Proceedings of the 5th WSEAS Int. Conf. on SIMULATION, MODELING AND OPTIMIZATION, Corfu, Greece, August 17-19, 2005 (pp347-352)

Parallel Implementation of a Random Search Procedure: An Experimental Study

NIKOLAI K. KRIVULIN*
Faculty of Mathematics and Mechanics
St. Petersburg State University
28 University Ave., St. Petersburg, 198504
RUSSIA

DENNIS GUSTER and CHARLES HALL
G.R. Herberger College of Business
St. Cloud State University
720 4th Ave. South, St. Cloud, MN 56301
USA

Abstract: We consider discrete optimization problems with the objective functions which can be defined as a response to a controllable real-time process, or obtained through computer simulation. To solve the problems, a random search algorithm and its parallel implementation are developed. A performance analysis of both serial and parallel algorithms is given, and related numerical results are discussed.

Key-Words: global optimization, random search, parallel algorithms, performance analysis of algorithms

1 Introduction

Together with other global optimization techniques [1, 2, 3, 4], including Simulated Annealing, evolutionary algorithms, and the Tunneling method, random search presents a powerful approach to solving discrete optimization problems when the objective function is too complex to obtain the solution analytically, or does not have an appropriate analytical representation. One can consider the functions with their values being obtained as a response from a controllable real-time process, or being evaluated through computer simulation. The optimization problems become even more difficult to solve if the evaluation of the function presents a very time-consuming procedure as is normally the case when the function is determined via computer simulation.

To solve the difficult optimization problems above, we propose a random search technique based on the Branch and Probability Bound (BPB) approach introduced in [5] and further developed in [6, 4, 7]. The BPB approach actually combines the usual branch and bound search scheme with statistical procedures of parameter estimation based on random sampling.

The key feature of the BPB approach is that it allows one to examine several regions within a feasible set concurrently in a natural way. Therefore, when solving multiextremal optimization problems, BPB algorithms normally offer an advantage over the other global optimization techniques that concentrate the search only on a single feasible region, and so could easily miss the solution.

If the sampling procedure including evaluation of the objective function at the sample points takes much more time than the core part of a search algorithm, it is quite natural to arrange the procedure so that it could work in parallel.

In this paper, we present a BPB random search algorithm together with its parallel implementation. A performance analysis of the parallel implementation is given based on solution of some test problems. As our computational experience shows, the parallel algorithm has a quite good potential to speedup the solution time when evaluation of the objective function is time-consuming.

*The work was partially supported by NATO under Expert Visit Grant SST.EV.980134.

2 Problem and Solution Approach

We consider the problem of finding

$$x_* = \arg\min_{x \in X} f(x),$$

where X is a discrete feasible set, and f is a real-valued function. As examples of X, one can consider the set of integer vectors $x = (x_1, \ldots, x_n)$ with their components $x_i \in \{1, \ldots, m\}$ for each $i = 1, \ldots, n$, or the set of all permutations from the permutation group of order n.

In order to solve the problem, we propose a global random search algorithm based on the BPB approach. As with many other adaptive random search techniques, the algorithm actually employs random sampling with both the feasible set and the sample probability distribution over the set being modified with each new iteration designed to exploit information about the function behavior, obtained in the course of the previous search.

The BPB approach offers an efficient technique of reducing the feasible set and rebuilding the sample distribution. It involves the partitioning of the feasible set into subsets followed by a statistical procedure which estimates the prospectiveness of each subset for further consideration. The procedure evaluates a criterion based on sample data, which has a two-fold implementation. It allows one to reduce the feasible set by removing the subsets that have a low criterion, and so could hardly contain the solution.

On the other hand, evaluation of the criterion plays the key role in rebuilding the sample distribution. In fact, the new distribution is defined in such a way that it provides for more intensive sampling resulting in more promising subsets with higher values of the prospectiveness criterion.

3 Basic Components of the Algorithm

3.1 Prospectiveness Criterion

Evaluation of prospectiveness of a subset for further search provides the basis of BPB algorithms. Consider a prospectiveness criterion introduced in [5] (see also [4, 7]).

Let $Z \subset X$ be a subset of the feasible set X, $\Xi = \{x_1, \ldots, x_K\}$ be a sample from a probability distribution $P(dx)$ over X, and y_* is the minimum value of the function f over Ξ:

$$y_* = \min_{x \in \Xi} f(x).$$

Assuming that $\Xi \cap Z \neq \emptyset$, one can evaluate $y = f(x)$ for each $x \in \Xi \cap Z$ to obtain a sample $\Upsilon = \{y_1, \ldots, y_N\}$, where $N = |\Xi \cap Z|$ is the cardinality of $\Xi \cap Z$, and define $y_{(1)} \leq \cdots \leq y_{(N)}$ to be the ordered statistics associated with Υ.

The prospectiveness criterion for the subset Z is defined as

$$\varphi_\Xi(Z) = \left(1 - \left(\frac{y_{(1)} - y_*}{y_{(k+1)} - y_*}\right)^\alpha\right)^k, \qquad (1)$$

where k is an integer, and α is a parameter.

As it has been shown in [5], the criterion has a natural statistical interpretation. If $k \to \infty$ and $k^2/N \to 0$ as $N \to \infty$, then $\varphi_\Xi(Z)$ converges to the probability that

$$\min_{x \in Z} f(x) \leq y_*.$$

In practice, the value of k can be taken according to the following conditions. If $N \geq 10$, then

$$k = \begin{cases} \lfloor N/10 \rfloor, & \text{if } N < 100, \\ 10, & \text{if } N \geq 100; \end{cases}$$

otherwise, one has to expand the sample Ξ until $N = |\Xi \cap Z| \geq 10$, and then to try to evaluate the criterion once again.

The parameter α is actually determined by the behavior of the function f on the entire feasible set X, and it is normally unknown. To estimate α, suppose that $y_{(1)} \leq \cdots \leq y_{(N)}$ are ordered statistics corresponding to the entire sample Ξ over X. It can be shown [5], that the estimate

$$\widehat{\alpha} = \ln 5 \Big/ \ln \frac{y_{(k+1)} - y_{(1)}}{y_{(m+1)} - y_{(1)}} \qquad (2)$$

converges to α, if $k \to \infty$, $k^2/N \to 0$, and $m/k \to 0.2$ as $N \to \infty$. To evaluate $\widehat{\alpha}$, one normally takes $k = 10$, $m = 2$, and $N \geq 100$.

3.2 Representation of the Feasible Set

At each iteration of the algorithm, the current feasible set X is represented as $X = Z_1 \cup \cdots \cup Z_k$, where Z_j, $j = 1, \ldots, k$, are subsets of a common simple structure. The basic subset type, hyperballs or hypercubes with respect to a metric ρ are normally taken to provide for efficient partitioning and sampling procedures. Since for some discrete spaces (e.g., permutation groups), the concept of a hypercube is not appropriate, we restrict ourselves to hyperballs $B_r(z, \rho) = \{x | \rho(z, x) \leq r\}$, where r is the radius, and z is a center.

Starting with a hyperball $Z = B_r(z, \rho)$ of a radius $r = R$, where R is large enough to cover the initial set X at the first iteration, the algorithm consecutively decrements the radius of hyperballs with every new iteration so as to allow for reduction of the feasible set and thereby concentrating the search on more promising subsets.

3.3 Reduction and Partition of the Sets

The reduction procedure is based on the partitioning of the current feasible set X into a subset Z and its complement $X \setminus Z$. In order to decide, if the complement can be removed, the procedure first evaluates its related criterion (1) to get $\gamma = \varphi_\Theta(X \setminus Z)$, where Θ is the set of all sample points currently available. If the value of γ appears to be less than the fixed low bound δ, which determines the lowest level for subsets to be considered as candidates for further search, then the complement is removed.

The procedure actually combines reduction of the feasible set and partition of the reduced set into subsets, and can be described in more detail as follows. Suppose that r is the common radius of hyperballs, and δ is the low bound for the criterion (1). Let us define

$$z_1 = \arg\min_{x \in \Theta} f(x), \qquad Z_1 = B_r(z_1, \rho),$$

and consider the value of $\gamma_1 = \varphi_\Theta(X \setminus Z_1)$.

If $\gamma_1 < \delta$, then the subset $X \setminus Z_1$ can be removed since it has a very low prospectiveness level. Otherwise, when $\gamma_1 \geq \delta$, the procedure has to be continued. Now we take

$$z_2 = \arg\min_{x \in \Theta \setminus Z_1} f(x), \qquad Z_2 = B_r(z_2, \rho).$$

After evaluation of $\gamma_2 = \varphi_\Theta(X \setminus (Z_1 \cup Z_2))$ the procedure may be continued or ended depending on the value of γ_2. If continued, the procedure is repeated as long as there is a subset to remove.

It may appear that there are not enough sample points available to evaluate the criterion. In this case, one has to stop the procedure, go back to extend the sample Θ, and then start the procedure from the beginning.

Suppose that the procedure is repeated k times before meeting the condition of removing a subset. Upon completion of the procedure, we have the current feasible set X reduced to the union $Z_1 \cup \cdots \cup Z_k$, and the current set of sample points Θ reduced to $\Theta \cap (Z_1 \cup \cdots \cup Z_k)$.

3.4 Sample Probability Distribution

To make a decision on how to reduce the current feasible set, the algorithm implements a statistical criterion based on random sampling over the set. This makes the sampling procedure a key component of the algorithm. The procedure applies a probability distribution, which is first set to the uniform distribution over the initial feasible set X, and then modified with each new iteration.

Suppose that the current set X is formed by k subsets (hyperballs): $X = Z_1 \cup \cdots \cup Z_k$. The distribution $P(dx)$ over X can be defined as a superposition of a probability distribution over the set of hyperballs and the uniform distribution over each hyperball. With a probability p_j assigned to the hyperball Z_j, we have

$$P(dx) = \sum_{j=1}^{k} p_j Q_j(dx),$$

where $Q_j(dx)$ denotes the uniform distribution over Z_j, $j = 1, \ldots, k$.

The algorithm sets probabilities $p_1, \ldots, p_k$ to be proportional to the criterion (1) determined by their related hyperballs. In this case, the probabilities actually control the search, allowing the algorithm to put more new sample points into the hyperballs with higher probabilities.

To get the probabilities, one can evaluates $q_j = \varphi_\Theta(Z_j)$ for each $j = 1, \ldots, k$, and then take

$$p_j = q_j \Big/ \sum_{m=1}^{k} q_m.$$

Note that there may be not enough sample points available when evaluating q_j for some j. In this case, it is quite natural to set $q_j = \delta$. If it appears that all q_j equal 0, we set $q_j = 1$ for every $j = 1, \ldots, k$.

4 Random Search BPB Algorithm

Now we summarize the ideas described above in the presentation of the entire search algorithm.

The algorithm actually offers both global and local search capabilities. With each new iteration of the global search, the algorithm decrements the radius of the hyperballs by 1 until the radius achieves 1. All further iterations are performed with the radius fixed at 1 until a local minimum is found. We consider the best sample

point found as a local minimum if all its nearest neighbors have already been examined, and are so included in the current set of sample points.

Algorithm 1.

Step 1. Fix values for K, R and δ. Set $i = 1$, $r_0 = R$, $\Gamma_0 = \emptyset$, $X_1 = X$, and $P_1(dx)$ to be the uniform distribution over X_1.

Step 2. Get a sample $\Xi_i = \{x_1^{(i)}, \ldots, x_K^{(i)}\}$ from $P_i(dx)$. For each $x \in \Xi_i$, evaluate $f(x)$.

Step 3. Set $\Theta_i = \Gamma_{i-1} \cup \Xi_i$, and find

$$y_*^{(i)} = \min_{x \in \Theta_i} f(x), \qquad x_*^{(i)} = \arg\min_{x \in \Theta_i} f(x).$$

Step 4. If $i = 1$, then evaluate $\widehat{\alpha}$ with (2).

Step 5. Put $r_i = \max\{r_{i-1} - 1, 1\}$.

Step 6. If $r_i = 1$ and $B_1(x_*^{(i)}, \rho) \subset \Theta_i$, then STOP.

Step 7. Set $k = 1$, $U_0^{(i)} = \emptyset$.

Step 8. Find $z_k^{(i)} = \arg \min_{x \in \Theta_i \setminus U_{k-1}^{(i)}} f(x)$.

Step 9. Set $Z_k^{(i)} = B_{r_i}(z_k^{(i)}, \rho)$, $U_k^{(i)} = U_{k-1}^{(i)} \cup Z_k^{(i)}$.

Step 10. If $|\Theta_i \cap (X_i \setminus U_k^{(i)})| \geq 10$, then evaluate $\gamma_k^{(i)} = \varphi_{\Theta_i}(X_i \setminus U_k^{(i)})$. Otherwise, replace Γ_{i-1} with Θ_i, and go to Step 2.

Step 11. If $\gamma_k^{(i)} \geq \delta$, then replace k with $k + 1$, and go to Step 8.

Step 12. Set $X_{i+1} = U_k^{(i)}$.

Step 13. Set $\Gamma_i = \Theta_i \cap U_k^{(i)}$.

Step 14. For each $j = 1, \ldots, k$, evaluate

$$q_j^{(i)} = \begin{cases} \varphi_{\Gamma_i}(Z_j^{(i)}), & \text{if } |\Gamma_i \cap Z_j^{(i)}| \geq 10, \\ \delta, & \text{otherwise.} \end{cases}$$

Step 15. For each $j = 1, \ldots, k$, evaluate

$$p_j^{(i)} = q_j^{(i)} \Big/ \sum_{m=1}^{k} q_m^{(i)}.$$

Step 16. Set $P_{i+1}(dx) = \sum_{j=1}^{k} p_j^{(i)} Q_j^{(i)}(dx)$.

Step 17. Replace i with $i + 1$, and go to Step 2.

5 Parallel Version of the Algorithm

In many practical situations, generating of sample points and/or evaluation of the objective function at the points are a time-consuming task. At the same time, the sampling procedure can normally be split into independent subprocedures each producing its own part of the sample. Therefore, when a sampling procedure takes sufficiently more time than the other steps of the algorithm, one can get higher performance by rearranging the procedure to work in parallel.

To investigate the performance, both serial and parallel versions of the algorithm were coded in C++ under the Linux RedHat 8.0 operating system. The parallel code is based on LAM 6.5.9. implementation [8] of the Message Passing Interface (MPI) communication standard [9].

The parallel application consists of two modules; first one intended to run on the master computer, and the second designed to support slave computers. The code running on the master controls the communication with the slaves, and performs all the steps of the algorithm except for the sampling procedure.

The master computer starts operating by establishing connections and broadcasting some general information, including the parameters n and m of the feasible set, among the slave computers. At each iteration of the algorithm, it sends requests to all slaves to produce samples. The request to a particular slave includes the current radius of hyperballs, and its own list of hyperball centers accompanied by the numbers of points to be generated in each hyperball.

The sample points and their related values of the function are sent back to the master. Upon completion of the current iteration, the next iteration is initiated until the stop condition is met.

The software was tested on a cluster of Intel Pentium II/ 500MHz/ 128Mb RAM/ 10Gb HDD computers with 100BaseTX 100Mbit LAN.

6 Test Problems

To test the algorithm, simple unimodal and multimodal functions are considered (see, e.g., [3] for more examples). We assume them to be defined on the set X of integer vectors $x = (x_1, \ldots, x_n)$ with $x_i \in \{1, \ldots, m\}$ for each $i = 1, \ldots, n$, provided that m is even, and $m < n$.

First, we consider an integer analog of the De

Jong's function:

$$f(x) = \sum_{i=1}^{n} (x_i - m/2)^2. \quad (3)$$

The function is unimodal with $f(x_*) = 0$ at the point $x_* = (m/2, \ldots, m/2)$.

The following integer function is of the Rastrigin type:

$$f(x) = nm + \sum_{i=1}^{n} (x_i - m/2)^2 - m\cos(k\pi(x_i - m/2)/m), \quad (4)$$

where k is an integer parameter. If $k = 0$, the function coincides with De Jong's function, and it is unimodal. As k increases, the function becomes multimodal. It has the global minimum $f(x_*) = 0$, where $x_* = (m/2, \ldots, m/2)$.

The function

$$f(x) = \sum_{i=1}^{n} |x_i - m/2| + \sum_{i=1}^{n-1} |x_i - x_{i+1}| + |x_n - x_1| \quad (5)$$

has the local minimum $f(x_*^{(i)}) = n|i - m/2|$ at $x_*^{(i)} = (i, \ldots, i)$, $i = 1, \ldots, m$; and the global minimum $f(x_*) = 0$ at $x_* = (m/2, \ldots, m/2)$.

7 Serial Algorithm Tests

We begin with the results of testing a serial version of the algorithm code, which actually does not include any MPI support. A series of test runs were performed with the test functions defined on the feasible set X with $n = 200$, $m = 50$. The low bound δ was set to 0.1, the sample size K and the initial radius of hyperballs R were both uniformly varied from 50 to 150 by 10.

Let S be the time spent generating the samples and evaluating the function (sampling time), and A be the time the algorithm takes to utilize the samples (algorithm time). The total solution time of the algorithm can be represented as

$$T_S = S + A. \quad (6)$$

Finally, let us denote the total number of sample points examined by the algorithm during solution process, as N, and define $S_1 = S/N$ and $A_1 = A/N$ to represent average sampling and algorithm time for one sample point.

In Table 1, we present a brief summary of the test results for the serial algorithm for each test function. The summary actually includes the average times and numbers of examined points, calculated over the entire series of test runs.

Test function	N	Run time (sec.)			Point time (msec.)	
		T_S	S	A	S_1	A_1
(3)	132994	212	147	66	1.10	0.50
(4)	129270	675	554	121	4.29	0.94
(5)	199244	914	343	570	1.72	2.86

Table 1: Summary results for the test runs.

8 Parallel Algorithm Analysis

The total time the parallel algorithm takes to get the solution can be written as

$$T_P = S/p + A + C, \quad (7)$$

where p is the number of slaves, C is the time the master spends on transmission of control/sample data to/from slaves (communication time).

Clearly, with (6) and (7) the speedup the parallel algorithm can achieve using one master and $p \geq 1$ slave computers, can be represented as

$$\sigma(p) = \frac{T_S}{T_P} = \frac{S + A}{S/p + A + C}.$$

Let us denote the average data transmission time for one sample point as C_1. Assuming the amount of control data the master sends to be well below that of the sample data it receives, one can expect $C_1 \approx C/N$. Now we can write

$$\sigma(p) \approx \frac{S_1 + A_1}{S_1/p + A_1 + C_1}. \quad (8)$$

With (8) one can examine the conditions required for the parallel algorithm to achieve a true speedup, and estimate actual speedup in particular problems. Specifically, in order to get a speedup $\sigma > 1$, one should have

$$r = \frac{S_1}{C_1} > \frac{p}{p - 1}.$$

If the algorithm time A_1 appears to be much less than both the sampling time S_1 and the communication time C_1, we have the speedup

$$\widetilde{\sigma}(p) = \frac{S_1}{S_1/p + C_1} = \frac{rp}{r + p}.$$

Since at a fixed r it holds that $\widetilde{\sigma}(p) \to r$ as $p \to \infty$, one can conclude that r presents the maximum asymptotic speedup of the parallel algorithm. Note, however, that actual speedup can be much lower than the upper bound r.

To evaluate expected speedup, we need an estimate of the communication time for one sample C_1. Our computational experience shows that the average time to transmit the point data is approximately equal to 1 millisecond.

With $C_1 = 1$, and parameters S_1 and A_1 taken from Table 1, one can apply (8) to evaluate the speedup for any $p \geq 1$ (see Fig. 1).

Speedup

Fig. 1: Predicted speedup for the test functions.

As it easy to see, one can expect an actual speedup only for the function (4) having the best value of $r = S_1/C_1 \approx 4.29$. For the other functions, any sufficient speedup can hardly be achieved because of the low level of $r = 1.10$ for (3), and a high magnitude of $A_1 = 2.86$ for (5).

In order to evaluate actual speedup for function (4), several series of test runs were performed for each $K = 50, 100, 150$, and $p = 1, 2, 3, 4, 5$. One series involves a particular run for every value of R varied from 50 to 150 by 10. The average total solution times over the values of R for each series is represented in Fig. 2, where $p = 0$ corresponds to the serial version of the algorithm.

One can see that the best speedup achieved was about 1.6-1.7 when using one master and 3 slave computers. Although the speedup appears to be relatively small, it does demonstrate the potential of parallelization. Since evaluation of the functions involves only a few operations, the sampling procedure does not take much time to produce samples. As the performance analysis shows,

Total Time (sec.)

Fig. 2: Average total solution time.

if this procedure is time-consuming, one can expect to achieve even greater efficiency.

References:

[1] R. Horst, H. Tuy, *Global Optimization*, Springer, Berlin, 1990.

[2] R. Horst, P.M. Pardalos, *Handbook of Global Optimization*, Kluwer, Dordrecht, 1995.

[3] M.A. Potter, K. De Jong, A Cooperative Coevolutionary Approach to Function Optimization, *Parallel Problem Solving From Nature*, Springer, Berlin, 1994, pp. 249-257.

[4] A.A. Zhigljavsky, *Theory of Global Random Search*, Kluwer, Dordrecht, 1991.

[5] A.A. Zhigljavsky, *Mathematical Theory of Global Random Search*, Leningrad University, Leningrad, 1985. (in Russian)

[6] A.A. Zhigljavsky, N.K. Krivulin, Optimization of Computer Systems Through Simulation and Random Search, *Problem-Oriented Tools of Computer System Performance Evaluation*, Kazan Aeronautical Institute, Kazan, 1989, pp. 40-50. (in Russian)

[7] A.A. Zhigljavsky, Semiparametric Statistical Inference in Global Random Search, *Acta Applicandae Mathematicae*, Vol. 33, 1993, pp. 69-88.

[8] *LAM/MPI User's Guide*. http://www.lam-mpi.org, 2003.

[9] W. Gropp, E. Lusk, A. Skjellum, *Using MPI: Portable Parallel Programming With the Message Passing Interface*, MIT Press, 1994.