JOURNAL OF OPTIMIZATION THEORY AND APPLICATIONS: Vol. 55, No. 1, OCTOBER 1987

Global Optimization Algorithms for a CAD Workstation[1]

W. L. PRICE[2]

Communicated by L. C. W. Dixon

Abstract. This paper describes two new versions of the controlled random search procedure for global optimization (CRS). Designed primarily to suit the user of a CAD workstation, these algorithms can also be used effectively in other contexts. The first, known as CRS3, speeds the final convergence of the optimization by combining a local optimization algorithm with the global search procedure. The second, called CCRS, is a concurrent version of CRS3. This algorithm is intended to drive an optimizing accelerator, based on a concurrent processing architecture, which can be attached to a workstation to achieve a significant increase in speed. The results are given of comparative trials which involve both unconstrained and constrained optimization.

Key Words. Numerical optimization, global search, nonlinear programming, parallel processing, concurrent algorithms, computer-aided design.

1. Introduction

Two versions, CRS1 and CRS2, of the controlled random search procedure for global optimization (CRS) have been described in previous papers (Price, Refs. 1 and 2). The CRS procedure is a simple, general purpose algorithm which is direct (it does not involve gradients) and is applicable to constrained as well as to unconstrained optimization. Because the algorithm emphasizes global search, rather than speed of convergence,

[1] This work was funded by the Science and Engineering Research Council.

[2] Senior Research Fellow, Electronic Systems Engineering, School of Information Systems, University of East Anglia, Norwich, England.

0022-3239/87/1000-0133$05.00/0

it is somewhat slower than algorithms designed for local optimization. However, the CRS algorithm is well suited to a parallel processing environment, giving the potential for a significant improvement in speed. Research at the Hatfield Polytechnic into adapting CRS for the ICL-DAP has produced promising results [Ducksbury (Ref. 3) and Dixon, Patel, and Ducksbury (Ref. 4)].

Rapid developments in computer technology have led to the increasing use of desk-top microcomputers as workstations for computer-aided engineering design. One of the CAD tools which the design engineer requires is an optimizing engine. The present paper describes adaptations of the CRS algorithm tailored specifically to the needs of the user of such a workstation.

Within the CAD environment, the principal facilities offered by a workstation are its immediacy of action and interaction together with high-quality graphics. In order to take full advantage of these facilities, the user needs to enter his problem, and to observe the results, as quickly and simply as possible. He might wish to interact with the procedure, e.g., choosing to stop it when he judges the optimization to have proceeded sufficiently far. He might then wish to restart the procedure with a change of parameters or a fresh set of data, using the workstation as a "what if?" design tool.

In this context, the CRS algorithm has certain advantages over many other global optimization procedures. Firstly, it requires minimum preparation of the problem: because the algorithm does not involve derivatives, only the function definition needs to be supplied. Secondly, the data in store at each iteration (and available to the workstation via a dynamic graphics display) provides the user with more information than merely the current best point. This data shows the current modality (the number of potential global minima being searched). The data also shows the sensitivity of the global minimum, i.e., the required tolerances on the engineering parameters which the variables represent. This kind of information aids the designer in his interactive decision-making. It is true that CRS requires more storage capacity than do some other global optimizers [for a function of n dimensions, CRS2 requires $10(n+1)^2$ floating point numbers], and this is the price paid for the additional information which CRS offers. But, as the cost and space requirements of storage media continue to fall, this is not likely to be a serious problem. Thirdly, CRS can be adapted readily for a parallel processing environment. Current research is concerned with the design of a hardware accelerator, based on concurrent processors, such as the INMOS transputer, which can be attached to a workstation. This accelerator, when driven by a concurrent version of CRS to be known as CCRS (concurrent CRS), could act as a fast optimizing engine. The intention

is to interface the accelerator with the workstation in such a way as to conceal it from the user, so that it becomes an integral part of the workstation.

For the interactive user of a workstation, a disadvantage of the CRS algorithm is its slowness in the final stage of convergence. Having decided that the algorithm has attained the region of a global minimum, the user might wish to switch to a faster, local optimization algorithm for the final refinement of the result. A new version of CRS2, to be known as CRS3, takes account of this need by building into the global optimizer a local optimization procedure.

Section 2 summarizes the principal features of the CRS2 version of CRS (CRS1, which requires more storage than CRS2, will not be discussed further). Section 3 describes the new version, CRS3, which extends CRS2 by the addition of a local optimization procedure. The comparative performances of CRS2 and CRS3 are discussed in Section 4. In Section 5, the concurrent version CCRS of the sequential algorithm CRS3 is described. The performances of CRS3 and CCRS are compared in Section 6.

2. CRS2 Algorithm

Given a function of n variables, an initial search domain V is defined by specifying limits to each variable. A predetermined number N of trial points are chosen at random over V, consistent with the additional constraints (if any). The function is evaluated at each trial and the position and function value corresponding to each point are stored in an array A. At each iteration, a new trial point P is selected randomly from a certain set of possible trial points. Provided that the position of P is within V and satisfies any additional constraints, the function is evaluated at P. The function value f_P is compared with f_M, M being the point which has the greatest function value of the N points presently stored in A. If $f_P < f_M$, then M is replaced in A by P. If either P fails to satisfy the constraints or $f_P > f_M$, then the trial is discarded and a fresh point is chosen from the potential trial set. As the algorithm proceeds, the current set of N stored points tend to cluster around minima which are lower than the current value of f_M.

The CRS2 procedure achieves a reasonable compromise between the conflicting requirements of search and convergence by defining the set of possible trial points in terms of the configuration of the N points currently stored. At each iteration, $n+1$ distinct points $R_1, \ldots, R_{n+1}$ are chosen from the N, $N \gg n$, in store. The point R_1 is always chosen as that point L which has the least function value f_L of the stored points. The other n points are chosen at random from the remaining $N-1$ points. These $n+1$ points

constitute a simplex in n-space. The point R_{n+1} is taken arbitrarily as the pole (designated vertex) of the simplex, and the next trial point P is defined as the image point of the pole with respect to the centroid G of the other n points. Thus,

$$P = 2G - R_{n+1},$$

where the symbols represent the position vectors, in n-space, of the corresponding points.

The number of potential trial points is $n \times {}^{N-1}C_n$. This number increases rapidly with n, provided that $N \gg n$, and it is sufficient that N increase linearly with n. For a given value of n, the greater the value of N the more thorough the search and the higher the probability of discovering a global minimum. On the other hand, the larger the value of N the greater is the demand on computer storage and the slower the convergence of the algorithm. The appropriate choice of N is a matter of experience, and the empirical rule adopted by the author is to take $N = 10(n+1)$. Thus, the array A, which is the database for CRS2, requires a storage capacity of $10(n+1)^2$ words.

This algorithm is summarized below. The user is free to define his own stop criterion. A criterion based on a comparison of the worst and best function values (f_M and f_L) is used in the trials described in Section 4. For further details of CRS2, and a discussion of performance, the reader is referred to the earlier paper (Price, Ref. 2).

CRS2 Algorithm

Step 1. Choose N points at random over V. Evaluate the function at each point. Store the position and function value in A.

Step 2. Find, in A, the worst point M with function value f_M and the best point L with function value f_L.

Step 3. Choose randomly n distinct points $R_2, \ldots, R_{n+1}$ excluding L. Let $R_1 = L$. Determine the centroid G of points $R_1, \ldots, R_n$. Compute the next trial point $P = 2G - R_{n+1}$.

Step 4. If P is within V and satisfies other constraints, then evaluate f_P and go to Step 5; else, return to Step 3.

Step 5. If $f_P < f_M$, then replace M by P in A and go to Step 6; else, return to Step 3.

Step 6. If the stop criterion is satisfied, then stop; else, return to Step 2.

3. CRS3 Algorithm

The CRS3 algorithm comprises the CRS2 algorithm together with a local optimization procedure called LOC. Because CRS2 is a nongradient algorithm based on the geometry of a simplex, it is appropriate to base LOC on the Nelder-Mead simplex algorithm (Ref. 5). The data required by this algorithm is explicitly available within the CRS2 database A. The $n+1$ best points of A (the bottom one-tenth of the array when arranged in descending order of function value) constitute a simplex in n-space. Then modified form of the Nelder-Mead algorithm adopted for LOC operates on this simplex as shown below.

CRS3 Algorithm

Step 1. Let W be the worst point of the simplex. Let G be the centroid of the other n points. Let S be the second worst point of the simplex. Compute three potential trial points,

$$P = 2G - W,$$

$$Q = (G + W)/2,$$

$$R = 4G - 3W.$$

Step 2. If P fails to satisfy the contraints, then go to Step 4; else, evaluate the function at P. If $f_p < f_s$, then go to Step 3; else, go to Step 4.

Step 3. If R fails to satisfy the contraints, then accept P as the replacement point and go to Step 5; else, evaluate the function at R. If $f_R < f_S$, then accept R as the replacement point and go to Step 5; else, accept P as the replacement point and go to Step 5.

Step 4. If Q fails to satisfy the constraints, then stop; else, evaluate the function at Q. If $f_Q < f_S$, then accept Q as the replacement point and go to Step 5; else, stop.

Step 5. Update the simplex by removing W and including the replacement point. Return to Step 1.

The composite CRS3 algorithm is described below.

Composite CRS3 Algorithm

Step 1. Run CRS2 until either CRS attains the stop criterion, in which case stop, or CRS2 generates a new point which falls within the bottom one-tenth of the ordered array A, in which case go to Step 2.

Step 2. Run LOC until LOC terminates; then, return to Step 1.

Two features of CRS3 should be noted.

Firstly, LOC operates only on the bottom one-tenth of A, and thus has a minimal effect on the global search performance of the CRS2 phase. Because CRS2 involves the best point of A (which may be improved by LOC), such effect as exists tends to speed the convergence of the algorithm and thus to reduce, to a small degree, the global search capability. If desired, it is easy to counter this effect by not requiring that CRS2 should invariably include the best point.

Secondly, LOC can operate at any stage of the CRS3 procedure. Hence, local optimization might occur around minima which later prove to be local, rather than global. The advantages of this feature are that it is entirely automatic (not requiring user intervention) and that it can provide the user with useful information concerning the progress of the search. However, CRS3 can be modified easily so as to permit the interactive user to switch LOC in or out as he chooses. He may then defer its use until he is satisfied that the global search phase is nearing completion. So as to be objective, the comparative trials described in Sections 4 and 6 made use of the CRS3 algorithm as defined by Table 3.

4. Comparative Performance of CRS2 and CRS3 Algorithms

The concurrent version of CRS3, to be described in Section 5, is written in the concurrent programming language OCCAM. Hence, for consistency and to achieve a fair comparison of the various algorithms, CRS2 and CRS3 have also been written in OCCAM. These programs were run on a VAX-11/730 computer.

The stop criterion used for CRS2 and for the CRS2 component of CRS3 was defined in terms of the worst point of the array (with function value f_M) and the best point (with function value f_L). When the convergence is such that

$$f_M/f_L < 1.001$$

the algorithm stops. This criterion is, of course, arbitrary and can be modified readily by the user. Indeed, the interactive user might well choose to terminate the procedure long before the built-in stop criterion is activated.

Results are given for three problems. Two of these, Goldstein/Price and Shekel, are standard test problems for global optimization. The third, a problem involving contrained optimization, arises in the context of transformer design and will be called Transformer. These problems are defined in the Appendix (Section 8).

Table 1. Results for the Goldstein/Price function.

	CRS2		CRS3	
	Global minimum	Final convergence	Global minimum	Final convergence
Min	480	484	86	274
Av.	536	568	239	394
Max	589	640	451	528

For each problem, the same series of five different random sequences was used. The performances of CRS2 and CRS3 are compared in terms of the number of function evaluations required, both to achieve the global minimum and to achieve the stop criterion. The tables which follow show the minimum, average, and maximum numbers of evaluations over the series of five trials.

The results for Goldstein/Price (Table 1) and for Shekel (Table 2) show that CRS3 is able to find the global minimum more quickly than CRS2 (by virtue of LOC), and hence tends to be somewhat faster in achieving final convergence. For CRS3, there is a significant difference between the minimum and maximum numbers of function evaluations, over the set of five trials, needed to find the global minimum. For some runs (and depending on the particular random sequence used), LOC is fortuitously placed to home in quickly on the global minimum, whereas for others it expends effort on minima which later prove to be local, rather than global.

Table 2. Results for the Shekel function with $m = 5, 7, 10$.

	CRS2		CRS3		
	Global minimum	Final convergence	Global minimum	Final convergence	
Min	2460	2545	584	2175	$m = 5$
Av.	2760	2815	1335	2635	$m = 5$
Max	2988	3118	2217	3461	$m = 5$
Min	2265	2348	186	1887	$m = 7$
Av.	2408	2507	935	2148	$m = 7$
Max	2755	2788	2564	2583	$m = 7$
Min	2362	2463	193	1814	$m = 10$
Av.	2876	2957	659	2165	$m = 10$
Max	3445	3522	1411	2434	$m = 10$

Table 3. Results for the transformer function.

	CRS2		CRS3	
	Within 4% of optimum	Final convergence	Within 4% of optimum	Final convergence
Min	3598	12607	1913	10719
Av.	4088	13522	3234	11737
Max	4429	14562	4765	12847

Because it involves constrained optimization, Transformer poses certain difficulties. One way of preparing this problem so as to avoid the use of penalty functions has been described in a previous paper (Price, Ref. 2). In the present context, however, the intention is to minimize the amount of problem preparation required of the interactive user. Hence, the penalty function approach has been adopted. For the Transformer problem, it so happens that the global minimum lies on a constraint boundary. This makes it particularly difficult for LOC to operate effectively, even though the CRS2 phase reaches readily the neighborhood of the global minimum. In many engineering design applications (including transformer design), a high degree of precision in the final result is not required. The results for Transformer (Table 3) indicate the number of function evaluations needed to approach within 4% of the global minimum. The interactive user might decide to stop the optimization at this stage.

The conclusions to be drawn from the comparative results for this problem are similar to those of the two previous problems. Of the two sequential algorithms, CRS3 is preferable to CRS2.

5. CCRS Algorithm

A high proportion of the processing time involved in running a typical optimization procedure is spent in function evaluation. It is for this reason that comparisons between different procedures are normally based on the number of function evaluations required. For CRS, the amount of time involved in data transference, updating, and control increases linearly with the dimensionality n of the function to be optimized. In contrast, the time involved in generating and evaluating a new trial point increases at least polynomially with n. When optimizing complicated multi-dimensional functions, it is therefore advantageous to run a number of processes concurrently. One processor can control the input/output communications and updating

of the database, while the remainder concentrates on trial generation and function evaluation.

A concurrent processing structure appropriate to CCRS is shown in Fig. 1. Each rectangular block represents a discrete processor and intercommunication is via high-speed serial channels. The INMOS transputer is designed for just such a role. However, the transputer can support a maximum of four communication channels (each possibly bidirectional), and the architecture of Fig. 1 reflects this contraint. Information from the database is transmitted down pipeline A, from which it may be copied, in passing, by any of the CRS2 processors. Each cell of this pipeline also stores the most recent information concerning the best point L of the database, and the value f_M of the worst point. Information about new trials is sent from the CRS2 processors to the database via pipeline B.

Each CRS2 processor runs three parallel processes: a cell of pipeline A, a cell of pipeline B, and the process which generates, tests (constraints), and evaluates a trial point P. The processor contains a random number source which enables the generator to copy, in addition to L, a randomly chosen set of database points in transit down pipeline A.

This random number source is seeded differently for each of the CRS2 processors so that it is highly improbable that any two will generate the same trial point. Having evaluated f_P, the processor compares this value with f_M (copied from pipeline A) and then either rejects the trial or forwards the result down pipeline B. A further acceptance test is made by the database processor on entry because, as a result of pipeline delay, the current value of f_M in the database might be less than that value on which the generating processor's decision was based.

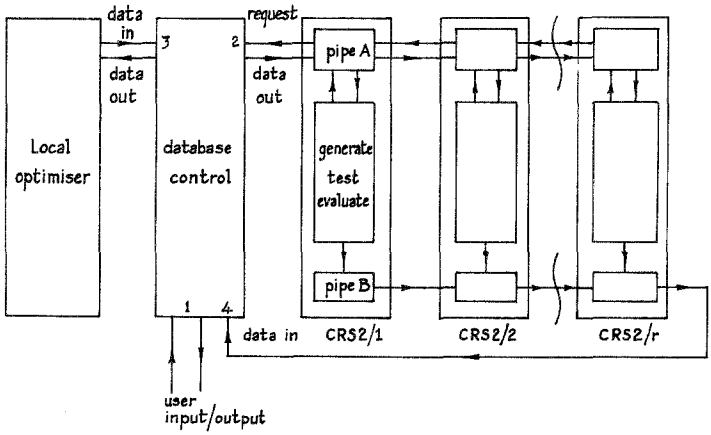

Fig. 1. A concurrent structure for the CCRS algorithm.

The processor devoted to LOC operates as described in Section 3. It waits until the database processor sends the relevant data; then, after each iteration, it returns a result to the database. The LOC process continues until the stop criterion is satisfied, after which the processor waits for a fresh set of data.

The database processor controls the input/output communications and manages the updating procedure. The priorities which this processor attaches to each of the four communication channels are shown in Fig. 1: here, 1 is the highest priority and 4 the lowest.

In principle, the upper limit to the number of CRS2 processors which can run concurrently is determined by the ability of the database controller to handle the data flows without bottlenecks occurring in the pipelines. However, this number increases with n (for the reasons discussed above), and in practice the maximum number is likely to be set by economic considerations. For the series of trials described in Section 6, ten CRS2 processors were simulated, making a total or twelve for the complete CCRS structure. The choice of ten was arbitrary: a hardware implementation would probably use a greater number of processors.

6. Comparative Performance of CRS3 and CCRS Algorithms

The CCRS algorithm is written in OCCAM, thus permitting the concurrent structure to be simulated on the VAX-11/730 computer.

Results are given in Tables 4, 5, 6 for the same series of test trials as was described in Section 4. The results for CRS3 are repeated for ease of comparison. The numbers of function evaluations quoted for CCRS are the sum totals for all ten CRS2 processors together with those evaluations performed by the LOC processor.

These results suggest that, in terms of the numbers of function evaluations, the performance of CCRS is statistically similar to that of CRS3.

Table 4. Results for the Goldstein/Price function.

	CRS3		CCRS	
	Global minimum	Final convergence	Global minimum	Final convergence
Min	86	274	73	172
Av.	239	394	325	433
Max	451	528	580	685

Table 5. Results for the Shekel function with $m = 5, 7, 10$.

	CRS3		CCRS		
	Global minimum	Final convergence	Global minimum	Final convergence	
Min	584	2175	718	2392	$m = 5$
Av.	1335	2635	1524	2796	$m = 5$
Max	2217	3461	2501	3120	$m = 5$
Min	186	1887	210	2045	$m = 7$
Av.	935	2148	896	2286	$m = 7$
Max	2564	2583	2722	2895	$m = 7$
Min	193	1814	253	2078	$m = 10$
Av.	659	2165	719	2426	$m = 10$
Max	1411	2434	1090	2813	$m = 10$

For Transformer, the agreement is remarkably close. It should be remembered that the dimensionalities of Goldstein/Price, Shekel, and Transformer are respectively 2, 4, and 6. The benefits of concurrency are likely to be more pronounced as the dimensionality increases. It is to be expected that CCRS would achieve about a tenfold improvement in speed if it were run on multi-processor hardware incorporating twelve processors (assuming, of course, that processing activity is dominated by trial generation and function evaluation).

7. Conclusions

For the interactive user of a CAD workstation, CRS3 and CCRS offer distinct advantages over CRS2. The CRS3 algorithm provides the user with

Table 6. Results for the transformer function.

	CRS3		CCRS	
	Within 4% of optimum	Final convergence	Within 4% of optimum	Final convergence
Min	1913	10719	1923	10033
Av.	3234	11737	3307	11620
Max	4765	12847	4965	15138

a local, as well as a global, search facility. The local optimization procedure can operate automatically, under control of CRS2, or can be switched in under user control. Because CRS3 is a sequential algorithm, it will run on a conventional microcomputer.

An optimizing accelerator, implemented in concurrent processing hardware, can be integrated with the workstation to achieve a significant improvement in speed. A concurrent version CCRS of the CRS3 algorithm has been simulated on the VAX-11/730 computer. The results of this simulation suggest that the speed-up factor should increase linearly with the number of discrete processors incorporated in the accelerator, provided that communication overheads are relatively insignificant.

Note Added in Proof. Since this paper was written, a small prototype accelerator, involving five transputers, has been built and tested. This accelerator achieves the expected fourfold reduction in running time.

8. Appendix: Test Functions

Goldstein/Price Function. It is required to minimize

$$f(x_1, x_2) = [1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)]$$
$$\times[30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)].$$

The search domain is

$$-2 \le x_1, \qquad x_2 \le 2.$$

There are four local minima. The global minimum occurs at $(0, -1)$.

Shekel Function. It is required to minimize

$$f(x) = -\sum_{i=1}^{m} \{1/[(x-a_i)^T(x-a_i)+c_i]\},$$

$$x = (x_1, .., x_n)^T, \qquad a_i = (a_{il}, \ldots, a_{in})^T, \qquad c_i > 0.$$

The search domain is

$$0 \le x_j \le 10, \qquad j = 1, \ldots, n.$$

Three cases are considered, using the data from Table 7 with $n = 4$ and $m = 5, 7, 10$.

Table 7. Data for the Shekel function.

i	a_i				c_i
1	4	4	4	4	0.1
2	1	1	1	1	0.2
3	8	8	8	8	0.2
4	6	6	6	6	0.4
5	3	7	3	7	0.4
6	2	9	2	9	0.6
7	5	5	3	3	0.3
8	8	1	8	1	0.7
9	6	2	6	2	0.5
10	7	3.6	7	3.6	0.5

Transformer Function. This six-variable problem arises in connection with transformer design [Ballard, Jelinek, and Schinzinger (Ref. 6)]. It is required to minimize

$$f = 0.0204x_1x_4(x_1+x_2+x_3)+0.0187x_2x_3(x_1+1.57x_2+x_4)$$
$$+0.0607x_1x_4x_5^5(x_1+x_2+x_3)+0.0437x_2x_3x_6^2(x_1+1.57x_2+x_4),$$

subject to the inequality constraints

$$x_i \geq 0, \qquad i = 1, \ldots, 6,$$
$$x_1x_2x_3x_4x_5x_6 - 2.07\times 10^3 = f_1 \geq 0,$$
$$1 - 0.00062x_1x_4x_5(x_1+x_2+x_3) - 0.00058x_2x_3x_6(x_1+1.57x_2+x_4) = f_2 \geq 0.$$

Using the penalty function technique, an objective function was defined by

$$F = f + \{1000 \min[0, f_1]\}^2 + \{1000 \min[0, f_2]\}^2$$

and was optimized within the search domain

$$x_1, \ldots, x_6 \geq 0.$$

References

1. PRICE, W. L., *A Controlled Random Search Procedure for Global Optimization*, Toward Global Optimization 2, Edited by L. C. W. Dixon and G. P. Szego, North-Holland Publishing Company, Amsterdam, Holland, pp. 71–84, 1978.
2. PRICE, W. L., *Global Optimization by Controlled Random Search*, Journal of Optimization Theory and Applications, Vol. 40, pp. 333–348, 1983.
3. DUCKSBURY, P. G., *The Implementation of a Parallel Version of Price's (CRS) Algorithm on an ICL-DAP*, Hatfield Polytechnic, Technical Report No. 127, 1982.

4. DIXON, L. C. W., PATEL, K. D., and DUCKSBURY, P. G., *Experience Running Optimization Algorithms on Parallel Processing Systems*, Hatfield Polytechnic, Technical Report No. 138, 1983.
5. NELDER, J. A., and MEAD, R., *A Simplex Method for Function Minimization*, Computer Journal, Vol. 7, pp. 308-313, 1965.
6. BALLARD, D. H., JELINEK, C. A., and SCHINZINGER, R., *An Algorithm for the Solution of Constrained Generalized Polynomial Programming Problems*, Computer Journal, Vol. 17, pp. 261-266, 1974.