and, for $k = 2$, e. g., $\psi_2 =$ '$*$', $v_{\alpha_2} = x_1$ and $v_{\beta_2} = v_1$.

The total differentiation of (6) yields the relations among $dx_1, \ldots, dx_n, dv_1, \ldots, dv_r$ such as follows:

$$\mathrm{d}v_k = \frac{\partial \psi_k}{\partial u_{k,1}}\mathrm{d}v_{\alpha_k} + \frac{\partial \psi_k}{\partial u_{k,2}}\mathrm{d}v_{\beta_k} \quad (k = 1, \ldots, r)\,. \tag{7}$$

The computation of the partial derivatives of the ith component of the final result $y_i = f_i(x_1, \ldots, x_n)$ in (1)

with respect to $x_1, \ldots, x_n$ is that of the coefficients of the relation among $dx_1, \ldots, dx_n$ and dy_i.

Here, new variables $\overline{x}_1, \ldots, \overline{x}_n, \overline{v}_1, \ldots, \overline{v}_r$ are introduced for the computation of those coefficients. Without loss of generality, we may assume that the value of y_i is computed at v_r. After Program 4 is executed in the natural order with all the information on intermediate results preserved, these new variables are initialized as $\overline{x}_j \leftarrow 0$ $(j = 1, \ldots, n)$, $\overline{v}_k \leftarrow 0$ $(k = 1, \ldots, r-1)$ and $\overline{v}_r \leftarrow 1$, then the relation

$$\mathrm{d}y = \sum_{j=1}^{n} \overline{x}_j \mathrm{d}x_j + \sum_{k=1}^{r} \overline{v}_k \mathrm{d}v_k \tag{8}$$

holds. Secondly, $dv_r, dv_{r-1}, \ldots, dv_k$ can be eliminated from (8) in this order by modifying

$$\overline{v}_{\alpha_k} \leftarrow \overline{v}_{\alpha_k} + \overline{v}_k \frac{\partial \psi_k}{\partial v_{\alpha_k}}, \tag{9}$$

$$\overline{v}_{\beta_k} \leftarrow \overline{v}_{\beta_k} + \overline{v}_k \frac{\partial \psi_k}{\partial v_{\beta_k}}. \tag{10}$$

Finally, if we change k in the reverse order, i. e., $k = r$, $r-1$, …, 1, we can successfully eliminate all the dv_k $(k = 1, \ldots, r)$ to have

$$\mathrm{d}y = \sum_{j=1}^{n} \overline{x}_j \mathrm{d}x_j . \tag{11}$$

The final coefficient $\overline{x}_j$ indicates the value of $\partial f_i / \partial x_j$ $(j = 1, \ldots, n)$. Program 6 in which modifications (9) and (10) are embedded is the reverse mode program, which is sometimes called the *adjoint program* of Program 4.

It is easy to extend the algorithms for computing a linear combination of the column vectors of the Jacobian matrix J with the forward mode, and a linear combination of the row vectors of J with the reverse mode.

Complexity

It is proved that, for a constant C ($= 4 \sim 6$, varying under different computational models), the total operation count for $\partial y_i / \partial x_j$'s with a fixed j in the forward mode algorithm, as well as that for $\partial y_i / \partial x_j$'s with a fixed i in the reverse mode algorithm, is at most $C \cdot r$, i. e., in $O(r)$. Roughly speaking, r is proportional to the execution time T of the given program, so that the time complexity is in $O(T)$. Furthermore, we have to repeat such computation n times to get all the required partial

	Forward sweep:
	(insert Program 4 here)
	Initialization: $(n = 3, r = 5)$
	$\overline{x}_j \leftarrow 0$ $(j = 1, \ldots, n)$,
	$\overline{v}_k \leftarrow 0$ $(k = 1, \ldots, r-1)$,
	$\overline{v}_r \leftarrow 1$,
	Reverse elimination:
$5''$	$\overline{v}_2 \leftarrow \overline{v}_2 + (1/v_4) * \overline{v}_5$,
	$\overline{v}_4 \leftarrow \overline{v}_4 + (-v_5/v_4) * \overline{v}_5$,
$4''$	$\overline{v}_3 \leftarrow \overline{v}_3 + 1 * \overline{v}_4$,
$3''$	$\overline{v}_2 \leftarrow \overline{v}_2 + v_3 * \overline{v}_3$,
$2''$	$\overline{x}_1 \leftarrow \overline{x}_1 + v_1 * \overline{v}_2$,
	$\overline{v}_1 \leftarrow \overline{v}_1 + x_1 * \overline{v}_2$,
$1''$	$\overline{x}_2 \leftarrow \overline{x}_2 + 1 * \overline{v}_1$,
	$\overline{x}_3 \leftarrow \overline{x}_3 + (-1) * \overline{v}_1$.

Automatic Differentiation: Introduction, History and Rounding Error Estimation, Program 6
Reverse mode program

derivatives by the forward mode, and m times by the reverse mode. What should be noted here is that the computational time of the forward or reverse mode algorithm for one set of derivatives does not depend on m or n but only on r.

Denoting the spatial complexity of the original program by S, that of the forward mode algorithm is in $O(S)$. However, the spatial complexity of the reverse mode is in $O(T)$, since the reverse mode requires a history of the forward sweep recorded in storage whose size is in $O(T)$.

A rough sketch of the proof is as follows. Without loss of generality, assume that the given program is expanded into a sequence of single assignment statements with a binary or unary basic operation as shown in Program 3 and 4. The operation count for computing the elementary partial derivatives (Table 1) is bounded by a constant. The additional operation count for modifying $\overline{v}_k$'s and $\overline{x}_j$'s in (5), (9) and (10) is also bounded since there are at most two additions and two multiplications. There are r operations in the original program, so that the total operation count in the forward mode algorithm as well as that in the reverse mode algorithm is in $O(r)$.

Note that the computational complexities of the forward mode and the reverse mode may not be optimal, but at least one can compute them in time proportional

to that for the computation of the given original program.

One can extend the AD algorithms to compute higher derivatives. In particular, it is well known how to compute a truncated Taylor series to get arbitrarily higher-order derivatives of a function with one variable [14]. One may regard a special function such as a Bessel function or a block of several arithmetic operations, such as the inner product of vectors, as a basic operation if the corresponding elementary partial derivatives are given with computational definitions. An analogy is pointed out in [7] between the algorithms for the partial derivatives and those of the computation of the shortest paths in an acyclic graph.

It has also been pointed out that there may be pitfalls in the derived program with AD. For example, a tricky program

IF (x.ne.1.0)
THEN $y = x*x$
ELSE $y = 1.0 + (x - 1.0) * b$
ENDIF

can compute the value of a function $f(x) = x^2$ correctly for all x. However, the derived program fails to compute $f'(1.0)$, because the differentiation of the second assignment with respect to x is not 2.0 but b. Thus conditional branches (or equations equivalent to conditional branches) should be carefully dealt with.

History

A brief history of AD is as follows. There were not a few researchers in the world who had more or less independently proposed essentially the same algorithms.

The first publication on the forward mode algorithm was presumably the paper by R.E. Wengert in 1964 [16]. After 15 years, books were published by L.B. Rall [14] and by H. Kagiwada et al. [9] which have been influential on the numerical-computational circle. The practical and famous software system for the forward mode automatic differentiation was Pascal-SC, and its descendants Pascal-XSC and C-XSC are popular now.

The paper [13] might be the first to propose systematically the reverse mode algorithm. But there are many ways through which to approach the reverse mode algorithm. In fact, it is related to Lagrange multipliers, error analysis, generation of adjoint systems, reduction of computational complexity of computing the gradient, neural networks, etc. Of course, the principles of the derived algorithms are the same. Some remarkable works on the reverse mode algorithm had been done by S. Linnainmaa [11] and W. Miller and C. Wrathall [12] from the viewpoint of the error analysis, by W. Baur and V. Strassen [1] from that of complexity, and by P.J. Werbos [17] from that of the optimization of neural networks. A practical program had been developed by B. Speelpenning in 1980 [15] and it was rewritten into Fortran by K.E. Hillstrom in 1985 (now registered in Netlib [5,6]).

Two proceedings of the international workshops held in 1991 and 1996 collect all the theories, techniques, practical programs, current works, and future problems as well as history on automatic differentiation [2,4]. It should be noted that, in 1992, A. Griewank proposed a drastic improvement of the reverse mode algorithm using the so-called checkpointing technique. He succeeded in reducing the order of the size of storage required for the reverse mode algorithm [3]. Several software tools for automatic differentiation have been developed and popular in the world, e. g., ADIC, ADIFOR, ADMIT-1, ADOL-C, ADOL-F, FADBAD, GRESS, Odyssée, PADRE2, TAMC, etc. (See [2,4].)

Estimates of Rounding Errors

In order to solve practical real-world problems, the approximation with floating-point numbers is inevitable so that it is important to analyze and estimate the accumulated rounding errors in a big numerical computation. Moreover, in terms of estimates of the accumulated rounding errors, one can define a normalized (or weighted) norm for a numerically computed vector, that is useful for checking whether the computed vector can be regarded as zero or not from the viewpoint of numerical computation [8].

For the previous example, let us denote as δ_k the rounding error generated at the execution of the basic operation to compute the value of v_k. Then, the rounding errors in the example is explicitly written:

$$
\begin{array}{l|l}
1 & \widetilde{v}_1 = \widetilde{x}_2 - \widetilde{x}_3 + \delta_1, \\
2 & \widetilde{v}_2 = \widetilde{x}_1 * \widetilde{v}_1 + \delta_2, \\
3 & \widetilde{v}_3 = \exp(\widetilde{v}_2) + \delta_3, \\
4 & \widetilde{v}_4 = \widetilde{v}_3 + 1 + \delta_4, \\
5 & \widetilde{v}_5 = \widetilde{v}_2 / \widetilde{v}_4 + \delta_5.
\end{array}
$$

Here, $\widetilde{v}_k$ is the value with accumulated rounding errors.

Defining a function $\widetilde{f}$ as

$$\widetilde{f}(x_1, x_2, x_3; \delta_1, \delta_2, \delta_3, \delta_4, \delta_5) = \frac{x_1(x_2 - x_3 + \delta_1) + \delta_2}{\exp(x_1(x_2 - x_3 + \delta_1) + \delta_2) + \delta_3 + 1 + \delta_4} + \delta_5,$$

one has

$$\widetilde{v}_5 = \widetilde{f}(x_1, x_2, x_3; \delta_1, \ldots, \delta_5),$$
$$v_5 = \widetilde{f}(x_1, x_2, x_3; 0, \ldots, 0) .$$

Here, $\widetilde{v}_5 - v_5$ is the accumulated rounding error in the function value. For $v_5 = v_2/v_4 = \varphi_5(v_2, v_4)$, one has

$$\widetilde{v}_5 - v_5 = \varphi_5(\widetilde{v}_2, \widetilde{v}_4) - \varphi_5(v_2, v_4) + \delta_5 = \frac{\partial \varphi_5}{\partial v_2}(\xi_2, \xi_4) \cdot (\widetilde{v}_2 - v_2) + \frac{\partial \varphi_5}{\partial v_4}(\xi_2, \xi_4) \cdot (\widetilde{v}_4 - v_4) + \delta_5 ,$$

where $\xi_2 = \theta'\widetilde{v}_2 + (1-\theta')v_2$ and $\xi_4 = \theta''\widetilde{v}_4 + (1-\theta'')v_4$ for $0 < \theta', \theta'' < 1$. Expanding $\widetilde{v}_2 - v_2$ and $\widetilde{v}_4 - v_4$ similarly and expanding the other intermediate variables sequentially, the approximation:

$$\widetilde{v}_5 - v_5 \simeq \sum_{k=1}^{5} \frac{\partial \widetilde{f}}{\partial \delta_k} \delta_k \tag{12}$$

is derived [10]. Note that $\frac{\partial \widetilde{f}}{\partial \delta_k}$ are computed as $\overline{v}_k$ in Program 6, which are the final results of (9) and (10).

The locally generated rounding error δ_k for the floating-point number system is bounded by

$$|\delta_k| \le c \cdot |v_k| \cdot \varepsilon_M , \tag{13}$$

where ε_M indicates so-called 'machine epsilon' and $c = 1$ may be adopted for arithmetic operations according to IEEE754 standard. Then $\Delta[f]_{\mathrm{A}}$, called *absolute estimation*, is defined by

$$\Delta[f]_{\mathrm{A}} \equiv \sum_{k=1}^{r} \left| \frac{\partial \widetilde{f}}{\partial \delta_k} \right| \cdot |v_k| \cdot \varepsilon_M , \tag{14}$$

which is an upper bound on the accumulated rounding error. Regarding the locally generated errors δ_k's as pseudo-probabilistic variables uniformly distributed over $[-|v_k|\,\varepsilon_M, |v_k|\varepsilon_M]$'s, $\Delta[f]_{\mathrm{P}}$, called *probabilistic estimate*, is defined by

$$\Delta[f]_{\mathrm{P}} \equiv \varepsilon_M \sqrt{\frac{1}{3} \sum_{k=1}^{r} \left(\frac{\partial \widetilde{f}}{\partial \delta_k} \cdot v_k \right)^2} . \tag{15}$$

There are several reports in which these estimates give quite good approximations to the actual accumulated rounding errors [8].

Moreover, one could answer the problem how to choose a norm for measuring the size of numerically computed vector. By means of the estimates of the rounding errors, a weighted norm of a vector $\mathbf{f} = [f_1, \ldots, f_m]$ whose components are numerically computed is defined by

$$||\mathbf{f}||_{\mathrm{N}} \equiv \left\| \left[\frac{f_1}{\Delta[f_1]_{\mathrm{A}}}, \ldots, \frac{f_m}{\Delta[f_m]_{\mathrm{A}}} \right] \right\|_p , \tag{16}$$

($p = 1,2$ or ∞). This weighted norm is called *normalized norm*, because it is normalized with respect to accumulated rounding errors. With this normalized norm, one can determine whether a computed vector approaches to zero or not in reference to the rounding errors accumulated in the components. Note that, since all the components of the vector are divided by the estimates of accumulated rounding errors, they have no physical dimension. The normalized norm may be used effectively as stopping criteria for iterative methods like the Newton–Raphson method.

See also

- Automatic Differentiation: Calculation of the Hessian
- Automatic Differentiation: Calculation of Newton Steps
- Automatic Differentiation: Geometry of Satellites and Tracking Stations
- Automatic Differentiation: Parallel Computation
- Automatic Differentiation: Point and Interval
- Automatic Differentiation: Point and Interval Taylor Operators

- ▶ Automatic Differentiation: Root Problem and Branch Problem
- ▶ Nonlocal Sensitivity Analysis with Automatic Differentiation

References

1. Baur W, Strassen V (1983) The complexity of partial derivatives. Theor Comput Sci 22:317–330
2. Berz M, Bischof C, Corliss G, Griewank A (eds) (1996) Computational differentiation: Techniques, applications, and tools. SIAM, Philadelphia
3. Griewank A (1992) Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. Optim Methods Soft 1:35–54
4. Griewank A, Corliss GF (eds) (1991) Automatic differentiation of algorithms: Theory, implementation, and application. SIAM, Philadelphia
5. Hillstrom KE (1985) Installation guide for JAKEF. Techn Memorandum Math and Computer Sci Div Argonne Nat Lab ANL/MCS-TM-17
6. Hillstrom KE (1985) User guide for JAKEF. Techn Memorandum Math and Computer Sci Div Argonne Nat Lab ANL/MCS-TM-16
7. Iri M (1984) Simultaneous computation of functions, partial derivatives and estimates of rounding errors – Complexity and practicality. Japan J Appl Math 1:223–252
8. Iri M, Tsuchiya T, Hoshi M (1988) Automatic computation of partial derivatives and rounding error estimates with applications to large-scale systems of nonlinear equations. J Comput Appl Math 24:365–392
9. Kagiwada H, Kalaba R, Rasakhoo N, Spingarn K (1986) Numerical derivatives and nonlinear analysis. Math. Concepts and Methods in Sci. and Engin., vol 31. Plenum, New York
10. Kubota K, Iri M (1991) Estimates of rounding errors with fast automatic differentiation and interval analysis. J Inform Process 14:508–515
11. Linnainmaa S (1976) Taylor expansion of the accumulated rounding error. BIT 16:146–160
12. Miller W, Wrathall C (1980) Software for roundoff analysis of matrix algorithms. Acad Press, New York
13. Ostrovskii GM, Wolin JM, Borisov WW (1971) Über die Berechnung von Ableitungen. Wiss Z Techn Hochschule Chemie 13:382–384
14. Rall LB (1981) Automatic differentiation – Techniques and applications. Lecture Notes Computer Science, vol 120. Springer, Berlin
15. Speelpenning B (1980) Compiling fast partial derivatives of functions given by algorithms. Report Dept Computer Sci Univ Illinois UIUCDCS-R-80-1002
16. Wengert RE (1964) A simple automatic derivative evaluation program. Comm ACM 7:463–464
17. Werbos P (1974) Beyond regression: New tools for prediction and analysis in the behavioral sciences. PhD Thesis, Appl. Math. Harvard University

Automatic Differentiation: Parallel Computation

CHRISTIAN H. BISCHOF[1], PAUL D. HOVLAND[2]

[1] Institute Sci. Computing, University Technol., Flachen, Germany

[2] Math. and Computer Sci. Div., Argonne National Lab., Argonne, USA

MSC2000: 65Y05, 68N20, 49-04

Article Outline

Keywords

Automatic differentiation; Parallel computing; MPI

Research in the field of *automatic differentiation* (AD) has blossomed since A. Griewank's paper [15] in 1989 and the Breckenridge conference [17] in 1991. During that same period, the power and availability of parallel machines have increased dramatically. A natural consequence of these developments has been research on the interplay between AD and *parallel computations*. This relationship can take one of two forms. One can examine how AD can be applied to existing parallel programs. Alternatively, one can consider how AD introduces new potential for parallelism into existing sequential programs.

Background

Automatic differentiation relies upon the fact that all programming languages are based on a finite number of elementary functions. By providing rules for the differentiation of these elementary functions, and by com-

bining these elementary derivatives according to the chain rule of differential calculus, an AD system can differentiate arbitrarily complex functions. The chain rule is associative—partial derivatives can be combined in any order. The *forward mode of AD* combines the partial derivatives in the order of evaluation of the elementary functions to which they correspond. The *reverse mode* combines them in the reverse order. For systems with a large ratio of dependent to independent variables, the reverse mode offers lower operation counts, at the cost of increased storage costs [15].

The forward and the reverse mode are the extreme ends of a wide algorithmic spectrum of accumulating derivatives. Recently, hybrid approaches have been developed which combine the forward and the reverse mode [5,10], or apply them in a hierarchical fashion [8,25]. In addition, efficient *checkpointing* schemes have been developed which address the potential storage explosion of the reverse mode by judicious recomputation of intermediate states [16,19]. Viewing the problem of automatic differentiation as an edge elimination problem on the program graph corresponding to a particular code, one can in fact show that the problem of computing derivatives with minimum cost is *NP*-hard [21]. The development of more efficient heuristics is an area of active research (see, for example, several of the papers in [3]).

Implementation Approaches

Automatic differentiation is a particular instantiation of a rule-based semantic transformation process. That is, whenever a floating-point variable changes, an associated derivative object must be updated according to the chain rule of differential calculus. For example, in the forward mode of AD, a derivative object carries the partial derivative(s) of an associated variable with respect to the independent variable(s). In the reverse mode of AD, a derivative object carries the partial derivative(s) of the dependent variable(s) with respect to an associated variable. Thus, any AD tool must provide an instantiation of a 'derivative object', maintain the association between an original variable and its derivative object, and update derivative objects in a timely fashion.

Typically AD is implemented in one of two ways: operator overloading or source transformation. In languages that allow operator overloading, such as C++ and Fortran90, each elementary function can be redefined so that in addition to the normal function, derivatives are computed as well, and either saved for later use or propagated by the chain rule. A simple class definition using the forward mode might be implemented as follows:

```
class adouble{
private:
  double value, grad[GRAD_LENGTH];
public:
  /* constructors omitted */
  friend adouble operator*(const
    adouble &, const adouble &);
  /* similar decs for other ops */
}
adouble operator*(const adouble &g1,
  const adouble &g2){
    int i;
    double newgrad[GRAD_LENGTH];
    for (i=0; i<GRAD_LENGTH;i++){
      newgrad[i] =
         (g1.value)*(g2.grad[i])+
         (g2.value)*(g1.grad[i]);
    }
    return adouble(g1.value*g2.value,
      newgrad);
}
```

An example of how this class could be used is given below.

In languages that do not support operator overloading, it can be faked by manually or automatically replacing operators such as + and ∗ with calls to subroutines.

```
main(){
  double temp[GRAD_LENGTH];
  adouble y;

  /* initialize x1 to (3.0,[1.0  0.0]),
                x2 to (4.0,[0.0 1.0])*/
  temp[0] = 1.0; temp[1] = 0.0;
  adouble  *x1 = new adouble(3.0,temp);
  temp[0] = 0.0; temp[1] = 1.0;
  adouble  *x2 = new adouble(4.0,temp);

   y = (*x1)*(x2);

  /* output (y,[dy/dx1 dy/dx2])  */
  cout << y;
  /* prints (12.0,[4.0 3.0]) */
}
```

As an alternative to operator overloading, a preprocessor can be used to transform source code for computing the function into source code for computing the function and its derivatives. This approach relies heavily on compiler technology and typically involves a combination of in-lining and subroutine calls to implement the propagation of derivatives. An example of ADIFOR-generated code (edited for clarity) invoking the SparsLinC library for transparent exploitation of *sparsity* [6] follows.

```
c derivation code for f=x*y/z

c preaccumulate partial derivates
      temp1 = x*y/z
      temp2 = 1.0/z
      temp3 = temp2*y
      temp4 = temp2*x
      temp5 = -temp1/z

c propagate derivatives
c (g_x, ... , g_f may by sparse)
      call sspg3q(g_f,temp3,g_x,temp4,
     +            g_y,temp5,g_z)

      f=temp1
```

The advantage of this approach is that it allows the exploitation of computational context in deciding how to propagate derivatives. For example, a recently developed *Hessian* module [1], adaptively determines the best strategy for each assignment statement in the code based on a machine-specific performance model for the implementation kernels employed.

A comparison of these two implementation approaches is provided in [9]. This paper also introduces an implementation design that separates the core issues of automatic differentiation from language-specific issues through the use of an interface layer called AIF (*AD intermediate form*), thus arriving at a system design that allows reuse of differentiation components across front-ends for different languages. Long-term, such a system design also allows the exploitation of the best features of both source transformation and operator overloading.

Current AD tools based on operator overloading include ADOL-C [18] and ADOL-F [29], both of which offer the option of using either the forward or the reverse mode, and to compute derivatives of arbitrary order Source transformation tools that use mostly the forward mode to provide first- and second order derivatives include ADIC [9] and ADIFOR [6]. The Odyssee [28] and TAMC [14] tools use the reverse mode in a source transformation context to provide first order derivatives. A more comprehensive survey of AD tools can be found at the website [31].

AD of Parallel Programs

In 1994, R.L. Hinkins reported on the application of AD to magnetic field calculations implemented in the data parallel languages MPFortran (MasPar Fortran) and CMFortran [22]. In 1997, P. Hovland addressed the larger issue of AD of parallel programs in general, paying close attention to message-passing parallel programs [23], but also considering other parallel programming paradigms, and A. Carle developed ADIFOR-MP, a prototype tool supporting a subset of MPI [30] and PVM [13] constructs. The focus on parallel programs employing a message-passing paradigm can be attributed to the popularity of this parallel programming paradigm and its relevance to all parallel programs targeting nonuniform memory access (NUMA) machines.

Correct AD of message-passing parallel programs requires that we maintain an association between a variable and its derivative object. In particular, when a variable is sent from one processor to another via a message, we must also send the associated derivative object. There are two ways of accomplishing this goal — we can pack the variable and derivative object together in one message or send two separate messages. Packing a variable and its associated derivative object into a single message may incur a copying overhead. On the other hand, sending separate messages requires a mechanism for associating the messages with one another at the receiving end and will increase delivery time on high-latency systems. In general, it is preferable to pack the variable and derivative object together in one message [24], minimizing copying cost through judiciously chosen derivative data structures. Other issues in ensuring correct AD of parallel programs include proper handling of nondeterminism, reduction operations at points of nondifferentiability, and seed matrix initialization [23].

In many instances, only a subset of the program input- and output variables is considered as indepen-

dent or dependent variables with respect to differentiation. An optimization technique that tries to exploit this fact is activity analysis, which seeks to reduce time and storage costs by identifying variables that do not lie on the computational path from independent to dependent variables. Such variables are termed passive and do not require an associated derivative object. Activity analysis depends on sophisticated compiler technology, namely interprocedural dataflow analysis. In message-passing parallel programs, sends and receives greatly complicate such an analysis. As the analysis needs to guarantee correctness, this fact leads to much more conservative assumptions, and as a result much optimization potential may be lost. Among the available options to circumvent this situation are user annotations, runtime analysis, or the use of a higher-level language such as HPF [26]. These issues are investigated in more detail in [23].

Another issue arising in the parallel setting is the computation of partial derivatives of new elementary functions, such as parallel *reduction operations*. For most of the common reduction operations, such as sum, maximum, and minimum, computing the partial derivatives is trivial. For the product reduction, the situation is more complex. The partial derivative of $y = \prod_{i=1}^{n} x_i$ with respect to x_i is $\partial y / \partial x_i = (\prod_{j=1}^{i-1} x_j)(\prod_{k=i+1}^{n} x_k)$. These partial derivatives can be computed using a parallel prefix and reverse parallel prefix operation. However, propagating the partial derivatives requires an additional sum reduction. We could instead combine the partial derivative computation and propagation into a single reduction. This increases the computational cost, but reduces the communication cost. In [24], Hovland and C. Bischof discuss the conditions under which each approach should be preferred and give experimental results to support the theory.

AD-Enabled Parallelism

As early as 1991, Bischof considered the problem of parallelizing the computation of derivatives computed via AD [4] to distribute the additional work introduced by AD. Applying AD to a program introduces two basic types of parallelism: data parallelism and time parallelism.

Data Parallelism

The potential for data parallelism arises whenever there are multiple independent variables (for the forward mode) or multiple dependent variables (for the reverse mode). Different processes can be employed to propagate partial derivatives with respect to a subset of the independent variables in parallel.

Such an implementation is feasible if one can employ light-weight threads for the parallel derivative computation. A limiting factor is the fact that the derivative computations are interspersed with the function computation. Thus, an alternative approach is to replicate the sequential computation on each processor, thereby virtually eliminating communication costs. This approach has proven effective for computations involving a large number of independent variables [7,32].

Time Parallelism

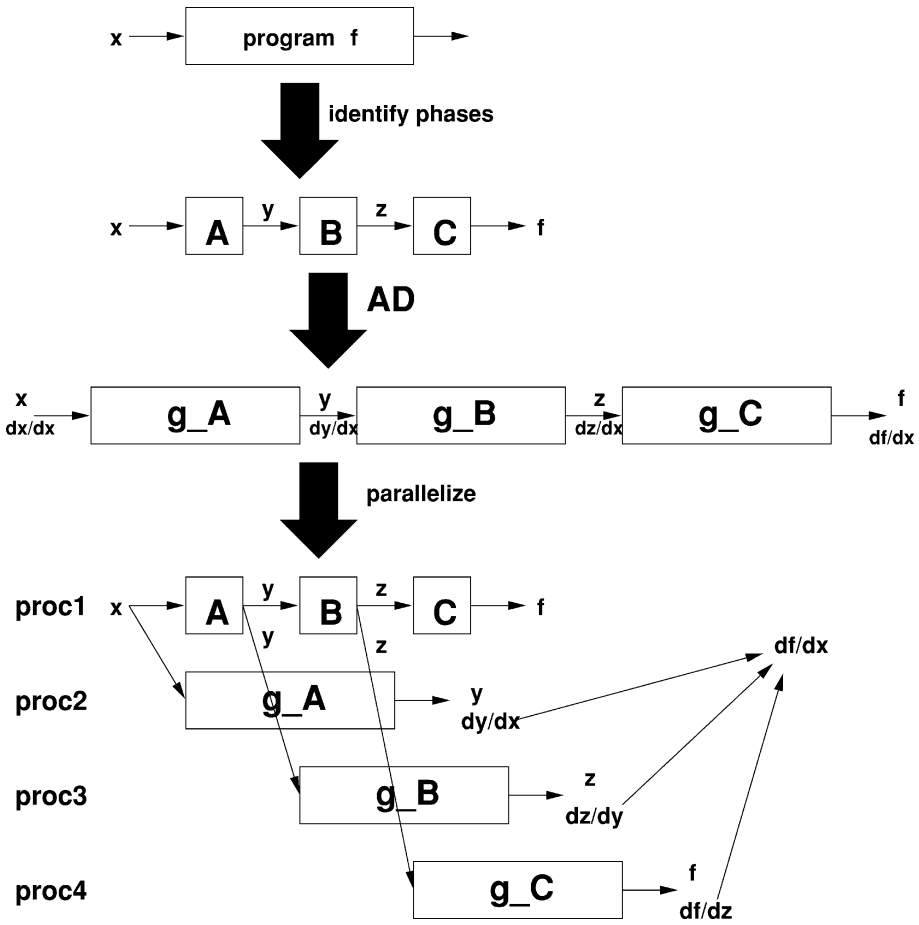
Time parallelism arises as a consequence of the associativity of the chain rule. By breaking the computation into several phases, we can compute and propagate partial derivatives over each phase simultaneously, then combine the results according to the chain rule. This approach is illustrated in Fig. 1. Before each phase, a derivative computation for that phase is forked off, using as input the results of the previous phase. At the conclusion of the derivative computations, the partial derivatives are combined according to the chain rule.

This illustration assumes the forward mode. If we were using the reverse mode, the derivative computation for phase A would be forked off after phase A had completed. The effectiveness of this approach has been demonstrated for both the forward mode [10] and the reverse mode [2]. The associativity of the chain rule makes it possible to apply this time-parallel approach to arbitrary computational structures, not just the linear schedule illustrated here.

Parallel AD Tools

Research in AD and parallelism is relatively new. Nonetheless, there are several such tools, at varying stages of development.

Hinkins developed special purpose libraries for the AD of programs written in MPFortran or CM-



Automatic Differentiation: Parallel Computation, Figure 1

Fortran [22]. Use of these libraries required that each arithmetic operation be manually replaced by a subroutine call. As part of his thesis [23], Hovland developed prototype tools for AD of FortranM [12], Fortran with a subset of MPI [20] message passing, and C with MPI. Carle is developing (1999) a prototype version of ADIFOR [11] supporting MPI and PVM. Roh is developing an extension to ADIC that seeks to automatically exploit the parallelism introduced by AD through the use of threads [27].

Summary

Since 1989, a great deal of progress has been made in the fields of automatic differentiation and parallel computation. Parallel computation and AD interact in two ways. AD can be applied to a parallel program. Alternatively, AD can be used as a source of new parallelism in a computation. Effective strategies exist for exploiting each of the two types of parallelism introduced: time parallelism and data parallelism.

In either case, ensuring that the resulting derivative computation is both correct and efficient requires AD tools that are more sophisticated than in the serial setting. Most of the existing tools are early in their development cycle, but can be expected to mature swiftly as they adopt advanced computational infrastructure developed in other fields of computer science, e. g., parallelizing compilers or parallel runtime systems. Thus, we expect the beginning of 2000 to also provide robust

and effective tools for the differentiation of parallel programs and the introduction of parallelism through differentiation.

See also

- ► Asynchronous Distributed Optimization Algorithms
- ► Automatic Differentiation: Calculation of the Hessian
- ► Automatic Differentiation: Calculation of Newton Steps
- ► Automatic Differentiation: Geometry of Satellites and Tracking Stations
- ► Automatic Differentiation: Introduction, History and Rounding Error Estimation
- ► Automatic Differentiation: Point and Interval
- ► Automatic Differentiation: Point and Interval Taylor Operators
- ► Automatic Differentiation: Root Problem and Branch Problem
- ► Heuristic Search
- ► Interval Analysis: Parallel Methods for Global Optimization
- ► Load Balancing for Parallel Optimization Techniques
- ► Nonlocal Sensitivity Analysis with Automatic Differentiation
- ► Parallel Computing: Complexity Classes
- ► Parallel Computing: Models
- ► Parallel Heuristic Search
- ► Stochastic Network Problems: Massively Parallel Solution

References

1. Abate J, Bischof Ch, Carle A, Roh L (1997) Algorithms and design for a second-order automatic differentiation module. Proc. Internat. Symp. Symbolic and Algebraic Computing (ISSAC) '97, ACM, New York, pp 149–155
2. Benary J (1996) Parallelism in the reverse mode. In: Berz M, Bischof Ch, Corliss G, Griewank A (eds) Computational Differentiation: Techniques, Applications, and Tools. SIAM, Philadelphia, pp 137–147
3. Berz M, Bischof Ch, Corliss G, Griewank A (1996) Computational differentiation: Techniques, applications, and tools. SIAM, Philadelphia
4. Bischof ChH (1991) Issues in parallel automatic differentiation. In: Griewank A, Corliss G (eds) Automatic Differentiation of Algorithms. SIAM, Philadelphia, pp 100–113
5. Bischof Ch, Carle A, Corliss G, Griewank A, Hovland P (1992) ADIFOR: Generating derivative codes from Fortran programs. Scientif Program 1(1):11–29
6. Bischof Ch, Carle A, Khademi P, Mauer A (1996) ADIFOR 2.0: Automatic differentiation of Fortran 77 programs. IEEE Comput Sci Eng 3(3):18–32
7. Bischof Ch, Green L, Haigler K, Knauff T (1994) Parallel calculation of sensitivity derivatives for aircraft design using automatic differentiation. Proc. 5th AIAA/NASA/USAF/ISSMO Symp. Multidisciplinary Analysis and Optimization, AIAA-94-4261, Amer Inst Aeronautics and Astronautics, Reston, VA, pp 73–84
8. Bischof ChH, Haghighat MR (1996) On hierarchical differentiation. In: Berz M, Bischof Ch, Corliss G, Griewank A (eds) Computational Differentiation: Techniques, Applications, and Tools. SIAM, Philadelphia, pp 83–94
9. Bischof Ch, Roh L, Mauer A (1997) ADIC – An extensible automatic differentiation tool for ANSI-C. Software Practice and Experience 27(12):1427–1456
10. Bischof Ch, Wu Po-Ting (1997) Time-parallel computation of pseudo-adjoints for a leapfrog scheme. Preprint Math and Computer Sci Div Argonne Nat Lab no. ANL/MCS-P639-0197
11. Carle A (1997) ADIFOR-MP – A prototype automatic differentiation tool for Fortran 77 with message-passing extensions. Personal communication
12. Foster IT, Chandy KM (1995) Fortran M: A language for modular parallel programming. J Parallel Distributed Comput 25(1)
13. Geist A, Beguelin A, Dongarra J, Jiang W, Manchek R, Sunderam V (1994) PVM – Parallel virtual machine: A users' guide and tutorial for network parallel computing. MIT, Cambridge, MA
14. Giering R, Kaminski Th (1996) Recipes for adjoint code construction. Max-Planck Inst Meteorologie, Hamburg no. 212
15. Griewank A (1989) On automatic differentiation. In: Iri M, Tanabe K (eds) Mathematical Programming: Recent Developments and Applications. Kluwer, Dordrecht, pp 83–108
16. Griewank A (1992) Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. Optim Methods Softw 1(1):35–54
17. Griewank A, Corliss G (1991) Automatic differentiation of algorithms. SIAM, Philadelphia
18. Griewank A, Juedes D, Utke J (1996) ADOL-C, a package for the automatic differentiation of algorithms written in C/C++. ACM Trans Math Softw 22(2):131–167
19. Grimm J, Pottier L, Rostaing-Schmidt N (1996) Optimal time and minimum space time product for reversing a certain class of programs. In: Berz M, Bischof Ch, Corliss G, Griewank A (eds) Computational Differentiation, Techniques, Applications, and Tools. SIAM, Philadelphia, pp 95–106

20. Gropp W, Lusk E, Skjellum A (1994) Using MPI – Portable parallel programming with the message passing interface. MIT, Cambridge, MA
21. Herley K (1993) On the NP-completeness of optimum accumulation by vertex elimination. Unpublished Manuscript
22. Hinkins R L (Sept. 1994) Parallel computation of automatic differentiation applied to magnetic field calculations. MSc Thesis Univ Calif
23. Hovland P (1997) Automatic differentiation of parallel programs. PhD Thesis Univ. Illinois at Urbana-Champaign
24. Hovland P, Bischof Ch (1998) Automatic differentiation of message-passing parallel programs. Proc. First Merged Internat. Parallel Processing Symp. and Symp. on Parallel and Distributed Processing, IEEE Computer Soc Press, New York
25. Hovland P, Bischof Ch, Spiegelman D, Casella M (1997) Efficient derivative codes through automatic differentiation and interface contraction: An application in biostatistics. SIAM J Sci Comput 18(4):1056–1066
26. Koelbel C, Loveman D, Schreiber R, Steele G Jr, Zosel M (1994) The high performance Fortran handbook. MIT, Cambridge, MA
27. Roh L (1997) Personal Communication
28. Rostaing N, Dalmas St, Galligo A (Oct. 1993) Automatic differentiation in Odyssee. Tellus 45a(5):558–568
29. Shiriaev D, Griewank A (1996) ADOL-F: Automatic differentiation of Fortran codes. In: Berz M, Bischof Ch, Corliss G, Griewank A (eds) Computational Differentiation: Techniques, Applications, and Tools. SIAM, Philadelphia, pp 375–384
30. Snir M, Otto SW, Huss-Lederman S, Walker DW, Dongarra Jack (1996) MPI: The complete reference. MIT, Cambridge, MA
31. WEB http://www.mcs.anl.gov/autodiff/adtools/
32. Zhang Y, Bischof Ch, Easter R, Wu Po-Ting (1997) Sensitivity analysis of O3 and photochemical indicators using a mixed-phase chemistry box model and automatic differentiation techniques. 90th Air and Waste Management Assoc. Annual Meeting and Exhibition June 8-13, 1997, Toronto. vol 97-WA68A.04, Air and Waste Management Assoc, Pittsburgh, PA, pp 1–16

Automatic Differentiation: Point and Interval

AD

L. B. Rall[1], George F. Corliss[2]
[1] University Wisconsin–Madison, Madison, USA
[2] Marquette University, Milwaukee, USA

MSC2000: 65H99, 65K99

Article Outline

Keywords

Differentiation; Computational methods

Automatic differentiation (abbreviated *AD*) is a computational method for evaluating derivatives or Taylor coefficients of algorithmically defined functions. Simply speaking, an *algorithmic definition* of a function is a step-by-step specification of its evaluation by arithmetic operations and library functions. Application of the rules of differentiation to the algorithmic definition of a differentiable function yields values of its derivatives. Examples of algorithmic definitions of functions are code lists, computer subroutines, and even entire computer programs.

Automatic differentiation differs from numerical differentiation based on difference quotients of function values in that automatic differentiation is exact in principle, but of course is subject to roundoff error in practice. In addition to roundoff error, difference quotients entail truncation error. Attempts to reduce this truncation error by decreasing stepsize results in cancellation of significant digits and a catastrophic increase in roundoff error in general. Automatic differentiation also differs significantly from the symbolic differentiation taught in school, the goal of which is the transformation of formulas for functions into formulas for their derivatives. Although automatic differentiation uses the same rules of differentiation as symbolic differentiation, these rules are applied to the algorithmic definition of the function, not to a formula for it, and the results are values of derivatives, not formulas. Furthermore, formulas may not be available for functions of interest defined only algorithmically by computer subroutines or programs to which automatic differentiation can be applied. In summary, automatic differentiation is more accurate than numerical differentiation and requires fewer resources and is more generally applicable than symbolic differentiation.

The simplest type of algorithmic definition of a function is a *code list*, which is similar to the segment of computer code for the evaluation of an expression

(i. e., a formula). For illustration, consider the function defined by the formula

$$f(x, y) = (xy + \sin x + 4)(3y^2 + 6).$$

An equivalent algorithmic definition of this function by a code list is

$$\begin{array}{lllllll} t_1 & = & x, & & t_6 & = & t_5 + 4, \\ t_2 & = & y, & & t_7 & = & t_2^2, \\ t_3 & = & t_1 t_2, & & t_8 & = & 3t_7, \\ t_4 & = & \sin t_1, & & t_9 & = & t_8 + 6, \\ t_5 & = & t_3 + t_4, & & t_{10} & = & t_6 t_9. \end{array}$$

Given the values of x and y, evaluation of the subsequent entries in the code list gives $t_{10} = f(x, y)$. Indeed, the first step in evaluation or symbolic differentiation of a function defined by a formula is to form a corresponding code list, perhaps subconsciously. The conversion of well-formed expressions into code lists is a fundamental process in computer science, sometimes called 'formula translation'. Although both automatic differentiation and symbolic differentiation are applicable in this case, automatic differentiation requires only the code list and produces only values of derivatives for given values of the input variables. To compute the *gradient* ∇f, the rules of differentiation applied to the code list above gives

$$\begin{aligned} \nabla t_1 &= \nabla x, \\ \nabla t_2 &= \nabla y, \\ \nabla t_3 &= t_1 \nabla t_2 + t_2 \nabla t_1, \\ \nabla t_4 &= (\cos t_1) \nabla t_1, \\ \nabla t_5 &= \nabla t_3 + \nabla t_4, \\ \nabla t_6 &= \nabla t_5, \\ \nabla t_7 &= 2t_2 \nabla t_2, \\ \nabla t_8 &= 3 \nabla t_7, \\ \nabla t_9 &= \nabla t_8, \\ \nabla t_{10} &= t_6 \nabla t_9 + t_9 \nabla t_6. \end{aligned}$$

It is evident from the chain rule that

$$\nabla t_{10} = \nabla f(x, y) = f_x(x, y) \nabla x + f_y(x, y) \nabla y.$$

Thus, once the code list for $f(x, y)$ is given and the 'seed' values of x, ∇x and y, ∇y are known, the values of the function and its gradient can be computed without formulas for either. In case x, y are independent variables, then $\nabla x = [1, 0]$, $\nabla y = [0, 1]$ and

$$\begin{aligned} \nabla f(x, y) &= [f_x(x, y), f_y(x, y)] \\ &= [t_9(t_2 + \cos t_1), 6t_2 t_6 + t_1 t_9]. \end{aligned}$$

This example illustrates the *forward mode* of automatic differentiation. This process is not restricted to first derivatives as long as the entries t_i of the code list have the desired number of derivatives.

Although the forward mode illustrated above is easy to understand and implement, it is usually more efficient to compute gradients in what is called the *reverse mode*. To explain this process, consider a general code list $t = (t_1, \ldots, t_n)$ which begins with m input variables $t_1, \ldots, t_m$, and ends with p output variables $t_{n-p+1}, \ldots, t_n$. For $i > m$, the entry $t_i = t_j \circ t_k$, where $j, k < i$ and $\circ$ denotes an arithmetic operation, or $t_i = \phi(t_j)$ with $j < i$, where ϕ is a function belonging to a library of standard functions. For convenience, arithmetic operations between constants and entries will be considered library functions in addition to the usual sine, cosine, and so on.

If K_i denotes the set of indices $k < i$ such that the entry t_i of the code list depends explicitly on t_k, then the forward mode of automatic differentiation consists of application of the chain rule in the form

$$\nabla t_i = \sum_{k \in K_i} \frac{\partial t_i}{\partial t_k} \nabla t_k$$

for $i = m + 1, \ldots, n$, to obtain the gradients of the intermediate variables and output. This process works because $\nabla t_1, \ldots, \nabla t_{i-1}$ are known or have been computed before they are needed for the evaluation of ∇t_i. If the seed gradients have dimension at most d, then the forward mode of automatic differentiation requires computational effort proportional to nd, that is, d times the effort required for evaluation of the output t_n. If $d > m$, then it is more efficient to consider the input variables to be independent and then compose ∇f by the standard formula given below. This limits the computational effort for the forward mode to an amount essentially proportional to nm.

The reverse mode is another way to apply the chain rule. Instead of propagating the seed gradients $\nabla t_1, \ldots, \nabla t_m$ throughout the computation, differentiation is applied to the code list in reverse order. In the case of

a single output variable t_n, first t_n is differentiated with respect to itself, then with respect to $t_{n-1}, \ldots, t_1$. The resulting *adjoints* $\partial t_n/\partial t_m, \ldots, \partial t_n/\partial t_1$ and the seed gradients then give

$$\nabla t_n = \sum_{i=1}^{m} \frac{\partial t_n}{\partial t_i} \nabla t_i .$$

Formally, the adjoints are given by

$$\frac{\partial t_n}{\partial t_n} = 1, \qquad \frac{\partial t_n}{\partial t_k} = \sum_{i \in I_k} \frac{\partial t_k}{\partial t_i} \frac{\partial t_i}{\partial t_k},$$

$k = n-1, \ldots, 1$, where I_k is the set of indices $i > k$ such that t_i depends explicitly on t_k. It follows that the computational effort to obtain adjoints in the reverse mode is proportional to n, the length of the code list, and is essentially independent of the number of input variables and the dimensionalities of the seed gradients. This can result in significant savings in computational time. In the general case of several output variables, the same technique is applied to each to obtain their gradients.

The reverse mode applied to the example code list gives

$$\begin{aligned}
\frac{\partial t_{10}}{\partial t_{10}} &= 1, \\
\frac{\partial t_{10}}{\partial t_9} &= t_6, \\
\frac{\partial t_{10}}{\partial t_8} &= \frac{\partial t_{10}}{\partial t_9}\frac{\partial t_9}{\partial t_8} = t_6 \cdot 1, \\
\frac{\partial t_{10}}{\partial t_7} &= \frac{\partial t_{10}}{\partial t_8}\frac{\partial t_8}{\partial t_7} = t_6 \cdot 3, \\
\frac{\partial t_{10}}{\partial t_6} &= t_9, \\
\frac{\partial t_{10}}{\partial t_5} &= \frac{\partial t_{10}}{\partial t_6}\frac{\partial t_6}{\partial t_5} = t_9 \cdot 1, \\
\frac{\partial t_{10}}{\partial t_4} &= \frac{\partial t_{10}}{\partial t_5}\frac{\partial t_5}{\partial t_4} = t_9 \cdot 1, \\
\frac{\partial t_{10}}{\partial t_3} &= \frac{\partial t_{10}}{\partial t_5}\frac{\partial t_5}{\partial t_3} = t_9 \cdot 1, \\
\frac{\partial t_{10}}{\partial t_2} &= \frac{\partial t_{10}}{\partial t_7}\frac{\partial t_7}{\partial t_2} + \frac{\partial t_{10}}{\partial t_3}\frac{\partial t_3}{\partial t_2} \\
&= (3t_6) \cdot (2t_2) + t_9 \cdot t_1, \\
\frac{\partial t_{10}}{\partial t_1} &= \frac{\partial t_{10}}{\partial t_4}\frac{\partial t_4}{\partial t_1} + \frac{\partial t_{10}}{\partial t_3}\frac{\partial t_3}{\partial t_1} = t_9 \cdot \cos t_1 + t_9 \cdot t_2 .
\end{aligned}$$

Although this computation appears to be complicated, a comparison of operation counts in the case x, y are independent variables shows that even for this low-dimensional example, the reverse mode requires 13 operations to evaluate ∇f in addition to the operations required to evaluate f itself, while the forward mode requires $22 = 2 + 10\ m$. In reverse mode, the entire code list has to be evaluated and its values stored before the reverse sweep begins. In forward mode, since the computation of t_i and each component of ∇t_i can be carried out independently, a parallel computer with a sufficient number of processors could compute t_n, ∇t_n in a single pass through the code list, that is, with effort proportional to n. A more detailed comparison of forward and reverse modes for calculating gradients can be found in the tutorial article [1, pp. 1–18] and the book [3].

Implementation of automatic differentiation can be by *interpretation*, *operator overloading*, or *code transformation*. Early software for automatic differentiation simply interpreted a code list by calling the appropriate subroutines for each arithmetic operation or library function. Although inefficient, this approach is still useful in interactive applications in which functions entered from the keyboard are parsed to form code lists, which are then interpreted to evaluate the functions and their derivatives.

Operator overloading is a familiar concept in mathematics, as the symbol '+' is used to denote addition of such disparate objects as integers, real or complex numbers, vectors, matrices, functions, etc. It follows that a code list as defined above can be evaluated in any mathematical system in which the required arithmetic operations and library function are available, including differentiation arithmetics [14, pp. 73–90]. These arithmetics can be used to compute derivatives or Taylor coefficients of any order of sufficiently smooth functions. In optimization, gradient and Hessian arithmetics are most frequently used. In gradient arithmetic, the basic data type is the ordered pair $(f, \nabla f)$ of a number and a vector representing values of a function and its gradient vector. Arithmetic operations in this system are defined by

$$\begin{aligned}
(f, \nabla f) \pm (g, \nabla g) &= (f \pm g, \nabla f \pm \nabla g), \\
(f, \nabla f)(g, \nabla g) &= (fg, f\nabla g + g\nabla f), \\
\frac{(f, \nabla f)}{(g, \nabla g)} &= \left(\frac{f}{g}, \frac{g\nabla f - f\nabla g}{g^2}\right),
\end{aligned}$$

division by 0 excluded. If ϕ is a differentiable library function, then its extension to gradient arithmetic is defined by

$$\phi(f, \nabla f) = (\phi(f), \phi'(f)\nabla f),$$

which is just the chain rule. Hessian arithmetic extends the same idea to triples $(f, \nabla f, Hf)$, where Hf is a matrix representing the value of the *Hessian* of f, $Hf = [\partial^2 f/\partial x_i\, \partial x_j]$.

Programming differentiation arithmetic is convenient in modern computer languages which support operator overloading [9, pp. 291–309]. In this setting, the program is written with expressions or routines for functions in the regular form, and the compiler produces executable code for evaluation of these functions and the desired derivatives. For straightforward implementations such as the one cited above, the differentiation mode will be forward, which has implications for efficiency.

Code transformation essentially consists of analyzing the code for functions to generate code for derivatives. This results in a new computer program which then can be compiled and run as usual. To illustrate this idea, note that in the simple example given above, the expressions

$$f_x(x, y) = t_9(t_2 + \cos t_1),$$
$$f_y(x, y) = 6t_2t_6 + t_1t_9,$$

were obtained for the partial derivatives of the function in either forward or reverse mode. This differs from symbolic differentiation in that values of intermediate entries in the code list for $f(x, y)$ are involved rather than the variables x, y. The corresponding lists for these expressions

$$tx_1 = \cos t_1,$$
$$tx_2 = t_2 + tx_1,$$
$$tx_3 = t_9tx_2,$$
$$ty_1 = t_2t_6,$$
$$ty_2 = 6ty_1,$$
$$ty_3 = t_1t_9,$$
$$ty_4 = ty_2 + ty_3,$$

can then be appended to the code list for the function to obtain a routine with output values $t_{10} = f(x, y)$, $tx_3 = f_x(x, y)$, and $ty_4 = f_y(x, y)$. Further, automatic differentiation can be applied to this list to obtain routines for higher derivatives of f [13]. As a practical matter, duplicate assignments can be removed from such lists before compilation.

Up to this point, the discussion has been of *point AD*, values have been assumed to be real or complex numbers with all operations and library functions evaluated exactly. In reality, the situation is quite different. Expressions, meaning their equivalent code lists, are evaluated in an approximate computer arithmetic known as floating-point arithmetic. This often yields very accurate results, but examples of simple expressions are known for which double and even higher precision calculation gives an answer in which even the sign is wrong for certain input values. Furthermore, such failures can occur without any outward indication of trouble. In addition, values of input variables may not be known exactly, thus increasing the uncertainty in the accuracy of outputs. The use of *interval arithmetic* (abbreviated *IA*) provides a computational way to attack these problems [11].

The basic quantities in interval arithmetic are finite closed real intervals $X = [x_1, x_2]$, which represent all real numbers x such that $x_1 \le x \le x_2$. Arithmetic operations $\circ$ on intervals are defined by

$$X \circ Y = \{x \circ y\colon\ x \in X, y \in Y\},$$

again an interval, division by an interval containing zero excluded. Library functions ϕ are similarly extended to interval functions Φ such that $\phi(x) \in \Phi(X)$ for all $x \in X$ with $\Phi(X)$ expected to be an accurate inclusion of the range $\phi(X)$ of ϕ on X. Thus, if $f(x)$ is a function defined by a code list, then assignment of the interval value X to the input variable and evaluation of the entries in interval arithmetic yields the output $F(X)$ such that $f(x) \in F(X)$ for all $x \in X$. The interval function F obtained in this way is called the *united extension* of f [11].

In the floating-point version of interval arithmetic, all endpoints are floating-point numbers and hence exactly representable in the computer. Results of arithmetic operations and calls of library functions are

rounded outwardly (upper endpoints up, lower endpoints down) to the closest or very close floating-point numbers to maintain the guarantee of inclusion. Thus, one is still certain that for the interval extension F of f actually computed, $f(x) \in F(X)$ for all $x \in X$. Thus, for example, an output interval $F(X)$ which is very wide for a point input interval $X = [x, x]$ would serve as a warning that the algorithm is inappropriate or ill-conditioned, in contrast to the lack of such information in ordinary floating-point arithmetic.

Automatic differentiation carried out in interval arithmetic is called *interval automatic differentiation*. Interval computation has numerous implications for optimization, with or without automatic differentiation [6]. Maxima and minima of functions can 'slip through' approximate sampling of values at points of the floating-point grid, but have to be contained in the computable interval inclusion $F(X)$ of $f(x)$ over the same interval region X, for example.

Although interval arithmetic properly applied can solve many optimization and other computational problems, a word of warning is in order. The properties of interval arithmetic differ significantly from those of real arithmetic, and simple 'plugging in' of intervals for numbers will not always yield useful results. In particular, interval arithmetic lacks additive and multiplicative inverses, and multiplication is only subdistributive across addition, $X(Y+Z) \subset XY+XZ$ [11]. A real algorithm which uses one or more of these properties of real arithmetic is usually inappropriate for interval computation, and should be replaced by one that is suitable if possible.

To this point, automatic differentiation has been applied only to code lists, which programmers customarily refer to as 'straight-line code'. Automatic differentiation also applies to subroutines and programs, which ordinarily contain loops and branches in addition to expressions. These latter present certain difficulties in many cases. A loop which is traversed a fixed number of times can be 'unrolled,' and thus is equivalent to straight-line code. However, in case the stopping criterion is based on result values, the derivatives may not have achieved the same accuracy as the function values. For example, if the inverse function of a known function is being computed by iterative solution of the equation $f(x) = y$ for $x = f^{-1}(y)$, then automatic differentiation should be applied to f and the derivative of the inverse function obtained from the standard formula $(f^{-1})'(y) = (f'(x))^{-1}$. Branches essentially produce piecewise defined functions, and automatic differentiation then provides the derivative of the function defined by whatever branch is taken. This can create difficulties as described by H. Fischer [4, pp. 43–50], especially since a smooth function can be approximated well in value by highly oscillatory or other nonsmooth functions such as result from table lookups and piecewise rational approximations. For example, one would not expect to obtain an accurate approximation to the cosine function by applying automatic differentiation to the library subroutine for the sine. As with any powerful tool, automatic differentiation should not be expected to provide good results if applied indiscriminately, especially to 'legacy' code. As with interval arithmetic, automatic differentiation will yield the best results if applied to programs written with it in mind.

Current state of the art software for point automatic differentiation of programs are ADOL-C, for programs written in C/C++ [5], and ADIFOR for programs in Fortran 77 [1, pp. 385–392].

Numerous applications of automatic differentiation to optimization and other problems can be found in the conference proceedings [1,4], which also contain extensive bibliographies. An important result with implications for optimization is that automatic differentiation can be used to obtain Newton steps *without* forming Jacobians and solving linear systems, see [1, pp. 253–264].

From a historical standpoint, the principles of automatic differentiation go back to the early days of calculus, but implementation is a product of the computer age, hence the designation 'automatic'. The terminology 'algorithmic differentiation', to which the acronym automatic differentiation also applies, is perhaps better. Since differentiation is widely understood, automatic differentiation literature contains many anticipations and rediscoveries. The 1962 Stanford Ph.D. thesis of R.E. Moore deals with both interval arithmetic and automatic differentiation of code lists to obtain Taylor coefficients of series solution of systems of ordinary differential equations. In 1964, R.E. Wengert [15] published on automatic differentiation of code lists and noted that derivatives could be recovered from Taylor coefficients. Early results in automatic differentiation were applied to code lists in forward mode, as described

in [13]. G. Kedem [8] showed that automatic differentiation applies to subroutines and programs, again in forward mode. The reverse mode was anticipated by S. Linnainmaa in 1976 [10], and in the Ph.D. thesis of B. Speelpenning (Illinois, 1980), and published in more complete form by M. Iri in 1984 [7]. automatic differentiation via operator overloading and the concept of differentiation arithmetics, which are commutative rings with identity, were introduced by L.B. Rall [9, pp. 291–309], [14, pp. 73–90], [4, pp. 17–24]. For additional information about the early history of automatic differentiation, see [13] and the article by Iri [4, pp. 3–16] for later developments.

Analysis of algorithms for automatic differentiation has been carried out on the basis of graph theory by Iri [7], A. Griewank [12, pp. 128–161], [3], and equivalent matrix formulation by Rall [2, pp. 233–240].

See also

- Automatic Differentiation: Calculation of the Hessian
- Automatic Differentiation: Calculation of Newton Steps
- Automatic Differentiation: Geometry of Satellites and Tracking Stations
- Automatic Differentiation: Introduction, History and Rounding Error Estimation
- Automatic Differentiation: Parallel Computation
- Automatic Differentiation: Point and Interval Taylor Operators
- Automatic Differentiation: Root Problem and Branch Problem
- Bounding Derivative Ranges
- Global Optimization: Application to Phase Equilibrium Problems
- Interval Analysis: Application to Chemical Engineering Design Problems
- Interval Analysis: Differential Equations
- Interval Analysis: Eigenvalue Bounds of Interval Matrices
- Interval Analysis: Intermediate Terms
- Interval Analysis: Nondifferentiable Problems
- Interval Analysis: Parallel Methods for Global Optimization
- Interval Analysis: Subdivision Directions in Interval Branch and Bound Methods
- Interval Analysis: Systems of Nonlinear Equations
- Interval Analysis: Unconstrained and Constrained Optimization
- Interval Analysis: Verifying Feasibility
- Interval Constraints
- Interval Fixed Point Theory
- Interval Global Optimization
- Interval Linear Systems
- Interval Newton Methods
- Nonlocal Sensitivity Analysis with Automatic Differentiation

References

1. Berz M, Bischof Ch, Corliss G, Griewank A (eds) (1996) Computational differentiation, techniques, applications, and tools. SIAM, Philadelphia
2. Fischer H, Riedmueller B, Schaeffler S (eds) (1996) Applied mathematics and parallel computing. Physica Verlag, Heidelberg
3. Griewank A (2000) Evaluating derivatives: Principles and techniques of algorithmic differentiation. SIAM, Philadelphia
4. Griewank A, Corliss GF (eds) (1991) Automatic differentiation of algorithms, theory, implementation, and application. SIAM, Philadelphia
5. Griewank A, Juedes D, Utke J (1996) ADOL-C, a package for the automatic differentiation of programs written in C/C++. ACM Trans Math Softw 22:131–167
6. Hansen E (1992) Global optimization using interval analysis. M. Dekker, New York
7. Iri M (1984) Simultaneous computation of functions, partial derivatives, and rounding errors: complexity and practicality. Japan J Appl Math 1:223–252
8. Kedem G (1980) Automatic differentiation of computer programs. ACM Trans Math Softw 6:150–165
9. Kulisch UW, Miranker WL (eds) (1983) A new approach to scientific computation. Acad. Press, New York
10. Linnainmaa S (1976) Taylor expansion of the accumulated rounding error. BIT 16:146–160
11. Moore RE (1979) Methods and applications of interval analysis. SIAM, Philadelphia
12. Pardalos PM (eds) (1993) Complexity in nonlinear optimization. World Sci, Singapore
13. Rall LB (1981) Automatic differentiation: Techniques and applications. Springer, Berlin
14. Ullrich C (eds) (1990) Computer arithmetic and self-validating numerical methods. Acad Press, New York
15. Wengert RE (1964) A simple automatic derivative evaluation program. Comm ACM 7:463–464

Automatic Differentiation: Point and Interval Taylor Operators

AD, Computational Differentiation

JAMES B. WALTERS, GEORGE F. CORLISS
Marquette University, Milwaukee, USA

MSC2000: 65K05, 90C30

Article Outline

Keywords

Automatic differentiation; Code list; Interval arithmetic; Overloaded operator; Taylor series

Frequently of use in optimization problems, automatic differentiation may be used to generate Taylor coefficients. Specialized software tools generate Taylor series approximations, one term at a time, more efficiently than the general AD software used to compute (partial) derivatives. Through the use of operator overloading, these tools provide a relatively easy-to-use interface that minimizes the complications of working with both point and interval operations.

Introduction

First, we briefly survey the tools of automatic differentiation and operator overloading used to compute point- and interval-valued Taylor coefficients. We assume that f is an analytic function $f : \mathbf{R} \to \mathbf{R}$. Automatic differentiation (AD or computational differentiation) is the process of computing the derivatives of a function f at a point $t = t_0$ by applying rules of calculus for differentiation [9,10,17,18]. One way to implement AD uses overloaded operators.

Operator Overloading

An overloaded (or generic) operator invokes a procedure corresponding to the types of its operands. Most programming languages implement this technique for arithmetic operations. The sums of two floating point numbers, two integers, or one floating point number and one integer are computed using three different procedures for addition. Fortran 77 or C denies the programmer the ability to replace or modify the various routines used implicitly for integer, floating point, or mixed-operand arithmetic, but Fortran 95, C++, and Ada support operator overloading for user-defined types. Once we have defined an overloaded operator for each rule of differentiation, AD software performs those operations on program code for f, as shown below. The operators either propagate derivative values or construct a code list for their computation. We give prototypical examples of operators overloaded to propagate Taylor coefficients below.

Automatic Differentiation

The AD process requires that we have f in the form of an algorithm (e. g. computer program) so that we can easily separate and order its operations. For example, given $f(t) = e^t/(2 + t)$, we can express f as an algorithm in Fortran 95 or in C++ (using an assumed AD module or class):

In this section, we use AD to compute first derivatives. In the next section, we extend to point- and interval-valued Taylor series. To understand the AD process, we parse the program above into a sequence of unary and binary operations, called a *code list*, *computational graph*, or 'tape' [9]:

$$x_0 = t_0, \qquad x_2 = 2 + x_0,$$
$$x_1 = \exp(x_0), \qquad x_3 = \frac{x_1}{x_2}.$$

```
program Example1
    use AD_Module
    type(AD_Independent) :: t
        AD_Independent(0)
    type(AD_Dependent) :: f
    f = exp(t)/(2 + t)
end program Example1
#include 'AD_class.h'
void main (void) {
    AD_Independent t(0);
    AD_Dependent f;
    f = exp(t)/(2 + t);
}
```

Automatic Differentiation: Point and Interval Taylor Operators, Figure 1
Fortran and C++ calls to AD operators

Differentiation is a simple mechanical process for propagating derivative values. Let $t = t_0$ represent the value of the independent variable with respect to which we differentiate. We know how to take the derivative of a variable, a constant, and unary and binary operations (i. e. +, −, ∗, /, sin, cos, exp, etc.). Then AD software annotates the code list:

$$
\begin{aligned}
x_0 &= t_0; \\
\nabla x_0 &= 1, \\
x_1 &= \exp(x_0); \\
\nabla x_1 &= \exp(x_0) * \nabla x_0, \\
x_2 &= 2 + x_0; \\
\nabla x_2 &= 0 + \nabla x_0, \\
x_3 &= \frac{x_1}{x_2}; \\
\nabla x_3 &= \frac{(\nabla x_1 - \nabla x_2 * x_3)}{x_2}.
\end{aligned}
$$

AD propagates values of derivatives, not expressions as symbolic differentiation does. AD values are exact (up to round-off), not approximations of unknown quality as finite differences. For more information regarding AD and its applications, see [2,8,9,10, 17,18], or the bibliography [21].

AD software can use overloaded operators in two different ways. Operators can propagate both the value x_i and its derivative ∇x_i, as suggested by the annotated code list above. This approach is easy to understand and to program. We give prototypical Taylor operators of this flavor below.

The second approach has the operators construct and store the code list. Various optimizations and parallel scheduling [1,4,12] may be applied to the code list. Then the code list is interpreted to propagate derivative values. This is the approach of AD tools such as *ADOL-C* [11], *ADOL-F* [20], *AD01* [16], or *INTOPT_90* [13]. The second approach is much more flexible, allowing the code list to be traversed in either the forward or reverse modes of AD (see [9]) or with various arithmetics (e. g. point- or interval-valued series).

AD may be applied to functions of more than one variable, in which partial derivatives with respect to each are computed in turn, and to vector functions, in which the component functions are differentiated in succession. In addition, we can compute higher order derivative values. One application of AD involving higher order derivatives of f is the computation of Taylor (series) coefficients to which we turn in the next section.

Source code transformation is a third approach to AD software used by *ATOMFT* [5] for Taylor series and by *ADIFOR* [3], *PADRE2* [14], or *Odyssée* [19] for partial derivatives. Such tools accept the algorithm for f as data, rather than for execution, and produce code for computing the desired derivatives. The resulting code often executes more rapidly than code using overloaded operators.

Taylor Coefficients

We define the Taylor coefficients of the analytic function f at the point $t = t_0$:

$$(f|t_0)_i := \frac{1}{i!}\frac{d^i f(t_0)}{dt^i},$$

for $i = 0, 1, \ldots$, and let $F := ((f|t_0)_i)$ denote the vector of Taylor coefficients. Then *Taylor's theorem* says

that there exists some point τ (usually not practically obtainable) between t and t_0 such that

$$f(t) = \sum_{i=0}^{p} (f|t_0)_i (t - t_0)^i + \frac{1}{(p+1)!} \frac{d^{p+1} f(\tau)}{dt^{p+1}} (t - t_0)^{p+1}. \tag{1}$$

Computation of Taylor coefficients requires differentiation of f. We generate Taylor coefficients automatically using recursion formulas for unary and binary operations. For example, the recurrences we need for our example $f(t) = e^t/(2 + t)$ are

$$\begin{aligned}
&x(t) = \exp u(t) \Rightarrow x' = xu', \\
&\quad (x)_0 = \exp(u)_0, \\
&\quad (x)_i = \sum_{j=0}^{i-1} (x)_j * (u)_{i-j} * \frac{(i-j)}{i}; \\
&x(t) = u(t) + v(t), \\
&\quad (x)_i = (u)_i + (v)_i; \\
&x(t) = \frac{u(t)}{v(t)} \Rightarrow xv = u, \\
&\quad (x)_i = \frac{\left((u)_i - \sum_{j=0}^{i-1} (x)_j * (v)_{i-j}\right)}{(v)_0}.
\end{aligned}$$

The recursion relations are described in more detail in [17]. Except for + and −, each recurrence follows from Leibniz' rule for the Taylor coefficients of a product. The relations can be viewed as a lower triangular system. The recurrence represents a solution by forward substitution, but there are sometimes accuracy or stability advantages in an iterative solution to the lower triangular system. The recurrences for each operation can be evaluated in floating-point, complex, interval, or other appropriate arithmetic.

To compute the formal series for $f(t) = e^t/(2 + t)$ expanded at $t = 0$,

$$\begin{cases}
X_0 := (t_0, 1, 0, \ldots)(0, 1, 0, \ldots), \\
X_1 := \exp X_0 = \left(1, 1, \frac{1}{2!}, \frac{1}{3!}, \ldots\right), \\
X_2 := 2 + X_0 = (2, 1, 0, \ldots), \\
X_3 := \frac{X_2}{X_3} = \left(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{48}, \ldots\right).
\end{cases} \tag{2}$$

```
class Taylor {      // Or make a template:
   private:
      cont int Max_Length = 20;
      Value_type coef[Max_Length];
   public:
      Taylor ( Value_type t_0 ) {
            // Constructor for Independents
         coef[0] = t_0; coef[1] = 1;
         for(int i  =  2;  i ¡ Max_Length;  i++)
            { coef[i] = 0;  }
      }
      Taylor ( void ) {
            // Constructor for Dependents
         for (int i = 0; i ¡  Max_Length; i++)
            { coef[i] = 0; }
      }
      Taylor ( Taylor &U) {
            // Copy Constructor
         for (int i=0; i ¡  Max_Length; i++)
            { coef[i] = Value_type(U.coef)[i]; }
      }
      friend Taylor operator +
            (int u, Taylor V) {
         V.coef[0] += u; return V;
      }
      friend Taylor operator /
            (Taylor U, Taylor V) {
         Taylor X;
         for (int i = 0; i ¡  Max_Length; i++) {
            Value_type sum = U.coef[i];
            for (int j = 0; j ¡  i; j++)
               { sum − =X.coef[j] * V.coef[i−j]; }
            X.coef[i] = sum / V.coef[0];
         }
         return X;
      }
      friend Taylor exp (Taylor U)
         { /* Similar to divide */ }
      Value_type getCoef (int i)
         { return coef[i]; }
};    // end class Taylor
```

Point and Interval Taylor Operators

As foreshadowed by this example, we define an abstract data type for Taylor series and use operator overloading to define actions on objects of that type using previously defined floating-point and interval operations.

Design of Operators

In this section, we give prototypical operators for the direct propagation of Taylor coefficients such as might be called from code similar to that shown in Fig. 1. Direct propagation of values works by translating each operation into a call to the appropriate AD routine at compile time. Thus, simply compiling the source code for f and linking it with the overloaded operator routines creates a program that computes the Taylor coefficients of f at $t = t_0$. For illustration, we provide only a stripped-down prototype with operators required for the example $f(t) = e^t/(2 + t)$. We suppress issues of references and the like that are essential to the design of a useful class. See [6] for a description of a set of interval Taylor operators in Ada.

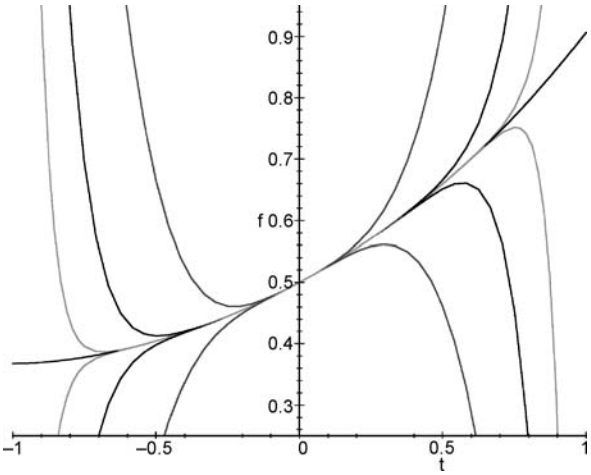
If instead, operators for AD_type record a code list, then an interpreter reads each node from the code list and calls the appropriate operator from class Taylor:

```
Taylor Operand[MemSize];
for (int i = 0; ¡ CodeSize; i++) {
   Node = getNextOperation ();
   switch (Node.OpCode) {
      case PLUS : Operand[Node.Result]
         = Operand[Node.Left]
            + Operand[Node.Right];
         break;
      …
      case EXP : Operand[Node.Result]
         = exp ( Operand[Node.Left] );
         break;
      …
   }
}
```

Use of Interval Operators

We have mentioned the possibility of working with interval values but not the significance of doing so. From equation (1) for an interval $\mathbf{t}$, and for all $t \in \mathbf{t}$,

$$f(t) \in \sum_{i=0}^{p} (f|t_0)_i (t - t_0)^i + \frac{1}{(p+1)!} \frac{d^{p+1} f(\mathbf{t})}{dt^{p+1}} (t - t_0)^{p+1}. \quad (3)$$



Automatic Differentiation: Point and Interval Taylor Operators, Figure 2
Taylor series enclosures of *f*

In a computer implementation, the summation is done in interval arithmetic to ensure enclosure. The series Taylor coefficients $(f|\mathbf{t}_0)_i$ are narrow intervals whose width comes only from outward rounding. The remainder term is the Taylor coefficient $(f|\mathbf{t})_i$, where the recurrence relations are evaluated in interval arithmetic. The series (3) can be used to bound the range of f, for validated quadrature [7], or for rigorous solution of ODEs [15]. For the example $f(t) = e^t/(2 + t)$, we repeat the sequence of computations of Equation (2) for the interval $\mathbf{t}_0 = [0, 0]$ and for $\mathbf{t} = [-1, 1]$:

$$\left((f|[0])_i\right) = \left(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{48}, \ldots\right),$$

$$\left((f|[-1,1])_i\right) = ([0.12, 2.72], [-2.59, 2.68], [-2.64, 4.04], \ldots).$$

Assembling these according to (3) yields enclosures for all $t \in [-1, 1]$:

$$\begin{aligned} f(t) &\in (f|[-1,1])_0 = [0.12, 2.72] \\ &\in (f|[0])_0 + (f|[-1,1])_1 (t - 0) \\ &= \frac{1}{2} + [-2.59, 2.68]t \\ &\in (f|[0])_0 + (f|[0])_1 (t - 0) \\ &\quad + (f|[-1,1])_2 (t - 0)^2 \\ &= \frac{1}{2} + \frac{1}{4}t + [-2.64, 4.04]t^2 \\ &\vdots \end{aligned}$$

To demonstrate the true power of this approximation technique, we plot the corresponding 5, 10, and 20 term enclosures in Fig. 2.

One-at-a-Time Coefficient Generation

The Taylor operators described the preceding section accept vectors of p Taylor coefficients for operands u and v and return Taylor coefficients for result x with *complexity* $O(p^2)$. However, for applications such as ODEs or order-adaptive quadrature, the entire operand series is not known, and we need to compute terms one at a time [6]. For example, for the DE

$$u' = f(t,u) = \frac{\exp(u)}{(2+t)}, \quad u(0) = 1,$$

initial condition $u(0) = 1$ implies

$$(u|0)_0 = 1,$$

and DE $u' = \exp(u)/(2+t)$ implies

$$(u|0)_1 = \frac{\exp(1)}{(2+0)} = \frac{e}{2},$$

$u'' = \frac{u'\exp(u)}{2+t} - \frac{\exp(u)}{(2+t)^2}$ implies

$$(u|0)_2 = \frac{e}{2}\frac{\exp(1)}{(2+0)} - \frac{e}{4} = \frac{e(e-1)}{4},$$

etc.

Successive terms can be computed by interpreting the code list for $f(t,u)$ repeatedly for series of increasing length for u. Each iteration of the automatic generation process yields an additional Taylor coefficient. Unfortunately, a simple implementation of Taylor operators has complexity $O(p^3)$ because already known coefficients of u' are recomputed. However, since the order of operations is the same in each iteration, we can increase the efficiency of the computations by storing intermediate results [6]. Each overloaded operator routine calls a memory allocation procedure that refers it to the next space in an array. If that space is empty, we store Taylor coefficient values for that variable. Otherwise, the space must contain the previously computed Taylor coefficients of that variable, which we can then use to more quickly compute the next coefficient in the set. With clever book-keeping, we compute p floating-point or interval-valued Taylor coefficients one at a time in $O(p^2)$ time.

Trade-Offs

We may strive for three goals when writing software for point and interval Taylor operations: storage space efficiency, time efficiency, and ease of use. These three factors are often at odds with each other.

Carefully implemented operator overloading provides an easy to use interface and provides reasonable time and space efficiency. We may achieve greater time and space efficiency by using source code transformation.

In conclusion, automatic differentiation through Taylor operators shows merit as a technique for computing guaranteed interval enclosures about a function f. Further efforts to refine this technique may provide us with a tool that handles multivariate functions, and runs significantly faster thanks to parallelization and improved optimization techniques.

See also

- ▶ Automatic Differentiation: Calculation of the Hessian
- ▶ Automatic Differentiation: Calculation of Newton Steps
- ▶ Automatic Differentiation: Geometry of Satellites and Tracking Stations
- ▶ Automatic Differentiation: Introduction, History and Rounding Error Estimation
- ▶ Automatic Differentiation: Parallel Computation
- ▶ Automatic Differentiation: Point and Interval
- ▶ Automatic Differentiation: Root Problem and Branch Problem
- ▶ Bounding Derivative Ranges
- ▶ Global Optimization: Application to Phase Equilibrium Problems
- ▶ Interval Analysis: Application to Chemical Engineering Design Problems
- ▶ Interval Analysis: Differential Equations
- ▶ Interval Analysis: Eigenvalue Bounds of Interval Matrices
- ▶ Interval Analysis: Intermediate Terms
- ▶ Interval Analysis: Nondifferentiable Problems
- ▶ Interval Analysis: Parallel Methods for Global Optimization
- ▶ Interval Analysis: Subdivision Directions in Interval Branch and Bound Methods
- ▶ Interval Analysis: Systems of Nonlinear Equations

- ▶ Interval Analysis: Unconstrained and Constrained Optimization
- ▶ Interval Analysis: Verifying Feasibility
- ▶ Interval Constraints
- ▶ Interval Fixed Point Theory
- ▶ Interval Global Optimization
- ▶ Interval Linear Systems
- ▶ Interval Newton Methods
- ▶ Nonlocal Sensitivity Analysis with Automatic Differentiation

References

1. Benary J (1996) Parallelism in the reverse mode. In: Berz M, Bischof Ch, Corliss G, Griewank A (eds) Computational Differentiation: Techniques, Applications, and Tools. SIAM, Philadelphia, pp 137–147
2. Berz M, Bischof Ch, Corliss G, Griewank A (eds) (1996) Computational differentiation: Techniques, applications, and tools. SIAM, Philadelphia
3. Bischof Ch, Carle A, Khademi PM, Mauer A, Hovland P (1994) ADIFOR: 2.0 user's guide. Techn Memorandum Math and Computer Sci Div Argonne Nat Lab ANL/MCS-TM-192
4. Bischof Ch, Haghighat MR (1996) Hierarchical approaches to automatic differentiation. In: Berz M, Bischof Ch, Corliss G, Griewank A (eds) Computational Differentiation: Techniques, Applications, and Tools. SIAM, Philadelphia, pp 83–94
5. Chang YF, Corliss GF (1994) ATOMFT: Solving ODEs and DAEs using Taylor series. Comput Math Appl 28:209–233
6. Corliss GF (1991) Overloading point and interval Taylor operators. In: Griewank A, Corliss GF (eds) Automatic Differentiation of Algorithms: Theory, Implementation, and Application. SIAM, Philadelphia, pp 139–146
7. Corliss GF, Rall LB (1987) Adaptive, self-validating quadrature. SIAM J Sci Statist Comput 8(5):831–847
8. Griewank A (1989) On automatic differentiation. In: Iri M, Tanabe K (eds) Mathematical Programming: Recent Developments and Applications. Kluwer, Dordrecht, pp 83–108
9. Griewank A (1991) The chain rule revisited in scientific computing. SIAM News 24(3):20 Also: Issue 4, page 8.
10. Griewank A, Corliss GF (eds) (1991) Automatic differentiation of algorithms: Theory, implementation, and application. SIAM, Philadelphia
11. Griewank A, Juedes D, Utke J (1996) ADOL-C: A package for the automatic differentiation of algorithms written in C/C++. ACM Trans Math Softw 22(2):131–167
12. Griewank A, Reese S (1991) On the calculation of Jacobian matrices by the Markowitz rule. In: Griewank A, Corliss GF (eds) Automatic Differentiation of Algorithms: Theory, Implementation, and Application. SIAM, Philadelphia, pp 126–135
13. Kearfott RB (1995) A Fortran 90 environment for research and prototyping of enclosure algorithms for nonlinear equations and global optimization. ACM Trans Math Softw 21(1):63–78
14. Kubota K (1996) PADRE2-Fortran precompiler for automatic differentiation and estimates of rounding error. In: Berz M, Bischof Ch, Corliss G, Griewank A (eds) Computational Differentiation: Techniques, Applications, and Tools. SIAM, Philadelphia, 367–374
15. Lohner RJ (1987) Enclosing the solutions of ordinary initial and boundary value problems. In: Kaucher EW, Kulisch UW, Ullrich C (eds) Computer Arithmetic: Scientific Computation and Programming Languages. Wiley and Teubner, Stuttgart, pp 255–286
16. Pryce JD, Reid JK (1997) AD01: A Fortran 90 code for automatic differentiation. Techn Report Rutherford–Appleton Lab RAL-TR-97xxx
17. Rall LB (1981) Automatic differentiation: Techniques and applications. Lecture Notes Computer Sci, vol 120. Springer, Berlin
18. Rall LB, Corliss GF (1996) An introduction to automatic differentiation. In: Berz M, Bischof Ch, Corliss G, Griewank A (eds) Computational Differentiation: Techniques, Applications, and Tools. SIAM, Philadelphia, pp 1–17
19. Rostaing N, Dalmas S, Galligo A (1993) Automatic differentiation in Odyssée. Tellus 45A:558–568
20. Shiriaev D (1996) ADOL-F: Automatic differentiation of Fortran codes. In: Berz M, Bischof Ch, Corliss G, Griewank A (eds) Computational Differentiation: Techniques, Applications, and Tools. SIAM, Philadelphia, pp 375–384
21. Yang W, Corliss G (1996) Bibliography of computational differentiation. In: Berz M, Bischof Ch, Corliss G, Griewank A (eds) Computational Differentiation: Techniques, Applications, and Tools. SIAM, Philadelphia, pp 393–418

Automatic Differentiation: Root Problem and Branch Problem

HERBERT FISCHER
Fakult. Math., Techn. University München, München, Germany

MSC2000: 65K05

Article Outline

Keywords

Automatic differentiation; Root problem; Branch problem

Automatic differentiation is a method in which a program for evaluating a function f is transformed into another program that evaluates both the function f and some of its derivatives. The key idea is the repeated use of the chain-rule for composing the derivatives of f from derivatives of parts of f. For more about automatic differentiation (AD), consult [2,3,5].

Proper combinations of differentiable functions produce differentiable functions. Some combinations of nondifferentiable functions also produce differentiable functions. Therefore the mere fact that a program defines a differentiable function is no guarantee that AD will work. Here we investigate two cases, where AD, applied to a program for a differentiable function, fails.

The *root problem* arises when a square-root is combined with other functions so that the resulting function is differentiable but the chain-rule is not applicable for certain arguments.

The *branch problem* arises when a program for evaluating a differentiable function f employs statements of the form $B(x)$ then $S1$ else $S2$, where x is from the domain of f, B is a Boolean function, and $S1$ and $S2$ represent subprograms. This reflects a piece-wise definition of the function f, and the derivative of one or the other piece may be quite different from the derivative of the function f.

Root Problem

An example that is typical of the root problem is shown in Table 1. The program P defines the function

$$f\colon\ \mathbb{R}^2 \to \mathbb{R}$$

with

$$f(x) = \sqrt{x_1^4 + x_2^4}.$$

This function is differentiable at any $x \in \mathbf{R}^2$, in particular $f'(0) = [0, 0]$. Standard AD (in the forward mode) transforms P into a program P' by inserting assignment statements for derivatives in proper places (see Table 2).

The program P' is supposed to compute $f(x)$ and $f'(x)$. But for $x = 0$ it does *not* compute the correct value $f'(0) = [0, 0]$, but rather it fails because of division by zero.

One can easily see that this failure is not limited to the forward mode, because the reverse mode encounters the same division-by-zero problem. Symbolic manipulation packages such as MAPLE also fail to produce $f'(0)$.

Automatic Differentiation: Root Problem and Branch Problem, Table 1
Program *P* for evaluating *f* at *x*

input: $x = (x_1, x_2) \in \mathbf{R}^2$		
y_1	$\leftarrow$	x_1
y_2	$\leftarrow$	x_2
y_3	$\leftarrow$	y_1^4
y_4	$\leftarrow$	y_2^4
y_5	$\leftarrow$	$y_3 + y_4$
y_6	$\leftarrow$	$\sqrt{y_5}$
$f(x)$	$\leftarrow$	y_6
output: $f(x)$		

Automatic Differentiation: Root Problem and Branch Problem, Table 2
Program *P′* for evaluating *f* and *f′* at *x*

input: $x = (x_1, x_2) \in \mathbf{R}^2$					
y_1	$\leftarrow$	x_1	y_1'	$\leftarrow$	$[1,0]$
y_2	$\leftarrow$	x_2	y_2'	$\leftarrow$	$[0,1]$
y_3	$\leftarrow$	y_1^4	y_3'	$\leftarrow$	$4y_1^3 \cdot y_1'$
y_4	$\leftarrow$	y_2^4	y_4'	$\leftarrow$	$4y_2^3 \cdot y_2'$
y_5	$\leftarrow$	$y_3 + y_4$	y_5'	$\leftarrow$	$y_3' + y_4'$
y_6	$\leftarrow$	$\sqrt{y_5}$	y_6'	$\leftarrow$	$\frac{1}{2\sqrt{y_5}} \cdot y_5'$
$f(x)$	$\leftarrow$	y_6	$f'(x)$	$\leftarrow$	y_6'
output: $f(x)$			output: $f'(x)$		

Automatic Differentiation: Root Problem and Branch Problem, Table 3
Program *Q* for evaluating *f* at *x*

input: $x \in D \subseteq \mathbf{R}^n$		
y_1	$\leftarrow$	$A(x)$
y_2	$\leftarrow$	$\sqrt{y_1}$
y_3	$\leftarrow$	$B(x, y_2)$
$f(x)$	$\leftarrow$	y_3
output: $f(x)$		

A more general setting for the root problem is shown in Table 3. Here, it is assumed that:

1) D_A is a nonempty open subset of $\mathbf{R}^n$;
2) the function A: $D_A \subseteq \mathbf{R}^n \to \mathbf{R}$ is differentiable;
3) D_B is a nonempty open subset of $\mathbf{R}^{n+1}$;
4) the function B: $D_B \subseteq \mathbf{R}^{n+1} \to \mathbf{R}$ is differentiable;
5) $D := \{x \in D_A : A(x) \geq 0, (x, A(x)) \in D_B\}$;
6) D is nonempty.

The program Q defines the function

$$f:\ D \subseteq \mathbb{R}^n \to \mathbb{R}$$

with

$$f(x) = B(x, \sqrt{A(x)}).$$

Standard AD (in the forward mode) transforms Q into a program Q'. The steps of Q' in evaluating $f'(x)$ can be seen in the formula

$$f'(x) = B_1(x, y_2) + B_2(x, y_2) \cdot \left(\frac{1}{2\sqrt{y_1}} \cdot y_1' \right),$$

where $[B_1(x, y_2), B_2(x, y_2)]$ is an appropriate partition of $B'(x, y_2)$. For $x \in D$ with $A(x) > 0$, the program Q' will produce $f'(x)$. And for $x \in D$ with $A(x) = 0$, the program Q' fails because of division by zero. The case in which $x^* \in D$ with $A(x^*) = 0$ is ambiguous. It says nothing about the existence of $f'(x^*)$. In this case, we distinguish the following four situations:

A) $f'(x^*)$ does not exist, for instance $n = 2$, $A(x) = x_1^2 + x_2^2$ and $B(x, y) = y$, $x^* = 0$.
B) A alone guarantees existence of $f'(x^*)$, for instance $n = 2$, $A(x) = x_1^4 + x_2^4$, $x^* = 0$.
C) B alone guarantees existence of $f'(x^*)$, for instance $B(x, y) = y^2$.
D) A and B together guarantee existence of $f'(x^*)$, for instance $n = 2$, $A(x) = x_1^2 + x_2^2$ and $B(x, y) = x_1 \cdot x_2 \cdot y$, $x^* = 0$.

What can be done to resolve the root problem?

The use of AD tools for higher derivatives may be helpful. Consider the simple case $n = 1$, $A \in \mathcal{C}^\infty$, $D_B = \mathbf{R}^{n+1}$, $B(x, y) = y$. So we have

$$D := \{x:\ x \in D_A,\ A(x) \geq 0\}$$

and f: $D \subseteq \mathbf{R} \to \mathbf{R}$ with $f(x) = \sqrt{A(x)}$.

Assume that for $x \in \mathbf{R}$ it can be decided whether or not $x \in D$, for instance by testing x in a program for evaluating A.

For $x^* \in D$, we require the value of the derivative $f'(x^*)$. Below, we list the relevant implications:

- $A(x^*) > 0 \Rightarrow f'(x^*) = \frac{1}{2\sqrt{A(x^*)}} \cdot A'(x^*)$.
- $A(x^*) = 0 \Rightarrow$ no answer possible.
- $A(x^*) = 0, A'(x^*) \neq 0 \Rightarrow f'(x^*)$ does not exist.
- $A(x^*) = 0, A'(x^*) = 0 \Rightarrow$ no answer possible.
- $A(x^*) = 0, A'(x^*) = 0, A''(x^*) \neq 0 \Rightarrow f'(x^*)$ does not exist.
- $A(x^*) = 0, A'(x^*) = 0, A''(x^*) = 0 \Rightarrow$ no answer possible.
- $A(x^*) = 0, A'(x^*) = 0, A''(x^*) = 0, A'''(x^*) \neq 0 \Rightarrow f'(x^*)$ does not exist.
- $A(x^*) = 0, A'(x^*) = 0, A''(x^*) = 0, A'''(x^*) = 0 \Rightarrow$ no answer possible.
- $A(x^*) = 0, A'(x^*) = 0, A''(x^*) = 0, A'''(x^*) = 0, A^{(4)}(x^*) > 0 \Rightarrow f'(x^*) = 0$.
- $A(x^*) = 0, A'(x^*) = 0, A''(x^*) = 0, A''(x^*) = 0, A^{(4)}(x^*) < 0 \Rightarrow f'(x^*)$ does not exist.
- $A(x^*) = 0, A'(x^*) = 0, A''(x^*) = 0, A'''(x^*) = 0, A^{(4)}(x^*) = 0 \Rightarrow$ no answer possible.

Let $n \in \{1, 2, 3 \ldots\}$ and $A^{(k)}(x^*) = 0$ for $k = 0, \ldots, 2n$.

- $A^{(2n+1)}(x^*) \neq 0 \Rightarrow f'(x^*)$ does not exist.
- $A^{(2n+1)}(x^*) = 0, A^{(2n+2)} > 0 \Rightarrow f'(x^*) = 0$.
- $A^{(2n+1)}(x^*) = 0, A^{(2n+2)} < 0 \Rightarrow f'(x^*)$ does not exist.
- $A^{(2n+1)}(x^*) = 0, A^{(2n+2)} = 0 \Rightarrow$ no answer possible.

For a nonstandard treatment of these implications see [6]. Of course in the general situation given in Table 3, the classification of cases is more problematic.

Branch Problem

A typical example for the branch problem is Gauss-elimination for solving a system of linear equations with parameters. For illustrative purposes, it suffices to consider two equations with a two-dimensional parameter x (see Table 4). Here, it is assumed that:

a) D is a nonempty open subset of $\mathbf{R}^2$;
b) the function M: $D \subseteq \mathbf{R}^2 \to \mathbf{R}^{2,2}$ is differentiable;
c) the function R: $D \subseteq \mathbf{R}^2 \to \mathbf{R}^2$ is differentiable;
d) $x \in D \Rightarrow$ the matrix $M(x)$ is regular.

The program GAUSS defines the function

$$F:\ D \subseteq \mathbb{R}^2 \to \mathbb{R}^2$$

with

$$M(x) \cdot F(x) = R(x).$$

Automatic Differentiation: Root Problem and Branch Problem, Table 4
Program GAUSS for evaluating *f* at *x*

	input: $x \in D$		
	M11	$\leftarrow$	$M_{11}(x)$
	M12	$\leftarrow$	$M_{12}(x)$
	M21	$\leftarrow$	$M_{21}(x)$
	M22	$\leftarrow$	$M_{22}(x)$
	R1	$\leftarrow$	$R_1(x)$
	R2	$\leftarrow$	$R_2(x)$
	IF M11 $\neq$ 0 THEN		
S1:	E	$\leftarrow$	M21 / M11
	M22	$\leftarrow$	M22 $-$ E $*$ M12
	R2	$\leftarrow$	R2 $-$ E $*$ R1
	F2	$\leftarrow$	R2 / M22
	F1	$\leftarrow$	(R1 $-$ M12 $*$ F2) / M11
	ELSE		
S2:	F2	$\leftarrow$	R1 / M12
	F1	$\leftarrow$	(R2 $-$ M22 $*$ F2) / M21
	output: $F(x)$ = (F1,F2)		

Since the matrix $M(x)$ is regular for $x \in D$, the program GAUSS and the function f are well-defined. Furthermore, the function f is differentiable.

Standard AD (in the forward mode) transforms GAUSS into a new program by inserting assignment statements for derivatives in proper places. The resulting program GAUSS' is also well-defined, and for $x \in D$ it is supposed to produce $F(x)$ and $F'(x)$.

Now choose

$$D = \{x \in \mathbb{R}^2 \colon\ 0 < x_1 < 2,\ 0 < x_2 < 2\}$$

and

$$M(x) = \begin{array}{|c|c|}\hline M_{11}(x) & M_{12}(x) \\ \hline M_{21}(x) & M_{22}(x) \\ \hline \end{array} = \begin{array}{|c|c|}\hline x_1 - x_2 & 1 \\ \hline 10 & x_1 + x_2 \\ \hline \end{array},$$

$$R(x) = \begin{array}{|c|}\hline R_1(x) \\ \hline R_2(x) \\ \hline \end{array} = \begin{array}{|c|}\hline 100(x_1 + 2x_2) \\ \hline 100(x_1 - 2x_2) \\ \hline \end{array}.$$

It is easy to see that D is a nonempty open subset of $\mathbf{R}^2$, that the functions M and R are differentiable, and that $M(x)$ is regular for $x \in D$.

GAUSS' produces

$$F'(1,1) = \begin{array}{|c|c|}\hline -40 & -90 \\ \hline 100 & 200 \\ \hline \end{array},$$

but the correct value is

$$F'(1,1) = \begin{array}{|c|c|}\hline -54 & -76 \\ \hline 170 & 130 \\ \hline \end{array}.$$

One can easily check that the wrong result is not limited to the forward mode, because the reverse mode yields exactly the same wrong result.

To better understand the situation we define

$$D_1 := \{x \colon\ x \in D, M_{11}(x) \neq 0\},$$
$$D_2 := \{x \colon\ x \in D, M_{11}(x) = 0\}.$$

The program GAUSS can be considered as a piecewise definition of the function F,

$$F(x) = \begin{cases} F(x) \text{ according to S1}, & \text{for } x \in D_1, \\ F(x) \text{ according to S2}, & \text{for } x \in D_2. \end{cases}$$

Normally, one is not too concerned about the domain of a function. But indeed in this case, we must be concerned.

Let $F|_{D_1}$ denote the restriction of F to D_1 and let $F|_{D_2}$ denote the restriction of F to D_2. Then, of course

$$F(x) = \begin{cases} (F|_{D_1})(x) & \text{for } x \in D_1, \\ (F|_{D_2})(x) & \text{for } x \in D_2. \end{cases}$$

The domain D_1 of the function $F|_{D_1}$ is an open set, $x \in D_1$ is an interior point of D_1, and hence

$$F'(x) = (F|_{D_1})'(x) \quad \text{for } x \in D_1,$$

and this is the value GAUSS' produces.

The domain D_2 of the function $F|_{D_2}$ is too thin, it has no interior points, and hence $F|_{D_2}$ is not differentiable. In other words, the function $F|_{D_2}$ does not provide enough information to obtain $F'(x)$ for $x \in D_2$. Thus GAUSS' *cannot* produce $F'(x)$ for $x \in D_2$. What GAUSS' actually presents for $F'(x)$ is the value for the derivative of another function, which is of no interest here. For more see [1].

In [4] it is claimed that the use of a certain branching function method makes the branch problem vanish.

This is true in certain cases, in our example the branching function method fails because it encounters division by zero. At least this suggests that something went wrong. For a partial solution to the branch problem, see [1] and for a nonstandard treatment of the branch problem, see [6].

A simple example of the branch problem is shown in the informal program

IF $x \neq 1$ THEN $f(x) \leftarrow x \cdot x$
ELSE $f(x) \leftarrow 1$.

This program defines the function

$$f\colon\ \mathbb{R} \to \mathbb{R} \quad \text{with } f(x) = x^2.$$

Of course, f is differentiable, in particular we have $f'(1) = 2$.

Standard AD software produces the wrong result $f'(1) = 0$. It is not surprising that symbolic manipulation packages produce the same wrong result. Here it is obvious that the else-branch does not carry enough information for computing the correct $f'(1)$.

Sometimes branching is done to *save work*. Consider the function

$$f\colon\ D \subseteq \mathbb{R}^n \to \mathbb{R}$$

with

$$f(x) = s(x) + c(x) \cdot E(x),$$

where D is an open set. The real-valued functions s, c, E may be given explicitly or by subroutines. Assume that $f(x)$ has to be evaluated many times for varying x-s, that $c(x) = 0$ for many interesting values of x, and that $E(x)$ is computationally costly. Then it is effective to set up a program for computing $f(x)$ as shown in Table 5.

Assume that the functions s, c, E are differentiable. Then f is differentiable too. For given $x \in D$ we ask for $f'(x)$.

Standard AD (in the forward mode) transforms SW into a new program by inserting assignment statements concerning derivatives. The resulting program SW' is well-defined, and for given $x \in D$ it is supposed to produce $f(x)$ and $f'(x)$.

Define the sets

$$D_1 := \{x\colon\ x \in D,\ c(x) \neq 0\},$$
$$D_2 := \{x\colon\ x \in D,\ c(x) = 0\}.$$

Automatic Differentiation: Root Problem and Branch Problem, Table 5
Program *SW* for computing *f*(*x*)

	input: $x \in D$		
	$c(x)$	$\leftarrow$	$\cdots$
	IF $c(x) \neq 0$ THEN		
S1:	$s(x)$	$\leftarrow$	$\cdots$
	$E(x)$	$\leftarrow$	$\cdots$
	$r(x)$	$\leftarrow$	$s(x) + c(x) \cdot E(x)$
	$f(x)$	$\leftarrow$	$r(x)$
	ELSE		
S2:	$s(x)$	$\leftarrow$	$\cdots$
	$f(x)$	$\leftarrow$	$s(x)$
	output: $f(x)$		

SW' works correctly to produce

$$f'(x) = r'(x) \quad \text{for } x \in D_1.$$

Looking at SW, it is tempting to assume:

$$f'(x) = s'(x) \quad \text{for } x \in D_2$$

and SW' actually follows this assumption. But it is clear that

$$f'(x) = s'(x) + E(x) \cdot c'(x) + c(x) \cdot E'(x) \quad \text{for } x \in D,$$

and in particular

$$f'(x) = s'(x) + E(x) \cdot c'(x) \quad \text{for } x \in D_2.$$

If $x \in D_2$, and if either $E(x) = 0$ or $c'(x) = 0$, then SW' produces the correct $F'(x)$, otherwise SW' fails.

See also

- Automatic Differentiation: Calculation of the Hessian
- Automatic Differentiation: Calculation of Newton Steps
- Automatic Differentiation: Geometry of Satellites and Tracking Stations
- Automatic Differentiation: Introduction, History and Rounding Error Estimation
- Automatic Differentiation: Parallel Computation

- ► Automatic Differentiation: Point and Interval
- ► Automatic Differentiation: Point and Interval Taylor Operators
- ► Nonlocal Sensitivity Analysis with Automatic Differentiation

References

1. Beck T, Fischer H (1994) The if-problem in automatic differentiation. J Comput Appl Math 50:119–131
2. Berz M, Bischof Ch, Corliss GF, Griewank A (eds) (1996) Computational differentiation: Techniques, applications, and tools. SIAM, Philadelphia
3. Griewank A, Corliss GF (eds) (1991) Automatic differentiation of algorithms: Theory, implementation, and application. SIAM, Philadelphia
4. Kearfott RB (1996) Rigorous global search: Continuous problems. Kluwer, Dordrecht
5. Rall LB (1981) Automatic differentiation: Techniques and applications. Lecture Notes Computer Sci, vol 120. Springer, Berlin
6. Shamseddine K, Berz M (1996) Exception handling in derivative computation with nonarchimedean calculus. In: Berz M, Bischof Ch, Corliss GF, Griewank A (eds) Computational Differentiation: Techniques, Applications, and Tools. SIAM, Philadelphia, pp 37–51