JOURNAL OF OPTIMIZATION THEORY AND APPLICATIONS: Vol. 26, No. 2, OCTOBER 1978

Current State of the Art of Algorithms and Computer Software for Geometric Programming[1,2]

R. S. DEMBO[3]

Communicated by M. Avriel

Abstract. This paper attempts to consolidate over 15 years of attempts at designing algorithms for geometric programming (GP) and its extensions. The pitfalls encountered when solving GP problems and some proposed remedies are discussed in detail. A comprehensive summary of published software for the solution of GP problems is included. Also included is a numerical comparison of some of the more promising recently developed computer codes for geometric programming on a specially chosen set of GP test problems. The relative performance of these codes is measured in terms of their robustness as well as speed of computation. The performance of some general nonlinear programming (NLP) codes on the same set of test problems is also given and compared with the results for the GP codes. The paper concludes with some suggestions for future research.

Key Words. Geometric programming, state of the art, signomial programhing, nonlinear programming, software.

1. Introduction

Ever since its inception, geometric programming (GP) has been somewhat of an outcast in the mainstream of mathematical programming

[1] An earlier version of this paper was presented at the ORSA/TIMS Conference, Chicago, 1975.

[2] This work was supported in part by the National Research Council of Canada, Grant No. A-3552, Canada Council Grant No. S74-0418, and a research grant from the School of Organization and Management, Yale University. The author wishes to thank D. Himmelblau, T. Jefferson, M. Rijckaert, X. M. Martens, A. Templeman, J. J. Dinkel, G. Kochenberger, M. Ratner, L. Lasdon, and A. Jain for their cooperation in making the comparative study possible.

[3] Assistant Professor of Operations Research, School of Organization and Management, Yale University, New Haven, Connecticut.

literature. Indeed, to this very day many prominent members of the mathematical programming community regard GP as a highly specialized curiosity and a "dead" area for research. The reason for this is twofold. Firstly, the historical development of computational procedures and duality theory in GP has, for the most part, taken place outside of the accepted state-of-the-art procedures in nonlinear programming (NLP). Secondly, there has been a failure on the part of many mathematical programmers to realize that a wide variety of important practical problems (for example, optimal engineering design problems) may be effectively modelled using geometric programming.

It is precisely because of its applicability to optimal engineering design that GP has been enthusiastically accepted by the engineering community. In fact, engineers have had almost an exclusive hand in the development of GP software. To some extent, which has been detrimental to GP in terms of improving its image in mathematical programming circles, mainly because GP software development has, as a result, lagged far behind general NLP software development. This is particularly true with regard to software for solving the linearly constrained dual GP. To be more precise, apart from work currently in progress (Ref. 1), to this author's knowledge there is no published algorithm for the linearly constrained dual that implements some specialized version of one of the latest numerically stable techniques for linearly constrained nonlinear programming, as discussed in Gill and Murray (Ref. 2). Matrix factorization is virtually unheard of in GP circles.

There has been, however, one fortunate byproduct of the above phenomenon. Since GP software developers were to a large extent not prejudiced by mathematical programming folklore, the implementation and testing of many algorithms that would otherwise have been shunned by "respectable" mathematical programmers has been carried out by GP researchers. A good example of this is Kelley's cutting-plane algorithm for convex programs. The method is purported to be at best geometrically convergent (see Wolfe, Ref. 3), numerically unstable, and definitely not an algorithm to be recommended for the solution of convex programming problems. Though theoretically there is definitely justification for the above hypothesis, there is no computational evidence to show that Kelley's algorithm does in fact perform worse (or better) than existing algorithms for convex programming.

It is shown in Section 5 and confirmed by Rijckaert and Martens (Ref. 4) that one of the most efficient[4] and robust[5] software packages currently

[4] In terms of standardized CPU time.

[5] By robust, we mean that the code will succeed in solving the majority of problems for which it was designed, to within prescribed tolerance limits.

available for solving GP's is a simple application of Kelley's cutting-plane algorithm. Furthermore, the algorithm has proved to be so successful that there are at least four known software packages based on it (Refs. 5–8) and in each case the respective authors have claimed excellent results.

The purpose of this paper is not only to summarize the current state-of-the-art of GP software but to identify the main sources of difficulty in designing such software and to point to directions for future research. To this end, we will discuss the following topics: solving GP's using general-purpose NLP software (Section 2); factors influencing the choice between primal-based and dual-based algorithms (Section 3); published extensions to signomial programming (Section 4); computational comparison of some of the above software (Section 5) and analysis of the results (Section 6); and conclusions and suggestions for future research (Section 7).

2. Solving GP's Using General-Purpose NLP Software

Contrary to popular belief, one cannot in general simply solve geometric programming problems using general NLP software without taking certain necessary precautions. To show this, we consider the primal and dual programs separately.

2.1. Solving Primal Geometric Programs. A primal geometric program (PGP) may be defined as the following nonlinear programming problem:

$$\text{(PGP)} \qquad \underset{x}{\text{minimize}} \qquad g_0(x) = \sum_{j \in J_0} c_j \prod_{i=1}^{m} x_i^{a_{ij}}, \tag{1}$$

$$\text{subject to} \qquad g_k(x) = \sum_{J \in J_k} c_j \prod_{i=1}^{m} x_i^{a_{ij}} \leq 1, \qquad k = 1, 2, \ldots, p, \tag{2}$$

$$x_i > 0, \qquad i = 1, 2, \ldots, m, \tag{3}$$

where (i) the sets J_k, $k = (0, 1, 2, \ldots, p)$, number terms in the objective function J_0 and the constraints J_k, $k = 1, 2, \ldots, p$, and (ii) the parameters c_j and a_{ij} are real constants with the restriction that $c_j > 0$ for

$$j \in \bigcup_{k=0}^{p} J_k = \{1, 2, \ldots, n\}.$$

The PGP has a number of special features that are important from the point of view of algorithmic design. They are the following.

(a) Derivatives of the objective and constraint functions of any order are available explicitly. Furthermore, they are relatively cheap to compute once function evaluations have been made. For example, in a posynomial with q terms, only q multiplications and q divisions are required to compute the first derivative of the function with respect to some variable, once the values of these terms are known.

(b) The problem is convex in the variables $\log x$, and thus may be solved by any convex programming algorithms that account for feature (c) below.

(c) Observe that the primal variables x_i, $i = 1, 2, \dots m$, are constrained to be *strictly positive*. Thus, the feasible region of PGP may not be compact and strictly speaking we should write "seek the infimum of" in place of "minimize" in the above programming problem. When a primal variable goes to zero or to some negative value, some of the terms in one or more posynomial functions might become undefined (for example, terms that contain the variable raised to a negative power).

The above characteristics of the primal indicate that any general NLP software may be used to solve GP problems provided that care is taken to avoid negative and, in some cases, zero values of the primal variables. One simple way of accomplishing this is to bound these variables from below using some small positive value. However, this approach may run into difficulty in cases where the GP is degenerate (see Duffin, Peterson, and Zener, Ref. 9) and some terms do go to zero in the optimal solution.

Since analytical derivatives are available and are relatively cheap to compute, it seems reasonable to expect gradient-based methods to be suitable for specialization to solving PGP problems. It is also not difficult to write a suitable front-end to any general gradient-based code that will read the term coefficients c_j, $j = 1, 2, \dots, n$, the exponent matrix a_{ij}, $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$, and numbers defining the sets J_k, $k = 0, 1, \dots, p$, and from this data compute function and gradient values. Furthermore, since under a simple transformation the primal is convex, the above discussion applies to convex programming software as well.

Geometric programming is also amenable to solution via separable programming techniques. The separable primal geometric program (SPGP) is given below.

$$\text{(SPGP)} \qquad \underset{y;z}{\text{minimize}} \quad \sum_{j\in J_0} c_j \exp(y_i), \tag{4}$$

$$\text{subject to} \quad \sum_{j\in J_k} c_j \exp(y_j) \le 1, \qquad k = 1, 2, \dots, p \tag{5}$$

$$y_j - \sum_{i=1}^{m} a_{ij} z_i = 0, \qquad j = 1, 2, \dots, n. \tag{6}$$

Equivalence of this separable formulation of the primal to the primal program PGP is easily recognized if we let

$$z_i = \log x_i, \qquad i = 1, 2, \ldots, m,$$

and define the constants c_j and a_{ij} and the sets J_k, $k = 0, 1, 2, \ldots, p$, as in PGP.

The SPGP formulation of the primal is a problem in $n + m$ variables as opposed to the m-variable formulation PGP. However, the addition of these variables results in a problem with a very special structure that may be exploited by special-purpose algorithms. Also, the above formulation would allow the solution of GP problems by widely available mathematical programming software systems such as MPSX (Ref. 10).

To this author's knowledge, there have only been two attempts at devising special-purpose algorithms for the solution of SPGP, namely Codes 1 and 3 (Appendix). Computational experience with these codes, both of which are based on linearization methods, is not encouraging. However, this should not be taken as conclusive evidence that solving SPGP is a poor way to approach the solution of GP problems. An algorithm more in keeping with the state-of-the-art in NLP would be to use a Newton-type method with an active constraint set strategy for maintaining feasibility of the nonlinear inequality constraints (5) and a projection method for handling the linear equality constraints. What makes this approach so attractive is the fact that the Hessian of a Lagrangian involving the constraints (5) would be a positive definite diagonal matrix, a fact that could surely be exploited computationally.

Another approach that is also in keeping with current practice in NLP would be to incorporate the nonlinear constraints (5) into an augmented Lagrangian and minimize this Lagrangian with respect to the linear equality constraints (6).

2.2. Solving Dual Geometric Programs. The dual geometric program (DGP) as defined by Duffin, Peterson, and Zener (Ref. 9) is the following linearly constrained nonlinear programming problem:

(DGP) maximize
$$v(\delta) = \prod_{j=1}^{n} (c_j/\delta_j)^{\delta_j} \prod_{k=1}^{p} \lambda_k^{\lambda_k}, \tag{7}$$

subject to
$$\sum_{j \in J_0} \delta_j = 1, \tag{8}$$

$$\sum_{k=1}^{p} \sum_{j \in J_k} a_{ij}\delta_j = 0, \qquad i = 1, 2, \ldots, m, \tag{9}$$

$$\delta_j \geq 0, \qquad j = 1, 2, \ldots n, \tag{10}$$

$$\text{where} \qquad \lambda_k = \sum_{j \in J_k} \delta_j, \qquad k = 1, 2, \ldots, p. \tag{11}$$

A subtle but important point is that the λ_k's are not treated as independent variables in the problem and the relationships in (11) are not treated as constraints, but rather as definitions. Whereas from a theoretical viewpoint this distinction might appear to be a case of semantics, it is extremely bad practice computationally. *In fact, it is the view of this author that the explicit elimination of the λ variables and the explicit formation of the reduced dual problem have been the singular most important factors in the failure to design efficient and numerically stable software for the dual problem.* These statements will be justified in the discussion below.

The reduced dual geometric program (RDGP) is obtained by eliminating $m+1$ *basic variables* from the program DGP and expressing them in terms of $d = n-(m+1)$ *nonbasic variables.*[6] This results in the following dual program in the variables r_i, $i = 1, 2, \ldots, d$:

(RDGP) $\underset{r}{\text{maximize}}$

$$\hat{v}(r) = K_o\left[\prod_{i=1}^{d} K_i^{r_i}\right]\left[\prod_{j=1}^{n} \delta_j(r)^{-\delta_j(r)}\right]\left[\prod_{k=1}^{p} \lambda_k(r)^{\lambda_k(r)}\right] \tag{12}$$

subject to

$$\delta_j(r) = b_j^{(0)} + \sum_{i=1}^{d} r_i b_j^{(i)} \geq 0, \qquad j = 1, 2, \ldots, n, \tag{13}$$

where

$$\lambda_k(r) \triangleq \left[\sum_{j \in J_k} b_j^0\right] + \sum_{i=1}^{d} r_i\left[\sum_{j \in J_k} b_j^{(i)}\right], \qquad k = 1, 2, \ldots, p, \tag{14}$$

$$K_i \triangleq \prod_{j=1}^{n} c_j^{b_j^{(i)}}, \qquad i = 0, 1, \ldots, d. \tag{15}$$

The reduced dual was strongly emphasized in Duffin, Peterson, and Zener (Ref. 9) as being a computationally useful formulation of the dual GP. Unfortunately, the theoretical exposition in Ref. 9 was taken far too literally by researchers who were attempting to design algorithms for the dual. Without exception, every dual-based code known to this author (see Appendix) *explicitly computes and stores* the basis vectors $b^{(i)}$, $i = 0, 1, \ldots, m$, using Gaussian elimination or some related technique. This is completely *contrary to accepted practice in nonlinear programming* (Ref. 2, Chapter 2). A far more stable approach numerically would be to carry out

[6] The quantity d is sometimes referred to as the degree of difficulty of a GP.

the reduction procedure implicitly by storing a matrix Z whose columns span the null space of the equality constraint coefficient matrix of DGP (Refs. 2-2). The matrix Z can be computed and stored explicitly or in product form by performing an orthogonal triangulation of the coefficient matrix. Incidentally, this same factorization can be used efficiently to find a stable least-square solution to the primal–dual optimality relationships (Ref. 11) in order to recover the optimal primal variables. For details on this approach the reader should consult (Refs. 1–2).

We feel that algorithms for the dual should be based on the following separable dual geometric program (SDGP), that is equivalent to DGP:

$$\text{(SDGP)} \quad \underset{\delta,\lambda}{\text{maximize}} \quad V(\delta, \lambda) = \sum_{j=1}^{n} \delta_j \log(c_j/\delta_j) + \sum_{k=1}^{p} \lambda_k \log \lambda_k, \tag{16}$$

$$\text{subject to} \quad A_o\delta = 1, \tag{17}$$

$$A\delta = 0, \tag{18}$$

$$B\delta - \lambda = 0, \tag{19}$$

$$\delta \geq 0. \tag{20}$$

The constraints (17)–(19) are a matrix representation of (9)–(11). Here,

$$\dim(A_0) = 1 \times n, \qquad \dim(A) = m \times n,$$

$$\dim(B) = p \times n,$$

where n = number of primal terms, m = number of primal variables, and p = number of primal constraints.

Notice that the above dual program has $n + p$ variables as opposed to n variables in the formulation given in (8)–(11). Also, the above formulation is a convex program whereas the original is not.

At first, it might seem ridiculous to increase both the number of variables and the number of constraints. However, this results in a problem with a very special structure. Firstly, the objective function is separable, and hence has a *diagonal Hessian* which can be utilized efficiently in a Newton-type algorithm for the dual problem (Ref. 1). The additional constraints should cause no consternation either. It is easy to construct algorithms which take implicit account of them. In fact, in Ref. 1 it is shown how the above problem may be solved by an algorithm in which the major computational effort involves either recurring an $m \times m$ *or* an $(n - m - 1) \times (n - m - 1)$ matrix at each iteration, depending on which is smaller.

The programs DGP, RDGP, and SDGP have a number of important characteristics, some of which preclude the direct application of NLP software for finding a numerical solution.

A major source of difficulty is the fact that the dual objective function is not differentiable with respect to the dual variables at points where they take on the value zero. To see this, for example, note that

$$\partial V/\partial \delta_j = \log(c_j/\delta_j) - 1, \tag{21}$$

which is undefined at $\delta_j = 0$. This fact (combined with the fact that, at an optimal solution, if for any $j \in J_k$, $\delta_j^* = 0$, then $\delta_j^* = 0$ for all $j \in J_k$) will cause general NLP software to fail if applied directly to these dual programs.

There are simple-minded remedies to the *nondifferentiability problem*, some of which are given below.

(a) Bound the variables such that $\delta_j \geq \epsilon > 0, j = 1, 2, \ldots, n$. This overcomes the differentiability problem, but causes other numerical problems such as an ill-conditioned Hessian at points near the solution. Also, most algorithms will tend to *zig-zag* between the constraints $\delta_j \geq \epsilon$, $j \in J_k$, until they have convinced themselves that $\delta_j^* = \epsilon$ for all $j \in J_k$.

More important is that simply zeroing all $\delta^* = \epsilon$ to $\delta^* = 0$, as is done in a large number of dual codes (see Appendix), will result in *infeasibilities* in the dual equality constraints (8) and (9). This could cause *poor estimates of the primal variables* to be computed using the primal–dual optimality conditions (Ref. 43). This author has long felt that this is precisely why it is often stated (see, for example, Ref. 12) that highly accurate dual solutions are required to obtain an even moderately accurate primal optimal solution. Thus, if the $\delta \geq \epsilon$ bounds are used, one must ensure that feasibility of the dual constraints is restored when the appropriate dual variables are zeroed.

The choice of an ϵ is also difficult. For example, in Problem 1 (Ref. 5) there are a substantial number of dual variables whose optimal value is less than 10^{-8}.

(b) A better approach than the one above is to approximate the jth term in the dual objective function by a quadratic[7] at points for which $\delta_j \leq \epsilon$. This is done as follows:

$$\delta_j \log(c_j/\delta_j) \simeq \alpha\delta_j^2 + \beta\delta_j, \qquad 0 \leq \delta \leq \epsilon, \tag{22}$$

where

$$\alpha = -1/\epsilon, \tag{23}$$

$$\beta = \log(c_j/\epsilon) + 1. \tag{24}$$

The advantage of the above approximation is that the objective function $V(\delta, \lambda)$ so defined will be continuous and differentiable, since α and β are chosen so that the derivatives and function values of $\delta_j \log(c_j/\delta_j)$ and

[7] The quadratic approximation presented here arose out of a series of discussions the author had with L. Lasdon and M. Saunders in an attempt to apply their general-purpose codes to the dual GP

$\alpha\delta_j^2 + \beta\delta_j$ are equal at $\delta_j = \epsilon$. Also, the two functions are equal at $\delta_j = 0$; however, their gradients differ at this point. The quadratic approximation has a gradient of β at $\delta_j = 0$, whereas as δ_j tends to zero the gradient of $\delta_j \log(c_j/\delta_j)$ tends to infinity.

Here, the choice of ϵ is not as difficult to make as in the bounding method discussed above. A balance must be struck between making ϵ too small, in which case the Hessian matrix will become ill-conditioned (since the contribution of this term will be the diagonal element $-2/\epsilon$), and making ϵ too large, in which case the gradient at $\delta_j = 0$ [namely, $\beta = \log(c_j/\epsilon) + 1$], will be too small to approximate the true behavior of the objective function at $\delta_j = 0$. A limited amount of experimentation with the method has shown that a value of $\epsilon = 10^{-5}$ seems to suffice.

It should be noted that the value of ϵ could be chosen dynamically by the algorithm under consideration. For example, in a Newton-type algorithm, ϵ could be set to its minimum value such that the condition number of the Hessian of the objective function $V(\delta, \lambda)$ would not be much larger than if this particular variable were not present. Since the Hessian is diagonal with elements $-\delta^{-1}$ and λ^{-1} (Ref. 13), the above criterion would result in an ϵ value that is not "very much" smaller than the smallest δ_j.

For algorithms based on an active constraint set strategy, the above quadratic approximation is only needed for variables that are exactly zero, as a means of providing the algorithm with approximate curvature information so that it can decide whether or not the active constraint $\delta_j = 0$ should be dropped from the basis at a particular iteration. Since

$$\beta = \log(c_j/\epsilon) + 1$$

will be positive for any $\epsilon < c_j e$, if for example we choose ϵ such that

$$\epsilon = \min\{0.9c_j e, 10^{-5}\}, \tag{25}$$

we are always assured that the gradient of the approximating quadratic will have the correct sign. Furthermore, this method will not be subject to the *zeroing problem* alluded to in (a) above.

We will not deal with the important topic of converting an optimal dual solution into an optimal primal solution, since this is covered in detail in Dembo (Ref. 11). It will suffice to say that the results of Ref. 11 indicate that any algorithm for the dual should compute the optimal Lagrange multipliers, ω_i^*, $i = 0, 1, 2, \ldots, m$, corresponding to the normality and orthogonality constraints (8) and (9), since they are related to the optimal primal variables x_i^* and optimal dual objective function V^* by

$$\omega_i^* = \log x_i^*, \qquad i = 1, 2, \ldots, m \tag{26}$$

$$\omega_0^* = 1 - V^*. \tag{27}$$

Thus, an optimal solution of the primal can be computed to the same degree of accuracy as the optimal dual multipliers. As is mentioned in Dembo (Ref. 11), (27) provides us with a useful check on the accuracy of the multipliers since V^* and ω_0^* may be computed independently. Also, a dual algorithm is not complete unless it provides for the case where the multipliers ω_i are not unique and a subsidiary problem (Ref. 11) might have to be solved in order to recover an optimal solution of the primal problem. Only one of the dual-based codes in the Appendix, namely CSGP, provides for such an eventuality.

3. Factors Influencing the Choice between Primal-Based and Dual-Based Algorithms

The question is often raised as to whether geometric programs should be solved using algorithms based on the dual program or by direct solution of the primal program. To ask whether the primal problem or the dual problem should be solved is an oversimplification. It would probably be more correct to ask *when* should the primal problem be solved as opposed to the dual, and vice versa. There are obvious cases where the dual program is a very much simpler problem than the corresponding primal (for example, a geometric program with zero degrees of difficulty). Similarly, it is easy to construct geometric programs where the primal problem may be very much easier to solve than the dual (for example, consider the minimization of a posynomial function of one variable with a large number of terms).

It is well known that linear programming (LP) is a special case of GP (see, for example, Duffin, Peterson, and Zener, Ref. 9). Therefore, as Templeman (Ref. 14) quite rightly points out, a special case of the above dilemma occurs in LP when one has to decide whether to solve a problem using primal-based or dual-based methods. For LP, the problem is much simpler and one can easily identify cases where a primal approach would be advantageous, and vice versa. Also, the same algorithm, namely the simplex method, may be applied to both the primal program and the dual.

In geometric programming, the decision as to whether to solve the dual program or the primal is a far less obvious one. For the general case, there seems to be no way out other than to draw on empirical evidence generated by computational comparisons such as the one described in Section 6 and also in Rijckaert and Martens (Ref. 4). There is however one special case of GP, other than LP, for which the same algorithm may be applied to both the primal and dual problems; hence, an a priori estimate of which of the two problems is easier to solve can be made with a fair degree of certainty.

Consider a pair of primal–dual GP problems in the case where there are no posynomial inequality constraints in the primal problem (this is often referred to as an unconstrained GP). Here, the primal program (UPGP) and dual program (UDGP) may both be written as convex, *separable* linearly constrained nonlinear programming problems:

$$\text{(UPGP)} \quad \underset{y,\omega}{\text{minimize}} \quad \log \sum_{j=1}^{n} c_j \exp(y_j), \tag{28}$$

$$\text{subject to} \quad y - A^T\omega = 0; \tag{29}$$

$$\text{(UDGP)} \quad \text{maximize} \quad \sum_{j=1}^{n} \delta_j \log(c_j/\delta_j), \tag{30}$$

$$\text{subject to} \quad \sum_{j=1}^{n} \delta_j = 1, \tag{31}$$

$$A\delta = 0, \tag{32}$$

$$\delta \geq 0. \tag{33}$$

Since the cost of function and derivative evaluations is roughly the same for UPGP and UDGP, the difference in computational effort required to solve them will be a function of the relative sizes (number of variables and constraints) of these dual programs, if the *same* algorithm is applied to both. In both cases, the equality constraints may be handled implicitly using projection matrices; however, the primal problem (UPGP) does have a slight edge over the dual, in that it does not possess inequality constraints. Also, the primal objective function is differentiable at all points in the primal feasible region, whereas the dual is not.

For constrained GP problems, the tables are turned. The primal problem (SPGP) is subject to the nonlinear inequality constraints (5), whereas the dual (SDGP) remains a linear constrained problem. The author's feeling is that, with few exceptions, the nonlinear inequality constraints of the primal problem make the dual (SDGP) a more attractive problem to solve, *even when the degree of difficulty* $(n-m-1)$ *is very large.* The reason for this is twofold. Firstly, nonlinear constraints are at least an order of magnitude more difficult to deal with than are linear constraints. Secondly, and this is what most researchers in the area of GP seem to be unaware of, the dual problem (SDGP) may be solved by a Newton-type algorithm where at each iteration the main amount of work involved lies in solving a square system of equations, whose dimension is either equal to the degree of difficulty of the problem $(n-m-1)$ or to the number of primal variables (m), depending on which of these two quantities is smaller. Details of such an algorithm are given in Dembo (Ref. 1).

4. Extensions to Signomial Programming

A signomial programming (SP) problem is a program of the form given in (1)–(3), (namely, PGP), in which the term coefficients may take on any real value. This type of programming problem is sometimes referred to as an algebraic program.

In general, SP problems are nonconvex, and the elegant duality theory associated with posynomial programs does not carry over to signomial programs. Attempts have been made at defining pseudo-dual problems (see Ref. 15 and Ref. 16, Chapter 5); however, the use of a pseudo-dual program as a vehicle for computing an optimal solution to a primal program PSP (that is, PGP where some $c_j < 0$) is not to be recommended unless certain safeguards are incorporated into the algorithm. This is because a local maximum of the pseudo-dual program might correspond to a *local maximum* of the primal (recall that PSP is a nonconvex *minimization* problem). Thus, if a dual approach is used to solve PSP, the algorithm must contain a built-in checking procedure to ascertain whether or not the computed stationary point of the primal problem is in fact a local minimum. If it is not, then the algorithm should invoke an alternative procedure until convergence to a local minimum is achieved. To the author's knowledge, *none* of the existing codes that solve SP problems via a pseudo-dual approach (see dual Codes 2 and 11 in the Appendix) have built-in safeguards.

Apart from the *pseudo-dual approach*, there are essentially three different ways in which algorithms for SP problems have been designed. These are discussed below.

4.1. Complementary Algorithm of Avriel and Williams. (*Ref. 17*). This algorithm solves an SP problem by solving a sequence of GP approximations. Each GP approximation is computed using posynomial condensation (Ref. 17). It was noted by Dembo (Ref. 17, also reported in Ref. 18) that the complementary algorithm may be accelerated if an exterior method[8] is used to solve the approximating GP primal.

The codes GGP, QUADGP, SIGNOPT, GEOEPS, and GEOLP (see Appendix) all use this approach to solving SP problems. Unfortunately, a feasible point is required to initiate the algorithm, and this in general means that a Phase 1 routine has to be incorporated into the code (Ref. 20).

4.2. Harmonic Method of Duffin and Peterson. (*Ref. 20*). Here too, the SP problem is solved by solving a sequence of approximating GP

[8] By *exterior method*, we mean that the sequence of points converging to an optimal solution remains infeasible until the solution is reached.

problems. Each *harmonic approximation* of Duffin and Peterson can be shown to be weaker than the complementary approximation (see, for example, Ref. 16) and in general results in an approximate GP problem whose dual has a larger degree of difficulty than in the complementary approximation. The harmonic approach, however, does have one important property that, to the author's knowledge, has never been exploited computationally. That is, the exponent matrix remains constant for every GP problem in the approximating sequence, which is not the case in the complementary algorithm. Without making specific use of this property, any algorithm based on the harmonic method will be *dominated* by one based on the condensation method, other things being equal. Jefferson's code GPROG (see Appendix) uses the harmonic approach. Bradley's code QUADGP (see Appendix) has an option to use either the harmonic algorithm or the complementary algorithm. In both these codes, the invariance of the exponent matrix is *not* used. Bradley (Ref. 21) demonstrates the obvious superiority of the condensation procedure on a number of test problems.

The same remarks in Section 4.2 regarding feasible starting points apply here also.

4.3. Direct Solution of the Signomial Programming Problem. To date there has only been one code developed to solve the SP problem directly. Rijckaert and Martens (Ref. 22) solve the nonlinear equations corresponding to the Kuhn–Tucker first-order necessary conditions for optimality of the primal SP problem (PSP). However, they do not indicate whether they have built in safeguards to ensure that they compute a local minimum of PSP and not a stationary point or a local maximum.

4.4. Convergence of the Complementary and Harmonic Algorithms. Unfortunately, it has been this author's experience that the complementary (and hence the harmonic) algorithm tends to converge linearly for most problems.[9] This makes it a poor method to use in a GP code, especially for problems with relatively few negative terms (see Test Problem 4A, Ref. 23). It is for this reason that the author feels that the best way to solve signomial problems in general is by a direct attack on the primal program written in separable form (that is, SPGP where the coefficients c_i are not all positive). This will surely be more efficient than solving a sequence of similar-sized GP problems. Some justification for this statement is given in the next section.

[9] A computational comparison of these algorithms is given in Ref. 24.

5. Numerical Comparison of Some GP Codes

This section summarizes the results of a Colville-type study (Ref. 25) that was undertaken by this author in the period from June 1974 to July 1976. The ambitious aims of the study were: (i) to identify which available GP codes were obviously superior to other in terms of computational efficiency; (ii) to test the robustness of various approaches; (iii) to isolate a good set of problems that would test various critical aspects of GP algorithms; and (iv) to answer the embarrassing question: are specialized GP codes more efficient in the solution of GP problems than good general-purpose NLP codes?

Only one of the above aims was achieved to any degree of satisfaction, namely, the study did produce a good set of test problems (Ref. 23). Our justification for this conclusion comes from the feedback from people who have actually attempted to solve these problems.[10] Their general conclusion is that the problem set contains a good mix of well-scaled, badly-scaled, easy, and difficult problems and also captures the inadequacies of various algorithmic approaches to GP. We will discuss the particular nature of each of the problems later, when the computational results are analyzed.

There is one major drawback, however, to conducting a comparative study based on a hand-picked sample of test problems. That is, very little in the way of inferences can be made as to the relative performance of the codes in question on a *different* set of problems (Ref. 27). It is precisely this sort of inference that one wishes to make; namely, since code X did better than code Y on the test problems, this will be true for a larger class of problems. Unfortunately, the methodology for designing comparative studies in mathematical programming is primitive, to say the least, and has only recently been considered as a serious topic for research.

The study reported here was conducted in the following way.[11] An attempt was made to obtain the participation of all authors of GP software that were known to the author at the time the study was conducted. Each participant was informed that the problems would be run at the particular author's home institution, on the computer for which the code was originally developed. In addition, Colville's standard timer (Ref. 25) was supplied in an attempt to standardize the CPU timing results and stopping criteria and

[10] The names and addresses of people other than those mentioned in this study who have solved the problems in (Ref. 23) is available on request from the author.

[11] As is mentioned in the text, the author is fully aware of the drawbacks of such a study (see Refs. 27–28) and cautions the reader to be wary of any conclusions drawn on the basis of the results presented here. In particular, the use of a standardized timing routine may in extreme cases make timing results meaningless.

tolerances were specified as reported in (Ref. 23). Where possible, an attempt was made to standardize the use of compilers. For example, participants using IBM machines were requested to use the FORTG compiler when compiling the timing program. In some cases, participants did not adhere strictly to the rules, and this has added some additional noise to the results.

To some extent, the experimental design did allow the participants to tune their codes to the set of problems in Ref. 23, and so the timing results in Table 4 represent the best results of each code on this set of problems, without a major alteration to the code design itself. Actually, in two cases (Refs. 8, 29) the participants redesigned the Phase I section of their codes as a result of repeated failures on some of the test problems.

An attempt was also made to include CPU timing results for some recently developed general-purpose NLP codes in order to compare with GP codes tested. Table 1 summarizes the characteristics of the general purpose NLP codes that participated in the study.

The GP codes that participated were SIGNOPT, GEOEPS-GEOGRAD, GEOLP, GPKTC, GPROG, and GGP. A summary of their main features is given in the Appendix. Details of the computer configuration and other aspects of the timing runs for the GP participants are given in Table 2.

Important characteristics of the test problems are summarized in Table 3.

The standardized times (actual CPU time divided by Colville standard time) for all participating codes on all test problems are given in Table 4. Blank entries in the table indicate that the code did not converge to a solution. The participants were all asked to solve the problems to within the convergence and constraint tolerance criteria specified in Table 3 (see Dembo, Ref. 23). Whereas most participants met the crucial primal feasibility criterion, stopping criteria and other internal tolerances were not equivalent from one code to the next. This does add an additional degree of uncertainty in interpreting the results. Only one of the GP codes, namely GPROG, did not solve problems to the required feasibility tolerances. The results in Table 4 for GPROG refer to a primal feasibility tolerance of 0.01, whereas the required feasibility tolerances specified in Ref. 23 range from 10^{-4} to 10^{-6}.

In an attempt to measure the sensitivity of various codes to achieve different degrees of primal feasibility, two levels of feasibility tolerances were specified in Dembo (Ref. 23). Since only a few of the participants responded to a request for runs at both tolerance levels, these results are not reported here. They are reported, however, for the code GGP in Ref. 23.

Table 1. Characteristics of general-purpose NLP codes tested.

Code	Algorithm	Language	Compiler	Participants*	Institution	Computer configuration	Colville standard time
GRG	Generalized reduced gradient (Ref. 30)	FORTRAN	WATFIV (NOCHECK)	Lasdon Ratner Jain	Stanford University (currently at Case Western Reserve University)	IBM 370/168	16.83 sec using WATFIV (NOCHECK)
COMET	Penalty function method	FORTRAN	RUN	Himmelblau	University of Texas at Austin	CDC 6600	20 secs
GREG	Generalized reduced gradient (Ref. 31)	FORTRAN	RUN	Himmelblau (Abadie/Guigou)	University of Texas at Austin	CDC 6600	20 secs
GPM/ GPMNLC	Extended gradient projection method (Ref. 32)	FORTRAN	RUN	Himmelblau (Kreuser/Rosen)	University of Texas at Austin	CDC 6600	20 secs
GAPF-QL	Penalty function method modified to avoid ill-conditioning of Hessian	FORTRAN	RUN	Himmelblau (Newell)	University of Texas at Austin	CDC 6600	20 secs

* Names in parentheses refer to original authors of the code.

Table 2. Characteristics of GP codes participating in study.

Code	Language	Compiler	Participants	Institution	Computer configuration	Colville standard time
SIGNOPT	FORTRAN	Not specified	Templeman	University of Liverpool	ICL 1906A CD6 7600 combination	2.22 secs
GEOPS/GEOGRAD	FORTRAN	FORTH	Dinkel	Pennsylvania State University	IBM 370/168	8.75 secs (FORTG)
GEOLP	FORTRAN	(OPT = 2)	Kochenberger			
GPKTC	FORTRAN	FORTH (OPT = 2)	Rijckaert Martens	Katholieke Universiteit Leuven	IBM 370/158	27.73 secs (FORTG)
GPROG	FORTRAN and COMPASS	Not specified	Jefferson	University of New South Wales Australia	CYBER 72	30 secs (approx.)
GGP	FORTRAN	FORTH (OPT = 2)	Dembo	University of Waterloo (currently at Yale University)	IBM 370/158	25.30 secs (FORTG)

Table 3. Test problem characteristics (Ref. 23).

					Required tolerances	
Problem	Type	Variables	Constraints*	Terms*	EPSCON†	EPSCGP‡
1A	GP	12	3	31	10^{-6}	10^{-4}
1B	GP	12	3	31	10^{-6}	10^{-4}
2	SP	5	6	32	10^{-5}	10^{-4}
3	SP	7	14	58	10^{-5}	10^{-4}
4A	SP	8	4	16	10^{-5}	10^{-4}
4B	SP	8	4	16	10^{-5}	10^{-3}
4C	GP	9	5	15	10^{-5}	10^{-4}
5	SP	8	6	19	10^{-5}	10^{-4}
6	SP	13	13	53	10^{-6}	10^{-4}
7	SP	16	19	85	10^{-5}	10^{-3}
8A	GP	7	4	18	10^{-6}	10^{-4}
8B	GP	7	4	18	10^{-6}	10^{-4}
8C	GP	7	4	18	10^{-6}	10^{-4}

* Does not include simple bounds on variables.
† If $g_k(x) \leq 1 + \text{EPSCON}$, $k = 1, 2, \ldots, p$, then the constraints are considered to be satisfied.
‡ Convergence tolerance, see Footnote 12.

6. Analysis of Comparative Study Results

We will analyze the results presented in Table 4 in terms of the original aims of the study. First, results for GP codes are analyzed. Later, these are compared with the results for NLP codes.

6.1. Results for GP Codes

(a) *Efficiency.* By glancing at the bracketed numbers in Table 4, we see immediately that the GP codes GPKTC and GGP stand out among the rest in terms of the speed with which the problems were solved. The code GPKTC was within 10% of the fastest standardized time for 7 of the 13 test problems, whereas GGP was within 10% of the fastest time in 8 of 13 cases. A closer scrutiny reveals that GGP seemed to do better on the larger problems and on problems with many simple bounding constraints. This is entirely consistent with the findings in Ref. 16.

An interesting result shown in Table 4 is that GGP consistently dominates GEOLP in terms of standardized times. This result is interesting because these two codes are based on the *identical mathematical algorithm.*

Table 4. Standardized times for test problems.

Problem	Type	Starting point (primal)	SIGNOPT (GP)	GRG (NLP)	COMET (NLP)	GREG (NLP)	GPM/ GPMNLO (NLP)	GAPF-QL (NLP)	GEOPS GEOGRAD (GP)	GEOLP (GP)	GPKTC (GP)	GPROG (GP)	GGP (GP)
1A	GP	NF						4.651	0.5648	1.1166	[0.0584]	0.1276	0.274
1B	GP	NF		[0.056]	1.5262	0.1197		0.5155	0.5648	1.1166	[0.0554]		0.271
2	SP	F	2.6259	0.010		0.2444	0.1349*	0.442*	0.0405	0.0325	0.0454	0.1937	[0.002]
3	SP	F			0.7497	0.1165		0.5820		0.2992	[0.0868]		[0.082]
4A	SP	NF	1.1524	0.038	0.0544	0.0503	0.1193*	0.0322		1.7721	[0.0186]	0.7730	0.280
4B	SP	NF	1.1524	0.034	NR	NR	NR	0.0329		NR	[0.0183]		0.132
4C	GP	NF	0.1650	0.052	0.0795	0.0818	0.0958	0.0404	0.0537	0.0843	0.0244		[0.021]
5	SP	NF	1.4330	0.049	0.1301*	0.0709	0.5404	1.0247	0.5057	0.2817	[0.0323]	1.6220	0.125
6	SP	NF				0.5747				0.5390	[1.3545]		[0.327]
7	SP	NF						2.4454		0.4581	0.6864		[0.240]
8A	GP	NF	3.8943	0.282	>3	0.4479	0.1684		[0.0877]	0.1695	0.2155		[0.095]
8B	GP	NF	6.0576	0.194	1.8736	0.2864	0.2942		0.1406	0.1559	0.1861		[0.095]
8C	GP	NF	31.782	0.443	0.9045	0.3043	0.2577		0.1731	0.1274	0.1405		[0.079]

Problem type: GP = Posynomial program, SP = Signomial program.
Starting Point: NF = Not feasible, F = Feasible.
Bracketed numbers indicate standardized times that are within 10% of the best time. Blank entries indicate that the code was unable to solve the particular problem.
Asterisk indicates that the error in the computed value of an optimal solution was between 5% and 10%.
NR = Not reported.

This underscores the fact that what we are testing in this study is the performance of codes and not algorithms.

The results, however, do shed some light on the underlying algorithms. Consider for example Problems 4A, 4B, and 4C. Problems 4A and 4B are both signomial problems in 8 variables, 4 signomial constraints, and 7 degrees of difficulty. The sole difference between these problems is that Problem 4A has a tighter convergence tolerance (EPSCGP $= 10^{-4}$) than Problem 4B (EPSCGP $= 10^{-3}$).[12] In Ref. 26 and Table 3, it is shown that GGP requires more than twice as much CPU time for Problem 4A as for Problem 4B. This is indicative of the sensitivity of the Avriel–Williams (Ref. 17) algorithm to this commonly used termination tolerance criterion. The code GPKTC solves signomial problems directly, and Table 4 shows that the algorithm used is not at all sensitive to such a criterion; that is, GPKTC requires roughly the same amount of CPU time to achieve optimality in Problems 4A and 4B.

An even more dramatic indication of how inefficient the Avriel–Williams procedure (Ref. 17) can be is found by examining the relative performance of GGP and GPKTC on Problems 4A and 4C. Problem 4A as mentioned above is a signomial problem. However, what was not mentioned was that *only two of* 16 *terms in the problem have negative coefficients.* Thus, Problem 4A is *almost posynomial.* Problem 4C is a GP approximation of 4A, which is obtained by *condensing-out* these two negative terms (Ref. 19). Therefore, Problem 4C is constructed to be a very similar-sized and similar-structured problem to Problem 4A but without any negative terms. For this problem (and using the same tolerance criteria as for Problem 4A) GGP is 14 times faster than for Problem 4A and is slightly faster than GPKTC. Note also that GPKTC needs approximately the same amount of CPU time for all three problems and, not unexpectedly, all the GP codes using the Avriel–Williams algorithm (namely, SIGNOPT and GEOLP) exhibit the same type of behavior as GGP on these three problems.

It is difficult to compare GEOEPS/GEOGRAD, GPROG, and SIGNOPT, since they have few data points in common. Very roughly speaking, GEOEPS/GEOGRAD seems to be faster than GPROG and SIGNOPT; when it converges, it is fairly competitive with GPKTC and GGP. The standardized times for SIGNOPT show that it can exhibit extremely slow convergence (see Problem 2, for example). The reader is

[12] The tolerance EPSCGP is defined in Ref. 23 by

$$|[g_0(x^i) - g_0(x^{i-1})]/g_0(x^{i-1})| \leq \text{EPSCGP},$$

where $g_0(x^i)$ is the objective function value at the *i*th *ϵ-feasible point* x^i and x^i, $i = 1, 2, \ldots$, is the sequence of points that converge to a minimum of the signomial problem. That is, if the above inequality is satisfied, then x^i is assumed to be in the vicinity of a stationary point.

asked to interpret this with caution, since it is this author's feeling that the inaccuracies introduced by using standardized timers penalize participants with very fast computers. The above tests using SIGNOPT were run on a CDC 7600 with a standardized time of 2.2 seconds. This is by far the fastest computer in the study. Fortunately, this effect is not present when comparing GGP and GPKTC, since IBM 370/158 computers were used in both cases.

(b) *Robustness.* The number of blank entries in Table 4 shows that, despite the *tuning effect* mentioned in Section 5, many codes were simply unable to solve certain problems. The most difficult problems (in terms of the number of codes that failed to solve them) were Problems 1A, 6, and 7. Problems 6 and 7 were the largest in the study.

Among the GP codes, GEOLP, GPKTC, and GGP stand out as being very robust on this set of test problems, since they never failed to converge to a solution. This is confirmed once again by the independently conducted study in Ref. 4. The codes GEOEPS/GEOGRAD and SIGNOPT each failed to converge for 5 of the 13 problems, and GPROG was the least robust, with only 4 successes out of a total of 13 problems.

6.2. Results for General NLP Codes

(a) *Efficiency.* Of the NLP codes tested, GRG ranks as the most efficient in terms of standardized CPU time and appears to do consistently better than GREG. This result is misleading and would probably be *reversed* if the timing runs for GRG were to be carried out using the FORTH (OPT = 2) compiler. The reason for this statement is graphically illustrated in Ref. 55. In Ref. 55, Problems 4C and 5 were solved by GRG using the FORTH (OPT = 2) compiler. The Colville timer, also run using FORTH (OPT = 2), yielded a standard time of 3.91 seconds for the IBM 370/168 [as compared with 16.83 using WATFIV (NOCHECK)]. The standardized times thus computed for Problems 4C and 5 were 0.109 and 0.069, as opposed to 0.052 and 0.049 using WATFIV (NOCHECK)!

The NLP codes all seemed to do badly on the most difficult problems in the set namely, Problems 1A, 6, and 7. The only NLP method that was able to solve the badly-scaled chemical equilibrium problem (Ref. 23) was the penalty method GAPF-QL. This is probably because care was taken in the coding of GAPF-QL to account for ill-conditioning (see Appendix). The only NLP code to solve Problem 6 was Abadie and Guigou's reduced gradient code GREG (Ref. 28). Since the majority of the effort in Problem 6 lies in finding a feasible solution (Phase 1), this *might* indicate that, among the general NLP codes tested, GREG has the best Phase 1 component.

On first attempt, the NLP code GAPF-QL converged only on Problems 3, 4, and 7 (Ref. 33). The values given in Table 4 therefore in some sense show the best results that could be attained by GAPF-QL on these problems and are a result of a number of trial runs.

The code GPM/GPMNLC seemed to produce the most inaccurate results; for the 3 of 7 problems solved, there was more than a 5% error in the *optimal solution value* computed. The fact that errors of this size appear in the results for COMET, GPM/GPMNLC, and GAPF-QL leads the author to suspect that the constraint tolerances specified in Dembo (Ref. 23) were not strictly adhered to when these solutions were computed. However, without direct information to the contrary, the author will assume that the values in the table are CPU times computed with the specified tolerances.

(b) *Robustness.* Among the general NLP codes, the two generalized reduced gradient codes that were tested, namely GRG and GREG, proved to be the most robust. The code GRG failed on 3 of 13 problems, whereas GREG only failed on 2 of the problems. The codes COMET and GAPF-QL each failed on 4 problems, and GPM/GPMNLC was the worst, with 5 failures and a record of inaccurate solutions for the problems that it did solve.

6.3. Performance of GP Codes versus General NLP Codes

(a) *Efficiency.* In order to demonstrate the relative efficiencies of the two sets of codes, standard times for the *best* two GP codes (GPKTC and GGP) are compared with the best time computed by any one of the 5 general NLP codes. This comparison is given in Table 5.

Table 5. Standardized times of general NLP codes versus GP codes.*

Problem	Type	Variables	Nonlinear constraints	$\frac{\text{Best NLP}}{\text{GPKTC}}$	$\frac{\text{Best NLP}}{\text{GGP}}$	$\frac{\text{Best NLP}}{\text{Best GP}}$
1A	GP	12	3	79.6	17.0	79.6
1B	GP	12	3	1.0	[0.2]	1.0
2	SP	5	6	[0.2]	5.0	5.0
3	SP	7	14	1.3	1.4	1.4
4A	SP	8	4	1.7	[0.1]	1.7
4B	SP	8	4	1.8	[0.2]	1.8
4C	GP	9	5	1.7	1.9	1.9
5	SP	8	6	1.5	[0.4]	1.5
6	SP	13	13	1.6	1.8	1.8
7	SP	16	19	1.4	4.0	4.0
8A	GP	7	4	[0.8]	1.8	1.8
8B	GP	7	4	1.0	2.0	2.0
8C	GP	7	4	1.8	3.3	3.3

* Bracketed numbers refer to problems where the best standardized NLP time was better than the standardized GP time in question.

An interesting observation is that the best GP time is *always better* than the best general NLP time. Also, each of the GP codes GPKTC and GGP outperform the best NLP result for the vast majority of the test problems. As expected, the standardized times for GGP on Problems 4A and 4B are considerably worse than those for the best NLP time, since these problems were specifically designed to demonstrate the worst features of the Avriel–Williams algorithm used in GGP.

(b) *Robustness.* The best GP codes were very much more robust on these problems than their NLP counterparts. There were, however, GP codes that did not perform as well as most of the NLP codes.

7. Conclusions and Suggestions for Future Research

Despite the drawbacks associated with the comparative study in Section 6, it is possible to draw some inferences regarding the behavior of the various codes. These inferences hold with some (imprecise) degree of certainty for the particular problem set tested. It is encouraging to note that *all* the conclusions drawn in this section are supported by the results of an independent comparative study (Ref. 4), done in a more controlled setting and using a different set of problems. In the Rijckaert–Martens study (Ref. 4), the two best GP codes (in terms of computational efficiency and robustness) were found to be GGP and GPKTC.[13] The same conclusion is evident in the results of Section 6.

Two of the general NLP codes in this study, namely GRG and GREG, are known to be among the best available general-purpose NLP codes. In the Colville study (Ref. 25) for example, Abadie and Guigou's GREG proved to be one of the most efficient and robust codes tested. Whereas these NLP codes appeared to be fairly efficient in solving our GP test problems, neither of them converged on the badly-scaled Problem 1A or the largest problem in the study, Problem 7. Also, their combined best time was equal to the best GP time for Problem 1B and was between 1.4 and 5 times slower for the remaining problems that they managed to solve.

The above result is an indication to this author that there is a need for specialized GP codes, if not for any other reason but robustness alone. This author comes to this conclusion despite the fact that he feels that even the best GP codes available (GPKTC and GGP) are in many ways primitive and lag far behind what could be achieved by specilizing current NLP technology to GP.

[13] It should be noted that the version of GGP referred to in Ref. 4 is an earlier and less efficient version than the one used here.

The GP problems SPGP and SDGP have a structure that makes them amenable to large-scale geometric programming (Ref. 1). As in LP, one can precisely define what is meant by a large sparse GP. Sparsity in the general NLP case is not easily defined.

In the summary, the conclusions are as follows.[14]

(i) Specialized GP codes do appear to offer improvement in computation times and to be more robust than general NLP codes, on GP problems.

(ii) Algorithms for the linearly constrained dual GP do not make use of the latest available technology that has been developed for linearly-constrained NLP.

(iii) Primal-based GP codes seem to dominate the field. The author feels that this is a reflection of (ii) above and *not* because of some inherent difficulty in the dual GP.

(iv) The Avriel–Williams (Ref. 17) and Duffin–Peterson (Ref. 34) algorithms for solving signomial programs are often extremely inefficient (refer to Problems 4A, 4B, and 4C) and experience shows that they often exhibit a linear rate of convergence.

(v) A methodology for comparison of mathematical programming software is sorely needed, in order that hypotheses about algorithm and code behavior may be tested in a scientific manner. Hopefully, Ref. 27 is a step in this direction.

7.1. Suggestions for Future Research

(i) There is a pressing need for an efficient and robust dual-based GP code (if only as an intellectual challenge). The author feels that the approach that should be taken is to specialize some of the numerically stable techniques for linearly constrained NLP as described in Ref. 2. A framework for doing this already exists (see Ref. 1). In particular, any strategy that is adopted should be amenable to extensions to large-scale applications. That is, the algorithm should be able to exploit sparsity and/or problem structure.

(ii) Some theoretical developments are that definitely needed, before dual-based algorithms can be competitive, are efficient algorithms for handling *simple bounding constraints on primal variables.* It is not unreasonable to postulate that the resulting special structure in the dual problem, when simple primal bounding constraints are present, could be exploited

[14] Strictly speaking, these conclusions are only valid for the test problems solved in this study. Since, however, Rijckaert and Martens (Ref. 4) reach similar conclusions on a *different* set of problems, there is reason to believe that these concludions will be true for a wide variety of GP problems.

computationally. This is an important consideration, because simple bounding constraints are invariably present in models of real systems.

(iii) Many nonlinear programming applications (Ref. 35) are signomial problems that may be written in the form:

$$\underset{y,z}{\text{minimize}} \quad \sum_{j\in J_0} c_j \exp(y_j), \tag{34}$$

$$\text{subject to} \quad \sum_{j\in J_k} c_j \exp(y_j) = 1, \qquad k = 1, 2, \ldots, q, \tag{35}$$

$$\sum_{j\in J_k} c_j \exp(y_j) \le 1, \qquad k = q+1, \ldots, p, \tag{36}$$

$$y_j - \sum_{i=1}^{m} a_{ij} z_j = 0, \qquad j = 1, 2, \ldots, n, \tag{37}$$

$$l_j \le z_j \le u_j, \qquad j = 1, 2, \ldots, n, \tag{38}$$

where the sets J_k are defined as before and the coefficients c_j, the "exponents" a_{ij}, and the variable bounds l_j and u_j are arbitrary real numbers (with $l_j < u_j$, of course).

The common approach to solving such problems (Ref. 35) has been to somehow convert Eqs. (35) to inequalities and then to apply the Avriel–Williams procedure, which converts the solution of the above signomial problem to the solution of a sequence of GP's. This strategy is fraught with difficulties and requires an experienced mathematical programmer for its implementation. It is this author's feeling that research into the development of software for signomial programs should concentrate on a direct solution of the above program. It has special features that would make either a generalized reduced gradient approach or an augmented Lagrangian approach attractive possibilities when designing a code. Naturally, if the code is to be competitive, the linear constraints (37) and (38) should be handled in some implicit fashion.

8. Appendix: Summary of GP Software Reported in the Literature

This appendix summarizes available information on GP codes that have appeared in the literature. We only include publications in which there is some evidence that the proposed algorithm has been coded and tested on a number of problems. There have been many attempts at coding GP algorithms, and it is hoped that none of these have been omitted here. Codes that are not mentioned in this section have not been omitted purposefully and were simply not known to this author at the time of writing.

Information on the codes is presented in the chronological order in which they appeared in the literature. An asterisk next to the code's name indicates that it appears in the computational study in Section 5.

Primal-Based Codes

Code 1

Name: DAP.

Author: G. V. Reklaitis (Ref. 36).

References: G. V. Reklaitis and D. J. Wilde (Refs. 37, 38, 16).

Algorithm: Solves SP directly using the differentiable algorithm of Wilde and Beightler (Ref. 39). Signomial programs solved via sequential GP approximation scheme of Duffin (Ref. 40).

Comments: No Phase 1 method reported. Reklaitis (Ref. 41) has compared DAP to the primal code GGP and in general has found GGP to be far more efficient.

Code 2

Name: *GGP**

Author: R. Dembo.

References: Dembo (Ref. 18, Ref. 5); Avriel, Dembo, and Passy (Ref. 19).

Algorithm: Solves PGP directly using a cutting-plane algorithm based on condensation. Signomial problems are solved using an accelerated AW algorithm.

Comments: Does not require a feasible starting point. Phase 1 algorithm operates on a modified problem to compute an initial feasible point if necessary. Experience with GGP has shown it to be both reliable and efficient. The code has been widely distributed and has been used to solve a large number of GP apllications. Feedback indicates that the method is robust and efficient for small- to medium-sized problems. A very similar algorithm was coded and tested as early as 1968 at Mobil Research Laboratories (Williams, Ref. 6).

Code 3

Authors: W. Gochet and Y. Smeers (Ref. 42).

Algorithm: Solves SPGP directly using a cutting-plane algorithm. Cuts are shown to be "deeper" than Kelley cuts.

Comments: No extension to signomial programs is reported. Computational experience is reported on two of Beck and Ecker's (Ref. 12) problems.

Code 4

Authors: G. S. Dawkins, B. C. McInnis, and S. K. Moonat (Ref. 43).

Algorithm: Tangential approximation method of Hartley and Hocking (Ref. 44). Operates on PGP in the variables $z = \log x$.

Comments: No extension to signomial programming reported. No details of the implementation are given and only the solution of a single problem is presented.

Code 5

Authors: J. G. Ecker and M. J. Zoracki (Ref. 45).

Algorithm: PGP is converted to a GP with at most two monomial terms in each constraint, according to the procedure outlined by Duffin and Peterson (Ref. 20). This posybinomial problem is solved using a hybrid of the tangential approximation and cutting-plane methods.

Comments: No extensions to signomial programming are given, and only a limited amount of computational experimentation is reported.

Code 6

Name: GPKTC*.

Authors: M. J. Rijckaert and X. M. Martens (Ref. 22).

Algorithm: The Kuhn–Tucker conditions for optimality of PGP are solved iteratively using a condensation procedure. This method is essentially equivalent to a Newton–Raphson algorithm for direct solution of the Kuhn–Tucker conditions expressed in terms of the variables $z = \log x$.

Comments: The code GPKTC has been extensively tested in Rijckaert and Martens (Ref. 4) and appears in the comparative study in Section 5. Experimentation with the code has shown it to be very robust and efficient especially for small GP problems. The code does not have an efficient mechanism for handling simple bounding constraints and constraints that are slack at optimality. A Phase 1 procedure is included in GPKTC and, judging from the results in Section 5, it appears to work well.

Code 7

Name: GEOLP*.

Authors: J. J. Dinkel, W. H. Elliott, and G. A. Kochenberger (Ref. 46).

References: Dembo (Ref. 18); Avriel, Dembo, and Passy (Ref. 19).

Algorithm: Essentially the same as GGP.

Comments: The authors claim to have successfully solved fairly large problems using GEOLP, and they feel (Ref. 46) that GEOLP works better than any other GP software that they have developed. It is interesting to note (see Section 6) that, despite the fact that GGP and GEOLP are based on the same algorithm, GGP seems to do consistently better than GEOLP on the test problems in Ref. 23. One possible explanation could be that GGP contains a number of algorithm refinements not contained in Refs. 18 and 19.

Code 8

Authors: M. Rammamurthy and G. H. Gallagher (Ref. 7).

References: Dembo (Ref. 18); Avriel, Dembo, and Passy (Ref. 19).

Algorithm: Essentially the same as GGP.

Comments: The authors claim to have solved a large number of civil engineering design problems using their code. Experience similar to that quoted in Code 2 above.

Dual-Based Codes

Code 1

Author: C. J. Frank.

References: Frank (Refs. 47, 48).

Algorithm: Solves the dual program DGP by applying the direct search method of Hooke and Jeeves (Ref. 49). Probably, the first published GP code. Experimentation has indicated that, in many cases, convergence of the method is extremely slow.

Comments: Code does not solve signomial problems.

Code 2

Name: GOMTRY

Authors: G. E. Blau and D. J. Wilde (Refs. 50, 51).

Algorithm: Solves the Kuhn–Tucker conditions for the dual program SDGP. Solves signomial programs in the above manner by attacking the necessary conditions for optimality of the pseudo-dual problem (Ref. 49).

Comments: GOMTRY's convergence is relatively good for small problems but experimentation (Ref. 4) shows that the code often fails to converge especially for medium-sized problems.

Code 3

Author: G. W. Westley (Ref. 52).

Algorithm: Based on the Murtagh and Sargent (Ref. 53) projection method for linearly constrained nonlinear programs. Nondifferentiability of the dual objective function is handled by placing arbitrary bounds on the dual variables.

Comments: Solves DGP; no extensions to signomial programs are reported. Bradley (Ref. 21) has tested the code extensively and reports that it often failed to solve even simple problems taken from the literature.

Code 4

Name: SIGNOPT*.

Authors: A. B. Templeman, A. J. Wilson, and S. K. Winterbottom (Ref. 54)

Algorithm: Signomial problems are solved using the Avriel and Williams (Ref. 17) procedure. The posynomial subproblems are solved by explicitly forming the reduced RDGP and solving it using a modified Fletcher–Reeves (Ref. 55) algorithm. Nondifferentiability of dual objective is handled by placing arbitrary lower bounds on the dual variables.

Comments: The code has been extensively tested by Templeman (Ref. 14) and Bradley (Ref. 21), both of whom claim a fair degree of success in solving small- to medium-size problems. However, these authors indicate that convergence can often be very slow. These findings are born out by the results in Section 6 and in Rijckaert and Martens (Ref. 4). No Phase 1 capability is included in SIGNOPT to initiate the Avriel and Williams procedure.

Code 5

Name: GPROG*.

Author: T. Jefferson (Refs. 56, 57).

Algorithm: Explicitly forms the reduced dual RDGP and solves it using a modified Newton algorithm. Nondifferentiabilities are avoided by adding slack variables to the primal in the manner of Duffin and Peterson (Ref. 58). Extension to signomials is carried out using the harmonic mean procedure of Duffin and Peterson (Ref. 28).

Comments: Unless the invariance of the dual coefficient matrix is exploited, the harmonic approach can be shown to be less desirable than the condensation algorithm of Avriel and Williams (Ref. 17). Experience with the code shows that it often fails to converge. This code and QUADGP, however, are the only ones mentioned in this paper with the capability of performing a detailed sensitivity analysis. GPROG does not contain a Phase 1 routine for signomial problems.

Code 6

Name: CSGP.

Authors: P. A. Beck and J. G. Ecker (Ref. 12).

Algorithm: The concave simplex method is applied to DGP with a modification that allows for blocks of variables to go to zero simultaneously. It is this modification that overcomes the nondifferentiability problem.

Comments: This code stands out as being the only dual-based code that attempts in a theoretically sound manner to overcome the nondifferentiability problem and to include an option to solve subsidiary problems, if they are needed, when converting to an optimal solution of the primal PGP. No provision is made in the code for signomial problems. The code has been tested extensively by Rijckaert and Martens (Ref. 4) and Beck and Ecker (Ref. 12). Experience shows that CSGP is sometimes slow relative to other codes but that it is fairly reliable.

Code 7

Authors: G. A. Kochenberger, R. E. D. Woolsey, and B. A. McCarl (Ref. 59).

Algorithm: Solves SDGP using separable programming.

Comments: No extensions to signomial programming are mentioned. Only computational experience reported is on one small problem and for this problem the method does poorly.

Code 8

Name: NEWTGP.

Author: J. Bradley (Ref. 60).

Algorithm: Explicitly reduces dual program DGP to the program RDGP. Solves the nonlinear equations resulting from the Kuhn–Tucker conditions for optimality of RDGP using a Newton–Raphson procedure. Nondifferentiability is avoided by setting a *pseudo boundary* which prevents the algorithm from hitting a $\delta \geq 0$ constraint. This is equivalent to artificially adding a $\delta \geq \epsilon$ constraint.

Comments: The code has been tested extensively by Rijckaert and Martens (Ref. 4), and the indications are that it often fails to converge; in cases where convergence is attained, the code does not compete well against the best primal methods.

Code 9

Name: GEOGRAD, GEOEPS*.

Authors: J. J. Dinkel, G. A. Kochenberger, and B. A. McCarl (Ref. 61).

Algorithm: The algorithm is essentially the same as the one used by Bradley in NEWTGP. Extension to signomials is carried out using the GEOEPS routine which executes the Avriel and Williams (Ref. 17) algorithm.

Comments: No Phase 1 method for signomials. The implementation seems to perform reasonably well when it converges. However, it is generally not competitive with the best primal methods.

Code 10

Name: QUADGP.

Author: J. Bradley (Ref. 21).

Algorithm: Solves posynomial programs by explicitly forming the reduced dual program RDGP which is then solved by successive quadratic approximations. Nondifferentiability of the dual objective function is handled by artificially bounding dual variables from below. This bound, in contrast to those mentioned previously, is dynamic and decreases rapidly from iteration to iteration. Dual variables are zeroed when an estimate of the primal constraint multiplier becomes very small. The code has options for both the Avriel and Williams (Ref. 17) and the Duffin and Peterson (Ref. 28) extensions to signomial programming, but Bradley (Ref. 21) indicates that the Avriel–Williams procedure is to be preferred.

Comments: A feasible point is required to initiate QUADGP for signomial problems. Bradley (Ref. 21) has tested QUADGP extensively. However, the only relative measure of effectiveness with the codes in Section 5 can be obtained from Problems 8A, 8B, and 8C, for which computation times are given in his thesis. An important feature of QUADGP is that it has the capability of performing sensitivity analysis.

Code 11

Names: LAM, SP, LM, NRF, NRT, NRVB, DCA.

Authors: M. J. Rijckaert and X. M. Martens (Ref. 4).

Algorithm: LAM is a linear approximation method for solving the dual and SP is based on a separable programming algorithm. LM, NRF, NRT, NRVB and DCA are all essentially based on Newton-type algorithms for solving the Kuhn–Tucker conditions of SDGP.

Comments: These codes are described and tested in (Ref. 4) and do not seem to be competitive with the best available software. In particular, they do not appear to be robust and often fail to converge on medium-sized problems.

Code 12

Author: J. R. McNamara (Ref. 62).

Algorithm: Constructs an augmented primal problem with zero degrees of difficulty. The augmented problem depends on a number of parameters and for certain realizations of these parameters the solution to the augmented dual is determined uniquely.

Comments: Solves posynomial problems only. Computational results are given for 2 trivial examples (Ref. 62), and mention is made of larger examples. The author indicates that the proposed method does not necessarily converge (Ref. 62, p. 23).

References

1. DEMBO, R. S., *Second-Order Algorithms for the Geometric Programming Dual, Part 1: Analysis,* Mathematical Programming (to appear).
2. GILL, P. E., and MURRAY, W., *Numerical Methods in Constrained Optimization,* Academic Press, New York, New York, 1974.
3. WOLFE, P., *Convergence Theory in Nonlinear Programming,* Integer and Nonlinear Programming, Edited by J. Abadie, North-Holland Publishing Company, Amsterdam, Holland, 1970.
4. RIJCKAERT, M. J., and MARTENS, X. M., *A Comparison of Generalized Geometric Programming Algorithms,* Katholieke Universiteit te Leuven, Report No. CE-RM-7503, 1975.
5. DEMBO, R. S., *GGP—A Computer Program for Solving Generalized Geometric Programting Problems,* Technion, Israel Institute of Technology, Department of Chemical Engineering, Users Manual, Report No. 72/59, 1972.
6. WILLIAMS, A. C., Private Communication, 1972.
7. RAMMAMURTHY, S., and GALLAGHER, R. H., *Generalized Geometric Programming in Light Gage Steel Design,* Paper Presented at the ORSA/TIMS Meeting, Miami, Florida, 1976.
8. DINKEL, J. J., and KOCHENBERGER, G. A., Private Communication, 1975.
9. DUFFIN, R. J., PETERSON, E. L., and ZENER, C., *Geometric Programming—Theory and Application,* John Wiley and Sons, New York, New York, 1967.
10. IBM Corporation, *Mathematical Programming System—Extended (MPSX) and Generalized Upper Bounding (GUB) Program Description,* Program No. 5734-XM4, 1972.
11. DEMBO, R. S., *Dual to Primal Conversion in Geometric Programming,* Journal of Optimization Theory and Applications, Vol. 26, No. 1, 1978.
12. BECK, P. A., and ECKER, J. G., *A Modified Concave Simplex Algorithm for Geometric Programming,* Journal of Optimization Theory and Applications, Vol. 15, pp. 189–202, 1975.
13. DEMBO, R. S., *Sensitivity Analysis in Geometric Programming,* Yale University, School of Organization and Management, Working Paper No. SOM-35, 1978.
14. TEMPLEMAN, A. B., Private Communication, 1975.
15. PASSY, U., and WILDE, D. J., *Generalized Polynomial Optimization,* SIAM Journal on Applied Mathematics, Vol. 15, pp. 1344–1356, 1967.
16. BEIGHTLER, C. S., and PHILLIPS, D. T., *Applied Geometric Programming,* John Wiley and Sons, New York, New York, 1976.
17. AVRIEL, M., and WILLIAMS, A. C., *Complementary Geometric Programming,* SIAM Journal on Applied Mathematics, Vol. 19, pp. 125–141, 1970.
18. DEMBO, R. S., *The Solution of Complementary Geometric Programming Problems,* Technion, Israel Institute of Technology, MS Thesis, 1972.
19. AVRIEL, M., DEMBO, R. S., and PASSY, U., *Solution of Generalized Geometric Programming Problems,* International Journal of Numerical Methods in Engineering, Vol. 9, pp. 141–169, 1975.

20. DUFFIN, R. J., and PETERSON, E. L., *Geometric Programming with Signomials*, Journal of Optimization Theory and Applications, Vol. 11, pp. 3–35, 1973.
21. BRADLEY, J., *The Development of Polynomial Programming Algorithms with Applications*, Dublin University, Department of Computer Science, PhD Thesis, 1975.
22. RIJCKAERTS, M. J., and MARTENS, X. M., *A Condensation Method for Generalized Geometric Programming*, Katholieke Universiteit te Leuven, Report No. CE-RM-7503, 1975.
23. DEMBO, R. S., *A Set of Geometric Programming Test Problems and Their Solutions*, Mathematical Programming, Vol. 10, pp. 192–213, 1976.
24. DINKEL, J. J., KOCHENBERGER, G. A., and MCCARL, B., *A Computational Study of Methods for Solving Polynomial Geometric Programs*, Journal of Optimization Theory and Applications, Vol. 19, pp. 233–259, 1976.
25. COLVILLE, A. R., *A Comparative Study of Nonlinear Programming Codes*, IBM, New York Scientific Center, Report No. 320–2949, 1968.
26. RATNER, M., LASDON, L. S., and JAIN, A., *Solving Geometric Programs Using GRG—Results and Comparisons*, Standford University, Systems Optimization Laboratory, Technical Report No. SOL-76-1, 1976.
27. DEMBO, R. S., and MULVEY, J. M., *On the Analysis and Comparison of Mathematical Programming Algorithms and Software*, Proceedings of the Bicentennial Conference on Mathematical Programming, Gaithersburg, Maryland, 1976.
28. HIMMELBLAU, D. M., *Applied Nonlinear Programming*, McGraw-Hill Book Company, New York, New York, 1972.
29. RIJCKAERT, M. J., Private Communication, 1975.
30. LASDON, L. S., WARREN, A. D., RATNER, M. W., and JAIN, A., *GRG System Documentation*, Cleveland State University, Technical Memorandum No. CIS-75-01, 1975.
31. ABADIE, J., and GUIGOU, J., *Numerical Experiments with the GRG Method*, Integer and Nonlinear Programming, Edited by J. Abadie, North Holland Publishing Company, Amsterdam, Holland, 1970.
32. KREUSER, J. L., and ROSEN, J. B., *GPM/GPMNLC Extended Gradient Projection Method Nonlinear Programming Subroutines*, University of Wisconsin, Academic Computer Center, 1971.
33. HIMMELBLAU, D. M., Private Communication, 1975.
34. DUFFIN, R. J., and PETERSON, E. L., *Reserved Geometric Programs Treated by Harmonic Means*, Carnegie–Mellon University, Research Report No. 71–79, 1971.
35. DEMBO, R. S., *Some Real-World Applications of Geometric Programming*, Applied Geometric Programming, Edited by C. S. Beightler and D. T. Phillips, Prentice-Hall, Englewood Cliffs, New Jersey, 1976.
36. REKLAITIS, G. V., *Singularity in Differentiable Optimization Theory: Differential Algorithm for Posynomial Programs*, Stanford University, PhD Thesis, 1969.
37. REKLAITIS, G. V., and WILDE, D. J., *A Differentiable Algorithm for Posynomial Programs*, DECHEMA Monographien, Vol. 67, pp. 503–542, 1971.

38. REKLAITIS, G. V., and WILDE, D. J., *Geometric Programming via a Primal Auxiliary Problem*, AIIE Transactions, Vol. 6, 1974.
39. WILDE, D. J., and BEIGHTLER, D. D., *Foundations of Optimization*, Prentice-Hall, Englewood Cliffs, New Jersey, 1967.
40. DUFFIN, R. J., *Linearizing Geometric Programs*, SIAM Review, Vol. 12, pp. 211–227, 1970.
41. REKLAITIS, G. V., Private Communication, 1976.
42. GOCHET, W., and SMEERS, Y., *On the Use of Linear Programs to Solve Prototype Geometric Programs*, Katholieke Universiteit te Leuven, Center for Operations Research and Econometrics, Discussion Paper No. 7229, 1972.
43. DAWKINS, G. S., MCINNIS, B. C., and MOONAT, S. K., *Solution to Geometric Programming Problems by Transformation to Convex Programming Problems*, International Journal of Solid Structures, Vol. 10, pp. 135–136, 1974.
44. HARTLEY, H. O., and HOCKING, R. R., *Convex Programming by Tangential Approximation*, Management Science, Vol. 9, pp. 600–612, 1963.
45. ECKER, J. G., and ZORACKI, M. J., *An Easy Primal Method for Geometric Programming*, Management Science, Vol. 23, pp. 71–77, 1976.
46. DINKEL, J. J., ELLIOTT, W. H., and KOCHENBERGER, G. A., *A Linear Programming Approach to Geometric Programs*, Naval Research Logistics Quarterly (to appear).
47. FRANK, C. J., *An Algorithm for Geometric Programming*, Recent Advances in Optimization Techniques, Edited by D. D. Lavi and D. D. Vogel, John Wiley and Sons, New York, 1966.
48. FRANK, C. J., *Development of a Computer Program for Geometric Programming*, Westinghouse Report No. 64-1, HO-124-R2, 1964.
49. HOOKE, R., and JEEVES, T. A., *Direct Search Solution of Numerical and Statistical Problems*, Journal of the Association for Computing Machinery, Vol. 8, pp. 212–219, 1961.
50. BLAU, G. E., and WILDE, D. J., *A Lagrangean Algorithm for Equality Constrained Generalized Polynomial Optimization*, AIChE Journal, Vol. 17, pp. 235–240, 1971.
51. KUESTER, J. L., and MIZE, J. H., *Optimization Techniques with FORTRAN Programs*, McGraw-Hill Book Company, New York, New York, 1973.
52. WESTLEY, G. W., *A Geometric Programming Algorithm*, Oak Ridge National Laboratory, Technical Report No. ORNL-4650, 1971.
53. MURTAGH, B. A., and SARGENT, R. W. H., *A Constrained Minimization Method with Quadratic Convergence*, Optimization, Edited by R. Fletcher, Academic Press, London, 1969.
54. TEMPLEMAN, A. B., WILSON, A. J., and WINTERBOTTOM, S. K., *SIGNOPT—A Computer Code for Solving Signomial Geometric Programming Problems*, University of Liverpool, Department of Civil Engineering, Research Report, 1972.
55. FLETCHER, R., and REEVES, C. M., *Function Minimization by Conjugate Gradients*, Computer Journal, Vol. 7, pp. 149–154, 1964.

56. JEFFERSON, T., *Geometric Programming, with an Application to Transportation Planning*, Northwestern University, PhD Thesis, 1972.
57. JEFFERSON, T., *Manual for the Geometric Programming Code GPROG (CDC) VERSION 2*, University of New South Wales, Australia, Mechanical and Industrial Engineering Department, Report No. 1974/OR/2, 1974.
58. DUFFIN, R. J., and PETERSON, E. L., *Geometric Programs Treated with Slack Variables*, Applied Analysis, Vol. 2, pp. 255–267, 1972.
59. KOCHENBERGER:, G. A., WOOLSEY, R. E. D., and MCCARL, B. A., *On the Solution of Geometric Programs via Separable Programming*, Operations Research Quarterly, Vol. 24, pp. 285–296, 1973.
60. BRADLEY, J., *An Algorithm for the Numerical Solution of Prototype Geometric Programs*, Institute of Industrial Research and Standards, Dublin, Ireland, 1973.
61. DINKEL, J. J., KOCHENBERGER, G. A. and MCCARL, B. A., *An Approach to the Numerical Solution of Geometric Programs*, Mathematical Programming, Vol., pp. 181–190, 1974.
62. MCNAMARA, J. R., *A Solution Procedure for Geometric Programming*, Operations Research, Vol. 24, pp. 15–25, 1976.
63. DUFFIN, R. J., and PETERSON, E. L., *The Proximity of (Algebraic) Geometric Programming to Linear Programming*, Mathematical Programming, Vol. 3, pp. 250–253, 1972.
64. SHAPLEY, M., and CUTLER, L., *Rand's Chemical Composition Program—A Manual*, The Rand Corporation, Report No. 495–PR, 1970.
65. CLASEN, R. J., *The Numerical Solution of the Chemical Equilibrium Problem*, The Rand Corporation, Report No. 4345-PR, 1965.