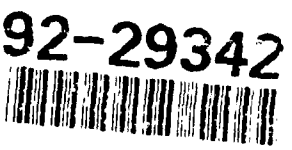
(20)

A TRIDENT SCHOLAR PROJECT REPORT

NO. 190

Implementation of a Parallel Stochastic Solving Method for Linearly Constrained Concave Global Optimization Problems Using Parallel Computing"

UNITED STATES NAVAL ACADEMY
ANNAPOLIS, MARYLAND



"Implementation of a Parallel Stochastic Solving Method for Linearly Constrained Concave Global Optimization Problems Using Parallel Computing"

A Trident Scholar Project Report

by

Midshipman Matthew F. McLaughlin, Class of 1992

U. S. Naval Academy

Annapolis, Maryland

Adviser: Assistant Professor Andrew T. Phillips
Computer Science Department

Accepted for Trident Scholar Committee

Chair

8 May 1992
Date

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	8 May 1992	Final 1991/92

4. TITLE AND SUBTITLE
IMPLEMENTATION OF A PARALLEL STOCHASTIC SOLVING METHOD FOR LINEARLY CONSTRAINED CONCAVE GLOBAL OPTIMIZATION PROBLEMS USING PARALLEL COMPUTING

5. FUNDING NUMBERS

6. AUTHOR(S)
McLaughlin, Matthew F.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S.Naval Academy, Annapolis, Md.

8. PERFORMING ORGANIZATION REPORT NUMBER
U.S.N.A. - TSPR; 190 (1992)

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Accepted by the U.S.Trident Scholar Committee

12a. DISTRIBUTION/AVAILABILITY STATEMENT
This document has been approved for public release; its distribution is UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

A parallel stochastic algorithm is implemented for solving the linearly constrained concave global optimization problem. The algorithm uses a multistart technique which repeatedly employs two phases, the global phase and the local phase. The global phase creates a random search direction to find a vertex of the linearly constrained feasible region. The local phase begins from that vertex and solves for a local minimum. The algorithm repeats the global and local phases to find all the local minima. The algorithm was programmed in FORTRAN on the Connection Machine CM-2 and Cray X-MP EA/464 supercomputers. Computational results are presented for more than 200 test problems in three categories: known problems from the literature, randomly generated concave quadratic problems, and randomly generated fixed-charge problems. The test problems fro the literature were run on both the Cray X-MP and the CM-2 and resulted in an analysis of the stochastic algorithm's efficiency on each machine. Computational results from randomly generated quadratic and fixed-charge functions resulted in an examination of how particular characteristics affect the difficulty of the problem. Lastly, the stochastic algorithm's performance on the Cray X-MP was analyzed and modeled using the timing results for random concave quadratic function problems.

14. Subject terms: algorithms; parallel programming(computer science); stochastic analysis; mathematical optimization; mathematical programming; connection machines

15. NUMBER OF PAGES
96

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	

Implementation of a Parallel Stochastic Solving Method for Linearly Constrained Concave Global Optimization Problems Using Parallel Computing

M.F. McLaughlin

Abstract

A parallel stochastic algorithm is implemented for solving the linearly constrained concave global optimization problem. The algorithm uses a multistart technique which repeatedly employs two phases, the global phase and the local phase. The global phase creates a random search direction to find a vertex of the linearly constrained feasible region. The local phase begins from that vertex and solves for a local minimum. The algorithm repeats the global and local phases to find all the local minima. Because the total number of local minima is unknown, an optimal bayesian estimate is used to determine when the algorithm can terminate by assessing the probability that the global minimum has been found. The global minimum solution is found by taking the smallest function value of all the local minima vertices. The algorithm was programmed in FORTRAN on the Connection Machine CM-2 and Cray X-MP EA/464 supercomputers. Computational results are presented for more than 200 test problems in three categories: known problems from the literature, randomly generated concave quadratic problems, and randomly generated fixed-charge problems. The test problems from the literature were run on both the Cray X-MP and the CM-2 and resulted in an analysis of the stochastic algorithm's efficiency on each machine. Computational results from randomly generated quadratic and fixed-charge functions resulted in an examination of how particular characteristics affect the difficulty of the problem. Lastly, the stochastic algorithm's performance on the Cray X-MP was analyzed and modeled using the timing results for random concave quadratic function problems.

Preface

I would like to preface my Trident Scholar Report by offering my thanks to those people who helped and supported me throughout the past year. I would like to acknowledge the staff at the Minnesota Supercomputer Institute for hosting me while I worked in their facility and answering numerous questions throughout my research. I owe gratitude to the entire Computer Science Department of the United States Naval Academy for their constant support of my project. In particular, I would like to give my deepest thanks to my advisor, Dr. Andrew T. Phillips. His enthusiasm, guidance, and ideas were the primary reasons for my success. Without his help, I would not have been able to produce this report.

Contents

Section 1
Introduction

This paper considers the implementation of a parallel approach for solving the problem :

$$\begin{array}{c} \text{global min } \Psi(x) \\ x \in \Omega \end{array}$$

(GP)

where $\Psi(x)$ is an arbitrary differentiable strictly concave function, the feasible region $\Omega = \{ x : Ax \leq b, x \geq 0 \}$ is assumed to be nonempty and bounded, and $x \in R^n$, $A \in R^{m \times n}$, and $b \in R^m$. This problem, in its most general form (GP), is known as the linearly constrained concave global minimization problem.

Many special cases of this general problem (GP) exist and are of particular interest. One special case is the concave quadratic global minimization problem. For this type of problem, $\Psi(x) = \frac{1}{2} x^t Q x + c^t x$ where $Q \in R^{n \times n}$ is symmetric and negative definite (i.e., all eigenvalues of Q are < 0). This special case of the general concave global minimization problem is known to be NP-hard (Phillips 1988), and therefore the more general problem (GP) is also NP-hard. A problem is considered to be NP-hard if it can be solved in nondeterministic polynomial time, but no known algorithm solves it in *deterministic* polynomial time. Simply stated, this means that these problems are intractable in the worst case and no method is known which can solve them in polynomial time (Martin 1991). An NP-hard problem can have special cases for which reasonable instances can be solved in polynomial time, but no algorithm is known which exhibits polynomial behavior for all instances. Since this general

problem (GP) is NP-hard, the computation time of the implemented algorithm is, in the worst case, expected to increase exponentially in relation to the number of problem variables (n).

One characteristic of the problem (GP) which is central to nearly all solution techniques, including the parallel algorithm to be considered shortly, is that the global minimum must occur at a vertex of the convex polytope Ω. In fact, due to the strict concavity of $\Psi(x)$, every local minimum point must be a vertex of Ω (Phillips 1988). More precisely, strict concavity of $\Psi(x)$ is *required* because, without it, a function could meet a linear constraint in such a way as to have the entire active constraint be a minimum function value. Hence, there would be *infinitely* many local and global minima. Therefore, we require strict concavity of $\Psi(x)$ to force the minimum to occur at a vertex. Because vertices are central to problem (GP), linear programming is an essential part of any computational algorithm used to solve it.

Techniques used to solve concave minimization problems can be classified into two general categories: deterministic and stochastic methods. Deterministic methods solve the problem with certainty while stochastic methods invoke probabilistic techniques to provide an answer within a range of certainty. Thus, stochastic methods can often decrease the amount of time necessary to solve a particular problem at the expense of not having absolute certainty in the answer.

There have been many deterministic techniques proposed for solving the constrained concave global minimization problem of the general form (GP). Extreme point ranking was the first method proposed (Cabot and Francis 1970). This algorithm searches among the vertices of Ω for the global minimum. Generally, linear underestimating functions are used to speed up the search through logical elimination. This type of algorithm is an example of vertex enumeration which has

proven to be inefficient for even reasonably small sized problems. In the worst case, vertex enumeration results in the need to test every vertex. Given a problem with n variables and m linear constraints, the maximum number of vertices of the polyhedron Ω is :

$$\left(\begin{matrix} m - \text{int}\left[\frac{n+1}{2}\right] \\ m - n \end{matrix}\right) \cdot \left(\begin{matrix} m - \text{int}\left[\frac{n+2}{2}\right] \\ m - n \end{matrix}\right)$$

where int[] rounds down to the nearest integer (Chvátal 1983). Obviously, as the number of variables and constraints increase, the number of vertices increases exponentially thus making it impossible to solve reasonably large sized problems using a vertex enumeration algorithm.

A cutting plane algorithm introduces additional linear constraints to reduce the size of the feasible region without eliminating the global minimum from consideration (Tui 1964). Branch and bound techniques partition the region defined by the linear constraints and then eliminate certain sub-regions based on lower and upper bounds on the global minimum (Falk and Soland 1969). Since it has previously been shown that concave minimization is equivalent to bilinear programming (Thieu 1980), deterministic algorithms used to solve bilinear programming problems have also been applied to (GP). Lastly, approximating subproblems (Falk and Hoffman 1976), where underestimation techniques are applied to smaller regions of the overall problem, has been one of the most successful techniques for solving concave minimization. However, none of these methods has been successfully applied to problems of the form (GP) with $n \geq 25$.

A stochastic process is a collection of random variables ordered over time, which are all defined on a common sample space (Law and Kelton 1991). Stochastic

models produce output that is an estimate of the true characteristics of the function to within a particular degree of certainty. The algorithm presented in this paper is stochastic because it repeatedly employs a random search procedure to find a vertex from the defined and limited set of feasible region vertices. Stochastic methods have previously been proposed for the unconstrained case of concave global minimization, but never for the constrained case. Thus, the implemented algorithm will be compared to the deterministic methods of solving (GP) to see if stochastic methods are a feasible alternative to present day deterministic ones.

Although these and other deterministic algorithms have been proposed for concave minimization, few have been extensively tested computationally. Those that have been tested are generally limited to problems with a small number of variables ($n \leq 25$), and have been ineffective for large problems. One of the main goals of this research is to extensively test the proposed stochastic approach for feasibility on a large number of problems with a greater size than those previously tested.

Section 2
Applications

Many applications of the linearly constrained concave global optimization problem exist in a variety of fields of interest. Two such examples are from the areas of economic production and microchip design.

Economic production problems exist when a company's objective is to maximize profit, but has limited manufacturing resources to be allocated. An example is a textile company which has the ability to manufacture three products: shirts, shorts, and pants. Each unit of each type of product produced costs a certain amount in hours of labor and square yards of cloth, both of which are limited resources. The limits on these resources are what create the linearly constrained "feasible region" over which the negative of the profit function is minimized, (note that the profit function is negated so that it can be applied to the general concave *minimization* problem). The negative of the profit function is made up of the fixed costs of production minus the aggregate profit of the produced goods. The aggregate profit is simply the sum of each product multiplied by its per item profit, (sales price minus variable cost). The fixed cost of production is the cost of renting a piece of machinery to produce a good. Note that this cost is independent of the number of items produced but depends only on which items are produced. This cost is what makes this type of production problem a "fixed-charge" problem. Further details on the exact formulation and results of computational testing of fixed-charge problems are presented in section 13.

Microchip design is a second area to which problem (GP) can be applied. The basic concept of microchip design is to pack the necessary components into the

available area on the chip, while minimizing the distance between those elements which must be electrically connected. The way in which problem (GP) applies to microchip design involves two steps. First, the orientation and placement of the components onto the chip are formulated as constraints. Linear inequalities which relate to orientation must be created, since a given rectangular component can be aligned either vertically or horizontally. Likewise, the linear inequalities for packing must take into account the limited height and width when placing the elements onto the chip. Lastly, linear constraints are created to assure that no components overlap. A concave quadratic distance formula, which is the aggregate distance between all the components within the chip that need to be electrically connected, is then minimized over the linear feasible space to provide a "minimum distance" chip design.

Section 3
Theory and Algorithm

The proposed global optimization algorithm is a multistart technique which repeatedly employs two phases, the global phase and the local phase. The global phase creates a random search direction to find a vertex on the feasible region Ω. The local phase begins from this vertex and attempts to find a local minimum. Although the objective is to find the unknown global minimum of $\Psi(x)$, the algorithm proceeds by finding *all* of the local minima, and hence the global minimum is found as well. Unfortunately, the total number of local minima is unknown, therefore, an optimal bayesian estimate of the number of local minima must be used to determine at which point the algorithm can terminate. The resulting global minimum is simply found by taking the smallest (in function value) of the local minima.

The local phase of the stochastic method, which results in a local minimum, requires the repeated solution of a series of linear programs. These linear programs are based on the following theorem (for further details and proofs of theorems presented in this section, see Phillips, Rosen, and Van Vliet 1991):

Theorem 1: Let v be a vertex of Ω. If , starting from vertex v, v' solves the linear program

$$\begin{aligned} &\min \nabla\Psi(v)^t (x - v) \\ &x \in \Omega \end{aligned}$$

(LP)

then either

1) $v = v'$ and hence v' is a Karush-Kuhn-Tucker point for problem (GP), or

2) $v \neq v'$ and hence $\Psi(v') < \Psi(v)$.

Note that theorem 1 solves for a vertex of (GP) which is a Karush-Kuhn-Tucker point, but *not necessarily* a local minimum. The set of Karush-Kuhn-Tucker points for a given problem (GP) is in fact a superset of all the local minima of the problem. While in most instances when $v = v'$, the vertex found is indeed a local minimum, there are cases where it is not.

For example, when the gradient of $\Psi(v)$ is orthogonal to the active constraints (those constraints which define the vertex), or a vertex coincides precisely with the global maximum of $\Psi(x)$, then theorem 1 solves for a Karush-Kuhn-Tucker point which is not a local minimum. Figure 3.1 shows the two-dimensional case of the gradient of $\Psi(x)$ being orthogonal to the active constraint.

The two dimensions in Figure 3.1 represent the two variables over which the function is minimized. The linearly constrained feasible region is the interior of the triangular polytope. The global maximum of $\Psi(x)$ is shown by the heavy dot, and the level lines, those regions of constant function value, emanating from this point show the progressively decreasing function values of $\Psi(x)$. Thus, the global maximum lies at some value above the two-dimensional variable plane and the level lines lie at some function value less than the global maximum. The concave function is best envisioned as a "upside-down bowl" lying over the two-dimensional space of the feasible region.

Figure 3.2 shows a two-dimensional example of a vertex of Ω coinciding with the global maximum of $\Psi(x)$.

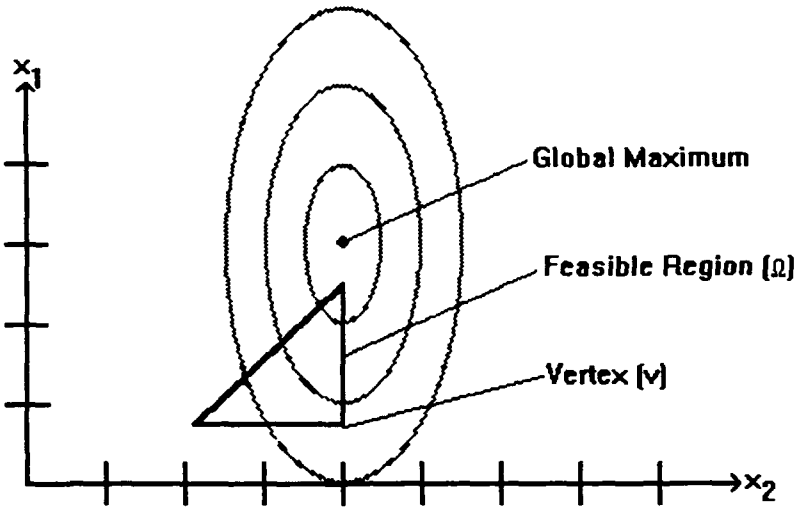


Figure 3.1: Active Constraints Orthogonal to the Gradient of Y(x)

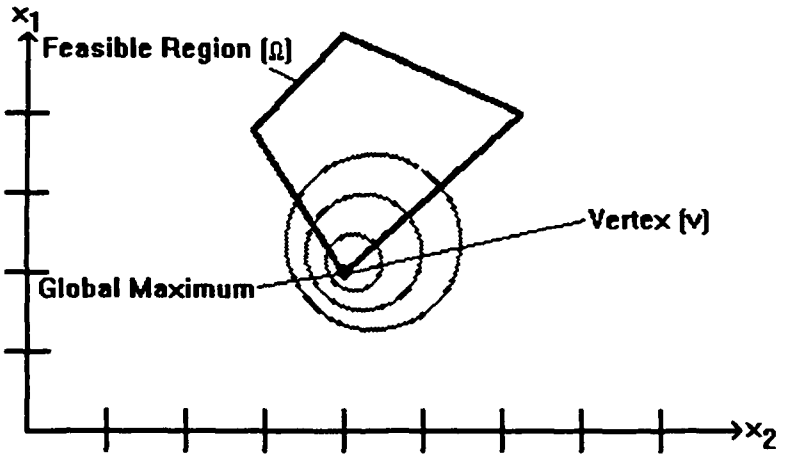


Figure 3.2: Global Maximum Coinciding with a Vertex of the Feasible Region

In Figure 3.1, the lower right-hand vertex of the triangular feasible region would be found as a Karush-Kuhn-Tucker point because the active constraint which is the base of the triangle is orthogonal to the gradient at that vertex (the gradient would be pointing right at the global maximum). In Figure 3.2, the bottom vertex of the feasible region polytope is not a local minimum since it coincides with the global maximum of the concave function. However, this vertex will be found as a Karush-Kuhn-Tucker point given the previous method of theorem 1. Although the proposed algorithm uses the requirements for a Karush-Kuhn-Tucker point as an approximate definition of a local minimum, the possible inconsistencies between the definitions, demonstrated by Figures 3.1 and 3.2, should not greatly affect the progress of the algorithm.

To be more precise, the probability that a vertex of the feasible region occurs exactly at the global maximum of $\Psi(x)$ is low. Compound this with the fact that the local phase of the algorithm follows the negative gradient of the function along constraints to find a new vertex at a *lesser* function value. Thus, the local phase could never find such a global maximum unless it is found first by the global phase's random search. In such a case, the local phase would terminate at $v = v'$ immediately and the global maximum would indeed be classified as a local minimum. The chances are similarly remote that the active constraints are orthogonal to the gradient of $\Psi(v)$, creating a "saddle point" coincident with the vertex.

Despite the fact that either of these events is improbable, the algorithm will not have a problem dealing with either of these situations if they were to arise. Neither of the special cases in which Karush-Kuhn-Tucker points are incorrectly classified as local minima could be considered as potential answers to (GP). Obviously, the global maximum is not the global minimum of (GP). Likewise, a

vertex which is a "saddle point" is a Karush-Kuhn-Tucker point that is not a local minimum. In this case the vertex's active constraints are orthogonal to the function gradient. Again, it is obvious that this vertex could not be the global minimum of (GP). Therefore, when the algorithm terminates and searches those "estimated" local minima it found, of which some are possibly Karush-Kuhn-Tucker points that are not local minima, it will still retrieve the least of the *actual local minima* as the correct answer for (GP).

Although finding a Karush-Kuhn-Tucker point that is not a local minimum will not affect the answer that the proposed stochastic algorithm finds, it will have an effect on the optimal bayesian estimate of the number of local minima. Therefore, this potential problem could adversely affect the running time of the algorithm. However, since it would be a very rare occurrence, it is not expected to be a major problem in the computational implementation of the stochastic algorithm.

The proposed stochastic method is based on solving a series of linear programs of the form (LP). The local phase will solve problem (LP) repeatedly for Karush-Kuhn-Tucker points, which include the set of local minima, until the optimal bayesian estimate shows that based on the number of "global phase" trials run and the number of distinct local minima found, all existing local minima have been found. For a strictly concave function, the number of Karush-Kuhn-Tucker points, and hence the number of local minima, is finite because Karush-Kuhn-Tucker points only occur at vertices (with the possible exception of the global *maximum*), and therefore Ω has a finite number of vertices. Hence, let $K = \{v_1, v_2, \ldots, v_\kappa\}$ represent the set of Karush-Kuhn-Tucker vertices of Ω for problem (GP).

We now define the *region of attraction* of a Karush-Kuhn-Tucker vertex $v \in K$, denoted by $R(v)$, to be the set of all search directions $u \in \mathbf{R}^n$ such that the following local search procedure results in obtaining the vertex v:

1. Set $j := 1$ and solve the linear program

$$\min u^t x$$
$$x \in \Omega$$

to get to the vertex z_0.

2. Starting from vertex z_{j-1} solve the linear program

$$\min \nabla\Psi(z_{j-1})^t (x - z_{j-1})$$
$$x \in \Omega$$

to get the vertex z_j.

3. If $\Psi(z_j) \neq \Psi(z_{j-1})$ then set $j := j + 1$ and go to step (2).

Otherwise stop.

The region of attraction for a specific Karush-Kuhn-Tucker point is simply that set of search directions which, when following the procedure above, results in obtaining that same Karush-Kuhn-Tucker vertex. Theorem 2, below, tells us that every Karush-Kuhn-Tucker point has at least one search direction in its region of attraction, and therefore can be found by the stochastic algorithm using *some* search direction. It also tells us that every possible search direction will lead to a Karush-Kuhn-Tucker point since the union of all the regions of attraction for a given problem results in $\mathbf{R}^n$.

Theorem 2: The regions of attraction $R(v_1)$, $R(v_2)$, ..., $R(v_\kappa)$ are nonempty and $R(v_1) \cup R(v_2) \cup \ldots \cup R(v_\kappa) = \mathbf{R}^n$.

Finally, theorem 3 tells us that if we take an infinite number of uniform random search directions, then we are certain to find every possible Karush-Kuhn-Tucker vertex, and hence all local (and therefore global) minima as well.

Theorem 3: If the search directions $u_1, u_2, \ldots, u_N$ are chosen from a uniform distribution over $\mathbf{R}^n$, then as $N \rightarrow \infty$, every $v_i \in K$, for $i = 1, \ldots, \kappa$, will be found with probability 1.

The algorithm can terminate once all of the Karush-Kuhn-Tucker vertices $v \in K$ have been found. Unfortunately, the number of Karush-Kuhn-Tucker vertices, κ, is unknown. Theorem 3 states that given an infinite number of random search trials, the set of Karush-Kuhn-Tucker points can be found with certainty. Obviously, running an infinite number of trials is computationally impossible. Hence, some reliable method of estimating κ is necessary to practically implement the algorithm. As mentioned earlier, only in certain rare cases is a Karush-Kuhn-Tucker point of problem (GP) not also a local minimum of $\Psi(x)$ over the feasible region, Ω. Therefore, a reliable method for approximating the number of local minima can also be used to estimate the number of Karush-Kuhn-Tucker vertices, κ. This estimate of the number of local minima is called an *optimal bayesian estimate* and was adapted from theorems put forth by Boender and Rinnooy Kan (1987) and is summarized as theorem 4:

Theorem 4: Let ω be the number of different observed local minima obtained as a result of performing N uniformly distributed random local searches. The optimal bayesian estimate of the number of local minima, for $N \geq \omega + 3$, is:

$$\frac{\omega * (N - 1)}{N - \omega - 2}.$$

Byrd, Dert, Rinnooy Kan, and Schnabel (1990) suggest that a practical implementation of theorem 4 would terminate the algorithm when this estimate exceeds ω by less than 0.5. A more general stopping rule would terminate the algorithm when:

$$\omega \leq \frac{\omega * (N - 1)}{N - \omega - 2} \leq \omega + \delta$$

for some $0 < \delta < 1$. The right hand side of this expression can be rewritten in the form:

$$N \geq \frac{\omega^2 + \omega}{\delta} + \omega + 2.$$

(SR1)

Thus, (SR1) uses the optimal bayesian estimate of the number of local minima to terminate. The constant δ allows a variation in the "strictness" of the stopping criteria. A larger δ requires fewer "global phase" trials (N) to be run per unique local minimum than a smaller δ.

One particular characteristic of this stopping rule that merits consideration is the relation of the number of local searches performed, N, to the number of local minima found, ω. Note that as new local minima are found, the number of trials that must be run increases quadratically. It would seem that as more local minima are found by the algorithm, then less trials are needed to find the global minimum. However, this stopping rule states that as more local minima are found by the algorithm, the number of trials must *increase quadratically* to satisfy the bayesian

stopping criteria. This factor is especially detrimental in larger problems since the number of local minima tends to increase *exponentially* in relation to the number of problem variables. This led to the development of the stopping rule based on the following theorem, also from Boender and Rinnooy Kan (1987):

Theorem 5: Let ω be the number of different observed local minima obtained as a result of performing N uniformly distributed random local searches. The optimal bayesian estimate of the total relative volume of the observed regions of attraction, for $N \geq \omega + 2$, is:

$$\frac{(N - \omega - 1) * (N + \omega)}{N * (N - 1)}.$$

Note that this value can be rewritten as:

$$1 - \frac{\omega * (\omega + 1)}{N * (N - 1)}.$$

Which satisfies the relation:

$$1 - \frac{\omega * (\omega + 1)}{N * (N - 1)} \geq 1 - \left[\frac{\omega + 1}{N - 1}\right]^2.$$

Thus, we can develop a stopping rule based on t, the fraction of the domain explored. This second stopping rule terminates the algorithm when:

$$1 - \left[\frac{\omega + 1}{N - 1}\right]^2 \geq t$$

or, rewritten in the form similar to (SR1) as:

$$N \geq \frac{\omega + 1}{\sqrt{1 - t}} + 1.$$
(SR2)

Thus, (SR2) uses the optimal bayesian estimate of the total relative volume observed to terminate the program. Note that in (SR2), N and ω are linearly related and thus (SR2) is a much more effective stopping criteria for larger problems. Figures 3.3 and 3.4 compare the stopping rules graphically.

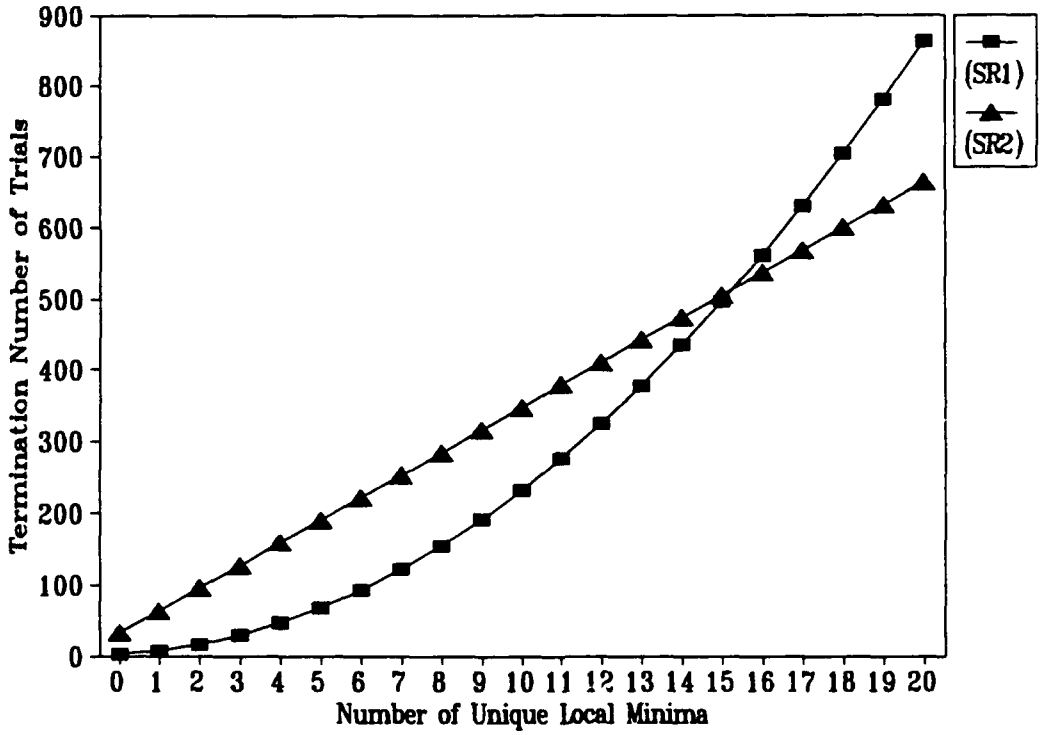


Figure 3.3: Comparison of Stopping Criteria for Low Numbers of Local Minima
($\delta = 0.5$, $t = 0.999$)

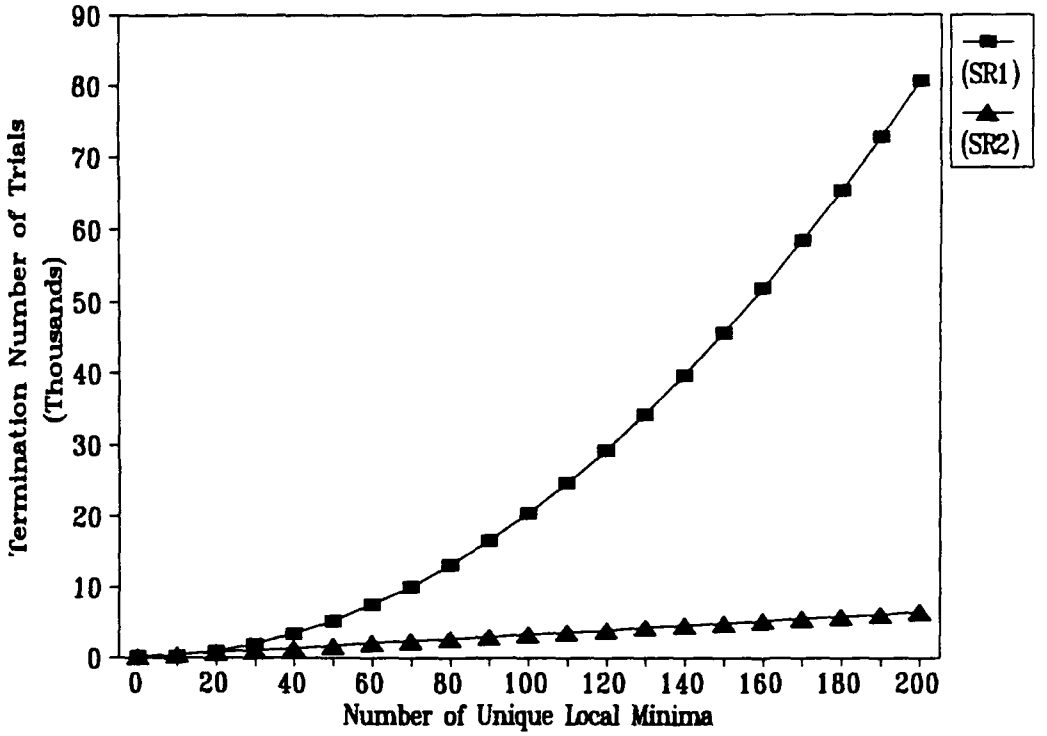


Figure 3.4: Comparison of Stopping Criteria for Greater Numbers of Local Minima
($\delta = 0.5$, $t = 0.999$)

From the graphs it is easy to see the relationship between the two stopping rules. These figures used a reasonably standard value of 0.5 for the strictness of (SR1), but a rigorous value of 0.999 for the fraction of domain explored in (SR2). Figure 3.3 shows that for problems with a small number of unique local minima, ω, (SR1) needs fewer number of trials, N, to terminate the algorithm than (SR2). This advantage continues up to about 15 local minima. Beyond 15 local minima, (SR2) needs fewer trials to terminate. The progression of the stopping rules is much more dramatic in Figure 3.4. (SR1) continues its quadratic relation between the number of trials necessary to terminate and the number of unique local minima. Meanwhile,

this relationship is linear for (SR2). For a reasonably sized problem with 200 unique local minima, (SR1) would have to run more than ten times the number of trials than (SR2). Obviously, (SR2) benefits even more over (SR1) as the problems increase in size.

Based on the presented theorems and stopping rules, a stochastic algorithm for solving problem (GP) can now be presented. Using P to represent the number of available processors, K to represent the set of all Karush-Kuhn-Tucker vertices found by the local search procedure, $\omega = |K|$, and N as the number of random search directions, then this procedure is as follows:

Stochastic Algorithm (Ψ, Ω, δ, t, P):

1. Find a feasible vertex $v \in \Omega$.

   Set $v^{(1)} := v^{(2)} := \ldots := v^{(P)} := v$.

   Set $\omega := 0$, $N := 0$, and $K := \{\}$.

For i := 1, 2, ..., P (in parallel) do steps (2) through (12):

2. Pick a random vector $u^{(i)} \in \mathbf{R}^n$.

   Set $j_i := 1$ and $N := N + 1$.

3. Starting from vertex $v^{(i)}$ solve the linear program

$$\min u^{(i)^t} x$$
$$x \in \Omega$$

   to get the vertex z_0^i.

4. Starting from vertex $z_{j_i-1}^{(i)}$ solve the linear program

$$\min \nabla\Psi(z_{j_i-1}^{(i)})^t (x - z_{j_i-1}^{(i)})$$

$$x \in \Omega$$

to get the vertex $z_{j_i}^{(i)}$.

5. If $\Psi(z_{j_i}^{(i)}) \neq \Psi(z_{j_i-1}^{(i)})$ then set $j_i := j_i + 1$ and go to step (4).

6. Set $v^{(i)} := z_{j_i}^{(i)}$.

7. If $K \neq K \cup \{ z_{j_i}^{(i)} \}$ then set $K := K \cup \{ z_{j_i}^{(i)} \}$ and $\omega := \omega + 1$.

8. If $N < \omega + 2$ then go to step (2).

9. If $N \geq \frac{\omega + 1}{\sqrt{1 - t}} + 1$ then go to step (12).

10. If $N < \omega + 3$ then go to step (2).

11. If $N < \frac{\omega^2 + \omega}{\delta} + \omega + 2$ then go to step (2).

12. Stop all processors ($i = 1, 2, \ldots, P$) and set $\Psi^* := \min \{ \Psi(z) : z \in K \}$.

Ideally, the stochastic nature of this algorithm will allow it to solve (GP) in less computing time than previous deterministic methods. Also, the new stopping rule developed from theorem 5 should allow the algorithm to terminate after fewer "global phase" trials relative to the number of unique local minima found. Thus, the hope for this proposed stochastic algorithm is that the implementation on the parallel Cray X-MP and Connection Machine CM-2 supercomputers will accurately solve larger, more difficult problems in less computation time.

Section 4
Parallel Computing and the Stochastic Algorithm

Parallel computing concerns the use of computers to process information utilizing concurrent manipulation of data elements belonging to one or more processors. It is a relatively young field, with the first parallel machines being introduced in the late 1970's. Although the field is relatively new, it is of great importance because it presents a method to significantly increase speed of computation (Quinn 1987). Many different architectures exist for today's parallel computers. The stochastic method was implemented on machines exhibiting two of these architectures. The Cray X-MP EA/464 has a multiple instruction stream, multiple data stream (MIMD) architecture and the Thinking Machines Corporation Connection Machine 2 (CM-2) has a single instruction stream, multiple data stream (SIMD) architecture.

A MIMD architecture implies multiple instruction and multiple data streams and is a more general model of parallel computing than SIMD. An *instruction stream* is the sequence of instructions performed by the machine and a *data stream* is the sequence of data manipulated by the instruction stream (Lazou 1988). MIMD machines usually contain several central processors which operate independently in parallel and asynchronously. The processors execute different instruction streams which can operate on that processor's memory area and/or the area of common memory to which access is shared by all processors.

A SIMD computer uses a single instruction stream but multiple data streams. It achieves multiple data streams from the single instruction stream using the processor array architecture. A *processor array* is a computer implemented as a set

of identical synchronized processing elements capable of simultaneously performing the same operation on different data (Quinn 1987). Although these elements run in parallel, it is possible to mask certain processors to ignore a particular instruction on a certain piece of data, thus allowing for *if...then...else* statements and other control structures used in data manipulation.

It is easy to see that the stochastic algorithm presented in section 3 is designed for use on a MIMD machine. The algorithm allows steps two through twelve to be run on a number of separate processors, P. These processors are executing the same series of instructions asynchronously and independent of one another (with the exception of certain common data manipulations which must be synchronized), thus creating multiple instruction streams. This allows for a significant speedup of execution time, based on the number of independent processors on which these parallel steps can be run.

Since the stochastic algorithm was designed to run on a computer with a MIMD architecture, the question exists as to how to run it in parallel on a machine with SIMD architecture. Since a SIMD computer such as the CM-2 uses a single instruction stream, then when implementing the algorithm put forth in section 3, we must assume that the number of available independent processors is one. Therefore, steps two through twelve are being executed sequentially on only one processor, not in parallel. If not running steps two through twelve in parallel, where does the parallel implementation of the stochastic algorithm occur when running on a SIMD machine?

The answer to the above question lies in the methods applied in the individual steps of the stochastic algorithm. The bulk of the computation occurs in steps three and four of the algorithm. These steps, and step one, involve the solution of a linear

program. A *linear program* is the minimization or maximization of a linear function subject to a finite number of linear constraints, either inequalities and/or equalities (Chvátal 1983). When implementing the stochastic algorithm, the revised simplex method was used for solving the linear programs in these steps. The revised simplex method frequently employs array calculations and manipulation. Often the calculations being executed are equivalent processes performed on several elements of the simplex method's data arrays. An example of this type of procedure would be the multiplication of every element in an array by a constant. This quality lends itself to parallelism on a processor array. The processor array can assign each of its individual processing units to perform the calculations necessary on an element in the data array. Thus, similar calculations on corresponding elements in the data array can be spread over the many calculating elements in the processor array. Using this technique, the stochastic algorithm achieves parallelism over individual array calculations, which are individual steps in the instruction stream, and are necessary to solve the linear programs in steps one, three, and four.

Section 5
Cray X-MP Specifications

The Cray X-MP, first released in 1983, is basically an improvement on the single processor supercomputer Cray-1. It has a faster clock cycle, an improved memory bandwidth, increased maximum memory size and up to four vectored, tightly-coupled processors (Quinn 1987).

The computational section of a Cray X-MP contains up to four CPUs with scalar and vector processing modes. Each CPU has fourteen fully segmented functional units and can access the 64-bit word basic addressable memory unit. The inter-CPU communications and control unit including a 64-bit real time clock handles interprocessor communications (Lazou 1988). These CPUs are not connected, but rather rely on Inter-CPU communication and control hardware as shown in Figure 5.1.

A single CPU in the Cray X-MP has a series of operating registers. There are eight 24-bit address registers which are supported by 64 24-bit rapid access intermediate registers. There are eight 64-bit scalar registers supported by 64 64-bit rapid access intermediate registers. Also, there are eight 64-element vector registers with each element consisting of a 64-bit word. A scalar instruction performs some function obtaining operands from two scalar registers and entering the result in another scalar register. A vector instruction works the same way using the vector registers. A special register, the vector length register, determines the number of operations to be performed by the vector instruction. Other special registers exist for communication between the processors.

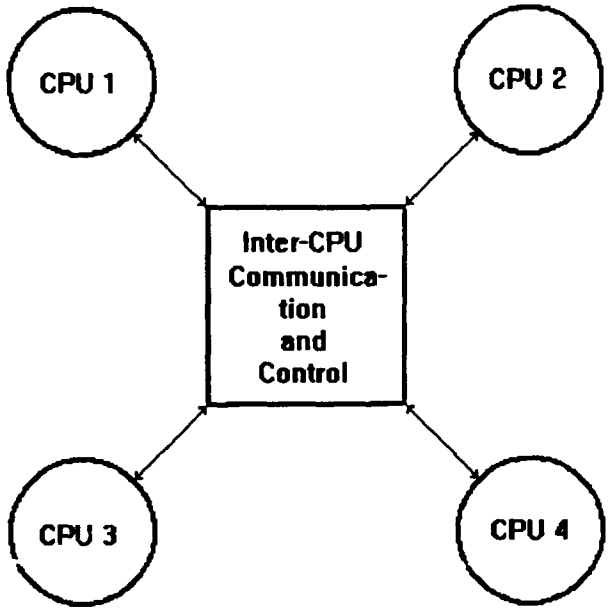


Figure 5.1: Cray X-MP Multiprocessor Configuration

A single Cray X-MP processor also contains four instruction buffers. These buffers each have 128 16-bit parcels to hold the 16 and 32-bit Cray instructions. Each is associated with a base address register which determines whether the current instruction resides in a buffer or needs to be read in from the central memory unit. The central memory unit contains up to 16 million 64-bit words arranged in up to 64 banks. The transfer rate from memory to a vector register is three words per clock period per CPU. The effective transfer rate from memory to a scalar or address register is one word every two clock periods. The transfer rate from memory to the instruction buffers is eight words per clock period.

Each CPU also contains fourteen functional units. The two address functional units provide 24-bit results using addition, subtraction, and multiplication. The four

scalar functional units provide 64-bit results to integer adds, shifts, logical operators and population, parity, and leading zero functions. The five functional units used for vector operations provide 64-bit CPUs on the same functions as the scalar units. There are three floating-point functional units used for both scalar and vector operations. They perform floating additions, multiplications, and reciprocal approximations.

The interconnection section of a Cray X-MP is composed of five clusters of registers for interprocessor communications and synchronization. Each cluster of shared registers consists of eight 24-bit address registers, eight 64-bit scalar registers, and 32 one-bit synchronization registers. Any processor may access a cluster which has been allocated to it in either user or monitor mode. The mode determines interruption and synchronization of the cluster communications. The hardware also includes a built-in system for detecting deadlock within a cluster and a 64-bit real time clock which is shared by all the processors.

The Cray X-MP EA/464 system which was used is located at the University of Minnesota Supercomputer Institute (MSI), Minneapolis, Minnesota. It is a four-processor vector machine with an 8.5 nanosecond clock period. It is capable of 950 Mflops at peak performance and has 64 megawords of common memory which is available to all four processors. It runs the UNICOS 5.1 operating system, a variant of UNIX System V, developed by Cray Research, Inc. (MSC User Guide - MSI Edition 1991).

On the Cray X-MP EA/464 at MSI, the stochastic algorithm was coded in CFT77, a full ANSI FORTRAN 77 standard dialect which is portable across all Cray systems. It contains all features required by the ANSI FORTRAN 77 standard and contains some extensions compatible with the proposed ANSI FORTRAN 90

standard and VAX FORTRAN. Likewise, it is capable of generating vectorized code and supports automatic parallelization. In general, CFT77 offers the most features and produces the fastest code of any of the FORTRAN compilers available on Cray supercomputers.

Section 6
Connection Machine CM-2 Specifications

The Connection Machine Model CM-2 is a SIMD parallel computing system where one processor is associated with each data element. Using this architecture, the CM-2 exploits the computational parallelism that is inherent to data-intensive problems. In the best case, execution times are reduced in proportion to the number of data elements used in parallel computation (Connection Machine Model CM-2 Technical Summary 1989).

The CM-2 is a combined system of both hardware and software. The hardware components include the front-end computers, a parallel processing unit of 64K data processors, and a parallel data I/O system. CM-2 software is based on the operating system used on the front-end computer and most of its instructions are transparent to the user (Hillis 1986).

The front-end computer communicates with the connection machine through a high-speed memory bus. Using this bus, it runs applications, transmitting instructions and data to the CM-2 parallel processing unit. Scalar data is held in the front-end computer which is responsible for any operations on it. The front-end computer also provides the applications development and debugging environment for the system. The operating system runs on this machine and allows application packages and data files to be stored on it. Additionally, any communications links to outside networks are controlled by the front-end. Lastly, the front-end is responsible for maintenance and operating utilities such as allocating and deallocating CM-2 resources, initializing the system, querying of system status, and diagnosing hardware problems.

The CM-2 houses up to 64K parallel processing units, each of which contains an arithmetic-logic unit (ALU) and associated latches, 64K or 256K bits of bit-addressable memory, four 1-bit hardware flag registers, an optional floating point accelerator, a router interface, a NEWS (North, East, West , and South) grid interface, a direct hypercube interface, and an I/O interface. Sixteen of these processors are located on every processor chip and other chips contain different pieces of the data processor hardware. A fully configured parallel processing CM-2 containing 64K data processors is a combination of 4096 processor chips, 2048 floating-point interface chips, 2048 floating-point execution chips and two gigabytes of RAM.

Each of the thousands of CM-2 data processor ALUs executes instructions one bit at a time. There are three input bits and two output bits. On any given bit cycle, the ALU can read two bits from its off-chip memory and write one bit back. Also, it can read any one of the four on-chip flag bits and write to any flag bit, including the one read. Thus, the logic element of the ALU can compute any two boolean functions on three inputs. This simple ALU can perform all Paris (Parallel Instruction Set) operations.

Interprocessor communications is accomplished by the router hardware. All processors can simultaneously send data to or get data from the local memories of other processors. Each CM-2 processor chip contains a router node which serves the 16 data processors on the chip. The router nodes are wired together to form the complete router network. The topology of this network is a boolean n-cube. A fully configured CM-2 uses a 12-cube network to connect the 4,096 processor chips. Thus, each router is connected to and communicates with 12 other routers. Message

traffic travels along this network until it reaches the processor chip containing the destination processor.

The NEWS grid and direct hypercube access are specialized hardware mechanisms which allow for more efficient communications than the general router mechanism. Figure 6.1 shows the NEWS grid of 16 processors on a processor chip:

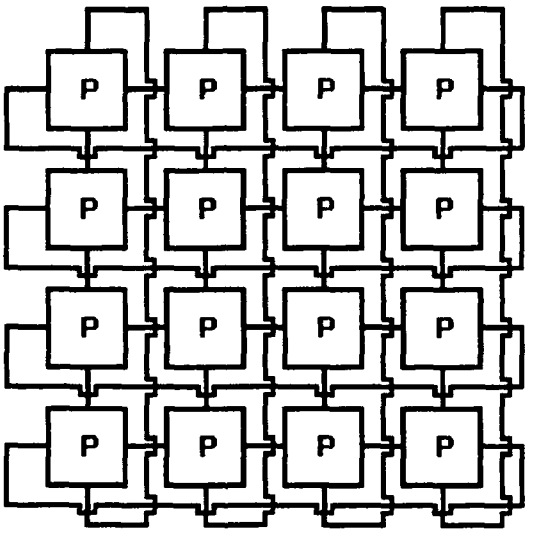


Figure 6.1 : NEWS Grid of 16 Processors on CM-2 Processor Chip

Each parallel data processor is directly connected to its neighboring processor to the North, East, West, and South; a physical four by four grid. This allows faster transfer of data bits between neighboring processors through direct communication. The direct hypercube access hardware allows the 12 hypercube wires entering a node to be directly connected to 12 of the 16 data processors. Thus, the NEWS grid within

a processor chip can be extended through the router wires without utilizing its delivery features, thus speeding up communications.

The CM-2 which was accessed for this research is the 32,768 processor machine located at the Army High Performance Computing Research Center in Minneapolis, Minnesota. It contains 1024 Kbits of memory per processor for a total of 4 Gbytes of memory. The real-time clock runs at 10 Mhertz and its peak performance is rated at 5,000 Mflops with actual applications running in the range of 1,000 to 2,000 Mflops. It has two Sun 4/490 front-ends each with 64 Mbytes of memory running under the UNIX operating system (Connection Machine User Guide 1991). Standard applications run on one of the Sun 4/490s at the approximate speed of 2 Mflops. The high speed memory bus operating between the front end machines and CM-2 parallel processors transmits its communications at a peak performance rate of 4 Mbytes/second. The CM-2 system also has a mass storage "Datavault" which is capable of holding 20 Gbytes of information on 42 drives.

The stochastic algorithm was encoded in CM FORTRAN on the host of the CM-2 system. CM FORTRAN is based on the ANSI FORTRAN 77 and incorporates some extensions proposed in the draft ANSI FORTRAN 8x standard (the precursor to FORTRAN 90). The Connection Machine FORTRAN associates each processor with each element of data in an array. Many other functions and powerful array handling features are implemented in Connection Machine FORTRAN to better utilize the parallel nature of the CM-2.

Section 7
Comparison of Expectations for the Cray X-MP and CM-2 Supercomputers

Section 4 described how the stochastic algorithm would be implemented on a SIMD machine (CM-2) and a MIMD machine (Cray X-MP). Although the peak performance data given in sections 5 and 6 indicates that the CM-2 runs approximately five times faster than the Cray X-MP (5,000 Mflops vs. 950 Mflops), the question remains as to how they will perform when running the stochastic algorithm. There are significant factors other than the peak performance speeds that must be considered when comparing running speed expectations for the two machines.

First, when comparing the peak performance speeds for the different machines, you must consider that they represent timing with all the processors running. The Cray X-MP is running four full processors and given the MIMD implementation detailed in section 3 and 4, we would expect the stochastic algorithm to use all processors for most of its operation. The CM-2 has all of its 32K processors running when peak performance data is timed. Implementing the stochastic algorithm for a SIMD machine requires spreading the data arrays onto these processors. The number of processors on which the data is spread is on the order of the square of the number of linear constraints. Since an extremely large problem would have one hundred constraints, we can expect to be using at most one-third of the processors on the CM-2. Thus, the performance of the stochastic algorithm can already be expected to be at best one-third the peak performance of the CM-2, and significantly less with smaller problem sizes.

The architecture of the CM-2 could also slow down its expected performance. First, scalar operations on the CM-2 system are run on the front end Sun 4/490s. While these hosts are fast, they in no way compare to the processing speed of the Cray X-MP. In fact, for standard applications, the Sun 4/490s are about 500 times slower than the Cray X-MP (2 Mflops vs. 950 Mflops). All scalar operations on the Cray X-MP run at the speed of a single full processor. This processor's peak performance is one-fourth that of the whole machine, (since it has four full processors), and equates to approximately 240 Mflops per second. That is a significant advantage over the scalar operations performed on the Sun 4/490 processors on the front end of the CM-2 system.

Also, since scalar and array calculations occur at two different locations on the CM-2 system, a transfer of data must take place whenever information needs to be exchanged. For example, if an integer counter needed to be incremented, this calculation would be done on the host machine. If the next statement was an array manipulation based on the integer counter, then the value of the integer counter would have to be broadcast across the high speed bus to the parallel processors. Although this example would not significantly slow down the CM-2 performance, repeated communications between the front end machines and the parallel processors, especially when coming between strictly parallel operations, could have a noticeable detrimental effect on the speed of the CM-2.

How the stochastic algorithm performs on the Cray X-MP compared with the CM-2 seems to be a function of problem size and architecture. While a cursory glance at the performance data may indicate the CM-2 should be faster when running the stochastic algorithm, the lack of full processor implementation and architecturally

separate scalar and parallel computations could significantly increase realized computation times.

Section 8
Random Problem Generation Concerns

It is impossible to create by hand enough large linearly constrained concave global minimization problems to fully test the implementation of the stochastic method. Thus, any computational testing must include procedures to create large random problems. It is quite possible that the methods used to create a random problem result in trivial or unrealistic cases. Therefore, steps must be taken to assure that these random problems which are generated are a realistic representation of application problems and are of significant difficulty. Two features of random data which significantly affect formulation of linearly constrained concave global minimization problems are random linear constraint generation and location of the concave function's *global maximum.*

When generating linear constraints, the type of random number generator used to produce the coefficients affects the shape of the created feasible region. Van Dam, Frenk, and Telgen (1983) showed that constraints generated using a uniform random number generator for the linear coefficients tend to cause the angle between the constraints to be the same as the dimension increases (Minkoff 1981).

Figure 8.1 provides a two-dimensional argument for this property. The square represents possible values for the coefficient vector, i.e., normal vector, of a constraint. In the figure, vectors a_1 and a_2 are shown. The lines perpendicular to these vectors are the linear constraints associated with them. They intersect at the shown angle, Θ. If the constraint normals occurred only within the circle then the angle Θ would be uniformly random. This circle of constraint normals can be created by using a normal random number generator for the coefficients. However,

when using a uniform random number generator for the coefficients, the constraint normals can also occur in the shaded region of the square. A greater proportion of the square is shaded as the number of dimensions increases. This causes more constraint normals to accumulate in the shaded region and a greater similarity in the angle between linear constraints. Thus, to generate a uniform distribution of angles of intersection between linear constraints, a normal distribution must be used to generate the constraint coefficients.

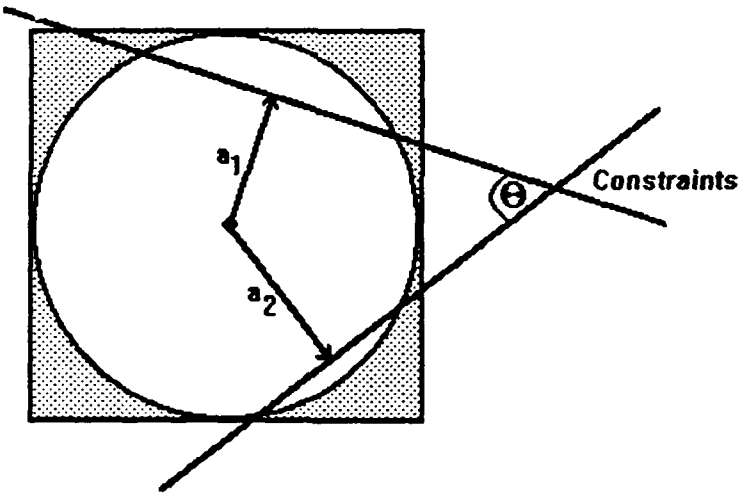


Figure 8.1: Angle Between Random Constraints

The shape of the feasible region is an important concern. In general, application problems of the form (GP) do not have a rounded, symmetrical feasible region. Thus, any random generator which created these constraints would not be testing the algorithm on reasonable problems. Instead, real-world problems often

have sharp, asymmetrical angles between the constraints. Theretore, when generating random problems to test the stochastic algorithm, it was necessary to implement a normal random number generator for constraint coefficients to produce constraints on large random problems that simulated those found on smaller application problems.

The location of the concave function maximum is more a concern of problem difficulty than similarity to applications. If the maximum is located interior to the feasible region, Ω, then the potential for a much more difficult problem exists than if it is exterior to the constraints. Consider Figures 8.2 and 8.3:

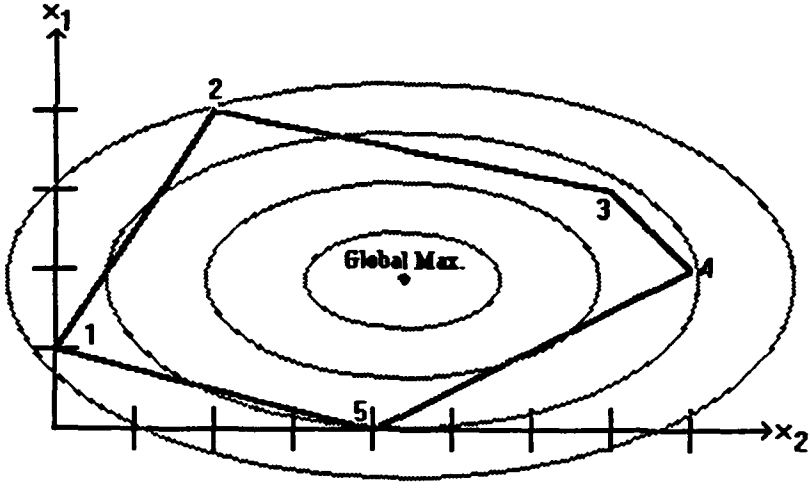


Figure 8.2: Interior Global Maximum

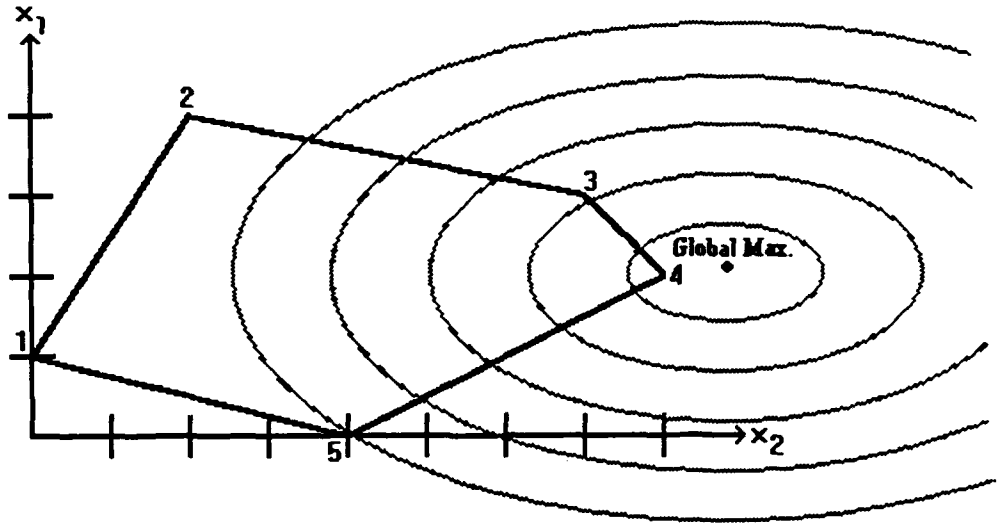


Figure 8.3: Exterior Global Maximum

Note that both figures have the same two-dimensional feasible region. The five vertices of the feasible region are numbered. Both figures also have the same concave function, although it is centered at a different location in each figure. This function is depicted by the labeled global maximum and shaded contour lines. Each contour line represents a specific function value and these values decrease as you move away from the global maximum of the concave function. The only difference between the two figures is the location of the function's global maximum in relation to the feasible region.

Since the stochastic algorithm relies on the number of local minima to determine how many "global phase" trials must be run, the difficulty of the problem depends on the number of vertices which are local minima. In Figure 8.2 the function maximum is within the feasible region. By looking at the function contour

lines, it is obvious that all five of the vertices are local minima. However, in Figure 8.3, which uses the same function and feasible region but the global maximum is located outside the feasible region, fewer vertices could be considered local minima. Vertices three, four, and five are certainly not local minima and vertex two may not be either. Assuming that the stochastic algorithm found all the local minima, that represents a more than fifty percent reduction in the number of trials that need to be run. Thus, in two problems with the same size (five constraints and two dimensions), same shape of the feasible region, and same shape of the concave function, the difficulty was significantly influenced by the location of the global maximum: either interior or exterior to the feasible region.

Given this characteristic of random problem generation, it was necessary to control the location of the function maximum when creating computational test problems. When formulating different sets of standard concave quadratic problems for testing the stochastic algorithm, the option of locating the function maximum was available. This allowed sets of problems with varying difficulty to be created, thus furthering the completeness of algorithm testing.

Section 9
Random Quadratic Function Formulation

The first class of functions tested were concave quadratic functions in separable form. These randomly generated quadratic functions had the following form:

$$\Psi(x) = \sum_{i=1}^{n} \lambda_i (x_i - v_i)^2$$

where v is the unconstrained global maximum of $\Psi(x)$, and $\lambda_i < 0$ for $i = 1, \ldots, n$. This formulation allowed problems to be set up with a known global maximum location, with either the global maximum interior or exterior to the feasible region, Ω, as discussed in section 8.

Another interesting consideration when dealing with this quadratic function formulation is what values to assign to the eigenvalues, $\lambda_1, \ldots, \lambda_n$. As stated, in order for the function to be strictly concave, all eigenvalues must be less than zero. However, the difficulty of the problem is greatly affected by allowing the eigenvalues to vary in their values. More precisely, consider the gradient of the concave quadratic function :

$$\nabla\Psi(x) = 2\sum_{i=1}^{n} \lambda_i (x_i - v_i).$$

Obviously, if the eigenvalues are the same for every dimension, then the gradient is simply twice the constant eigenvalue times the sum of the distances away from the global maximum in every dimension. This is a difficult problem because it

separates the feasible region, Ω, into many regions of attraction of roughly the same size. If, however, you have an eigenvalue whose magnitude is much greater than those in all other dimensions, you usually have a much easier problem. The region of attraction for a vertex which lies distant from the global maximum in the dimension of this large eigenvalue will be much larger than the rest of the local minima. This skewing of the function tends to decrease the number of local minima thus making the problem easier to solve.

Numerous sets of the random quadratic concave function were tested on the Cray X-MP version of the stochastic algorithm. These sets varied in size, location of global maximum, and eigenvalue generation. The results from these computations are presented in section 12.

Section 10
Random Fixed-Charge Function Formulation

The other class of randomly generated concave functions considered was the "fixed-charge" problem. The fixed-charge problem is

$$\min_{x \in \Omega} \; c^t x + \sum_{i \in J(x)} f_i$$

where $\Omega = \{ x \in R^n : Ax \leq b, x \geq 0 \}$ is a nonempty bounded polytope, $f_i \geq 0$ for $i = 1, \ldots, n$, $J(x) = \{ j : x_j > 0 \}$, and c is negative. That is, a "fixed-charge" is incurred for any variable that has a positive value at the solution point x, but no charge is incurred for problem variables at the origin. Figure 10.1 represents a single dimension of the fixed-charge function.

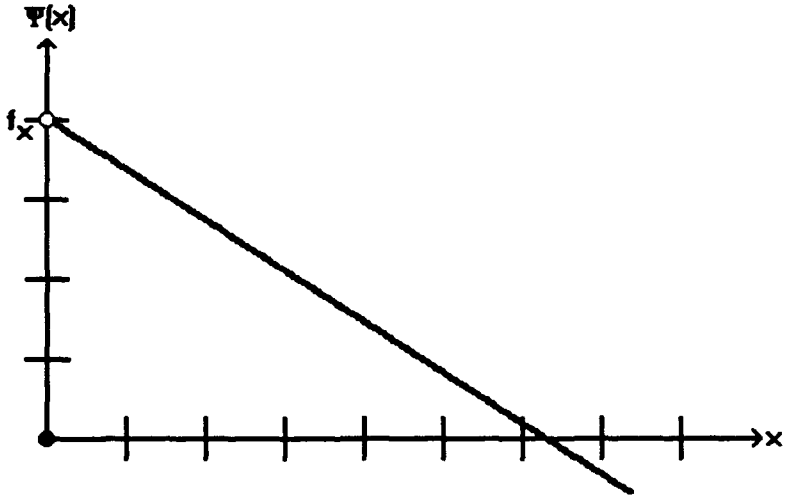


Figure 10.1: One Dimensional Fixed-Charge Function

These problems are very common in practical applications and are usually solved as 0-1 mixed integer linear programs. Hence, it is expected that the number of local minima may grow exponentially with the number of problem variables n (Phillips, Rosen, and van Vliet 1991).

The random "fixed-charge" function was modeled as a two-stage function whose function definition is as follows:

$$\Psi(x) = \sum_{i=1}^{n} \begin{cases} x_i < \varepsilon : 0 \\ x_i \geq \varepsilon : f_i + c_i x_i \end{cases}$$

and whose gradient definition is as follows:

$$\nabla\Psi(x) = \sum_{i=1}^{n} \begin{cases} x_i < \varepsilon : M \\ x_i \geq \varepsilon : c_i \end{cases}$$

where ε is a very small positive constant and M is a very large positive constant.

When randomly generating this problem, steps were taken to assure that f_i was a reasonable value so that a significant "charge" was being imposed for moving a variable away from 0. Several sets of the random "fixed-charge" concave function were tested on the Cray X-MP version of the stochastic algorithm. These sets had various numbers of variables and constraints. The results from these computations are presented in section 13.

Section 11
Results of Test Problems with Known Solutions

Although sections 8, 9, and 10 dealt with random function generation and formulation, the first test problems run on the stochastic algorithms were quadratic programming test problems from Floudas and Pardalos (1990) that had known solutions. The purpose of running these problems first was to assure that the algorithm worked correctly on both the Cray X-MP and the CM-2 before moving on to harder sets of randomly generated test problems. An unexpected result of these tests was the elimination of the CM-2 as a workable platform for larger problems.

The general format of the quadratic test problems in Floudas and Pardalos was:

$$\Psi(x) = c^t x - \tfrac{1}{2} x^t Q x.$$

where c is a constant vector of length n (the number of variables) and Q is a positive definite diagonal matrix of order n. This problem is the same as the quadratic formulation presented in section 9 except for an additional constant term. More precisely, the formulation from section 9 for the concave quadratic function was:

$$\Psi(x) = \sum_{i=1}^{n} \lambda_i (x_i - \nu_i)^2.$$

This same problem written in the vector format of Floudas and Pardalos would be:

$$\Psi(x) = (x-\nu)^t \Lambda (x-\nu),$$

where Λ = diagonal($\lambda_1, \lambda_2, \ldots, \lambda_n$) and $\nu = (\nu_1, \nu_2, \ldots, \nu_n)$. When the terms of this function are expanded, one obtains:

$$\Psi(x) = x^t \Lambda x + \nu^t \Lambda \nu - 2 \nu^t \Lambda x.$$

Thus, we have two terms of the same format, and one additional term. First :

$$x^t \Lambda x = -\tfrac{1}{2} x^t Q x$$

where

$$\Lambda = -\tfrac{1}{2} Q.$$

Also :

$$-2 \nu^t \Lambda x = c^t x$$

where

$$-2 \nu^t \Lambda = c^t.$$

Thus, the only term in our quadratic function formulation not accounted for in the Floudas and Pardalos method is:

$$\nu^t \Lambda \nu.$$

Since ν is a constant vector and Λ is a constant matrix, this term ($d \equiv \nu^t \Lambda \nu$) is also a constant. Thus, the Floudas and Pardalos problems are equivalent to our formulation with the exception of a constant. The following example demonstrates how to change the same problem between the two formulations. Suppose a two variable problem was developed in the method presented in section 9 :

$$\Psi(x) = (x-\nu)^t \Lambda (x-\nu)$$

where

$$\Lambda = diag(-3, -5)$$

and

$$\nu = (2, 1).$$

Using the formulas in the previous derivation, we could find the equivalent Floudas and Pardalos notation by using:

$$\Lambda = -\tfrac{1}{2} Q$$

therefore:

$$Q = diag(6, 10),$$

and

$$-2\, \nu^t \Lambda = c^t$$

so that

$$c^t = (12, 10).$$

The constant term (d) which would have to be added to the Floudas and Pardalos formulation is:

$$d = \nu^t \Lambda \nu$$

therefore:

$$d = (-12, -5).$$

The reason for this difference in formulation is that we wanted to control the position of the global maximum and our function technique easily allows this by assigning it to the constant vector v. In the above example, the global maximum would occur at (x_1=2, x_2=1).

When solving the concave quadratic minimization problem, the stochastic algorithm views the two function formulations identically, even without the constant in the Floudas and Pardalos style. In the above example, the same local minima vertices and global minimum vertex would be found. The only difference would be that without the constant term d in the Floudas and Pardalos formulation, the value of $\Psi(x)$ at each of these vertices would be 17 higher than when using our formulation. Since the d term is a simple constant, it only serves to raise or lower the function over the feasible region, and doesn't affect the stochastic method's solution.

In total, five test problems (2.2, 2.3, 2.4, 2.5, and 2.6) from Floudas and Pardalos (1990) were tested using the stochastic algorithm on both the Cray X-MP and CM-2. Each problem was run between three and five times on each machine to ensure accurate results were obtained. For all problems run on both computers, the stochastic algorithm achieved the same solution as put forth in Pardalos and Floudas. Table 11.1 is a standard output of results from a single problem run. Tables 11.2, 11.3, 11.4, 11.5, and 11.6 summarize the local minima and timing information for the five test problems. Any space in the tables indicates that less than five runs were completed on that machine for that problem. Note that all runs for the stochastic algorithm in this section used the constants $\delta = 0.5$ and $t = 0.99$.

```
***** GLOBAL MINIMUM VERTEX *****

X[ 1] =       1.0000
X[ 2] =       0.0000
X[ 3] =       0.0000
X[ 4] =       1.0000
X[ 5] =       1.0000
X[ 6] =       1.0000
X[ 7] =       0.0000
X[ 8] =       1.0000
X[ 9] =       1.0000
X[10] =       1.0000

***** GLOBAL MINIMUM FUNCTION VALUE *****

PHI =     -39.0000

***** PROBLEM STATISTICS *****

Number of Local Minima             =      67
Number of Trials                   =     676
Trials since Unique Local Minimum  =      19
Global Minimum was Local Minimum   #      34

Most Frequently Found was Local Minimum #      11
                                Found   :      46 Times
                                Value   =    -29.0000

Number of Times Global Minimum was Found =    17
Global Minimum was Found on Trial Number =    80

Average Local Minimum Function Value   =     -13.463
Range on Local Minimum Function Values =      58.305

Total Number of Pivots       =    4971
Total Number of LP Problems  =    1990
Avg. Pivots/LP Problem       =  2.4980

Initial Random Number Seed   =    5649
CPU  Time (Seconds)          =   30.702
Wall Time (Seconds)          =    7.869
```

Table 11.1: Sample Output Results from Single Run of the Stochastic Algorithm on the Cray X-MP for Problem 2.6 from Floudas and Pardalos

	Cray X-MP			CM-2		
	Unique Local Minima	Processor Time (Seconds)	Wall Time (Seconds)	Unique Local Minima	Elapsed Time (Seconds)	Busy Time (Seconds)
Run 1	1	0.382	0.143	1	10.46	2.645
Run 2	1	0.435	0.156	1	7.590	2.245
Run 3	1	0.458	0.219	1	10.90	1.929
Run 4	1	0.537	0.357	1	11.94	2.524
Run 5	1	0.580	0.161	1	9.310	1.796
Averages	1	0.478	0.207	1	10.04	2.228

Table 11.2: Results of Problem 2.2 (m = 7, n = 6)

	Cray X-MP			CM-2		
	Unique Local Minima	Processor Time (Seconds)	Wall Time (Seconds)	Unique Local Minima	Elapsed Time (Seconds)	Busy Time (Seconds)
Run 1	10	6.684	2.516	10	390.3	84.07
Run 2	10	4.769	2.976	10	212.8	79.48
Run 3	10	5.930	2.197	10	149.8	73.26
Run 4	10	6.512	2.454			
Run 5	10	6.159	2.208			
Averages	10	6.011	2.470	10	251.0	78.94

Table 11.3: Results of Problem 2.3 (m = 19, n = 13)

	Cray X-MP			CM-2		
	Unique Local Minima	Processor Time (Seconds)	Wall Time (Seconds)	Unique Local Minima	Elapsed Time (Seconds)	Busy Time (Seconds)
Run 1	2	0.574	0.173	2	7.133	4.271
Run 2	2	0.471	0.167	2	6.269	4.702
Run 3	2	0.492	0.159	2	6.361	4.753
Run 4	2	0.534	0.166	2	6.017	4.620
Run 5	2	0.478	0.166	2	7.386	4.935
Averages	2	0.510	0.166	2	6.633	4.656

Table 11.4: Results of Problem 2.4 (m = 8, n = 6)

	Cray X-MP			CM-2		
	Unique Local Minima	Processor Time (Seconds)	Wall Time (Seconds)	Unique Local Minima	Elapsed Time (Seconds)	Busy Time (Seconds)
Run 1	1	1.128	0.425	1	13.35	6.403
Run 2	1	0.942	0.443	1	8.36	5.134
Run 3	1	0.920	0.379	1	7.89	5.228
Run 4	1	1.144	0.442	1	7.02	4.478
Run 5				1	6.57	4.077
Averages	1	1.034	0.422	1	8.64	5.064

Table 11.5: Results of Problem 2.5 (m = 21, n = 10)

	Cray X-MP			CM-2		
	Unique Local Minima	Processor Time (Seconds)	Wall Time (Seconds)	Unique Local Minima	Elapsed Time (Seconds)	Busy Time (Seconds)
Run 1	67	30.702	7.869	19*	70.7*	28.1*
Run 2	65	28.198	7.940	65	878.1	316.7
Run 3	62	27.953	7.535	63	966.7	369.2
Run 4	61	28.529	7.354			
Run 5	68	30.263	8.491			
Averages	64.6	29.13	7.838	64	922.4	343.0

Table 11.6: Results of Problem 2.6 (m = 15, n = 10)

* = Results not included in averages. See ensuing discussion.

The data from these sample problems lead to a number of conclusions about the stochastic algorithm. The first conclusion that can be drawn from the data is that the stochastic algorithm is effective for solving small problems of the form (GP). This conclusion is based on two factors. First, for every run on every problem, the stochastic algorithm found the proper global minimum vertex as reported by Floudas and Pardalos. Second, there was reasonable stability in the number of local minima the algorithm found for a given problem.

As previously stated, the stochastic algorithm found the correct answer for every problem presented in this section. For every problem Floudas and Pardalos presented a "Best Known Solution" and these were verified by solving the problems on an algorithm that solves (GP) *deterministically*. In fact, on these and other test problems run on the Cray X-MP and CM-2, the stochastic algorithm found the Floudas and Pardalos "Best Known Solution" in every instance except one.

This one exception occurred when the stochastic algorithm obtained a *better* solution than Floudas and Pardalos. This case occurred when running case 5 of problem 2.7. According to their solution, the minimum function value is -4105.2779. The solution arrived at by the stochastic algorithm was a different vertex of the feasible region, with a function value of -4150.4101. Although this difference in function values is relatively small, it is significant that a better vertex was found for the solution of this problem.

How extensively this problem had been tested by Pardalos and Floudas is unknown, but this demonstrates a benefit of the stochastic algorithm over deterministic methods. A deterministic method may be forced to find every possible solution vertex and for problems with many local minima and larger dimensions, the computing time required to provide guaranteed solutions may become infeasible. For large problems, a very stringent program (which requires a high degree of accuracy in all calculations) would be infeasible because it wouldn't be able to test all the potential vertices in a reasonable amount of time. The stochastic algorithm doesn't suffer from this problem because it attempts to randomly cover the feasible region and all the regions of attraction using the "stochastic" global phase. This allows larger problems, such as this one, to be tested with the same accuracy and probability of finding the global minimum.

The fact that the stochastic algorithm achieved at least as good a solution as Floudas and Pardalos for every run of every problem is one indication of it's accuracy. Another indication is the stability in the number of local minima found for different runs of a given problem. Problems 2.2, 2.3, 2.4, and 2.5 all found the same number of local minima on every run for each machine. For example, when solving problem 2.3 (Table 11.3), both the CM-2 and Cray X-MP found ten local minima on

each run. Although this is a small number of local minima, it shows that the stochastic algorithm is sampling a reasonable piece of the feasible region and finding most or all of the local minima.

Problem 2.6 (Table 11.6) is the first to show any fluctuation in the number of local minima found for a single problem. Note that the Cray X-MP found an average of 64.6 local minima with values on a single run ranging from 61 to 68. Although differing number of local minima were found, it is important to note that the variance was very small and that the correct global minimum was found every time. This variance in the number of global minima found on each run is simply a characteristic of the algorithm being stochastic and solving (GP) using probabilistic methods rather than deterministic ones. The fact that there was not a great deal of variance in the number of local minima found and that it correctly obtained the solution to (GP) on every run suggests that the algorithm is an effective method for finding local minima and that appropriate values for the stopping criteria constants are being used.

The only aberration to the relative constancy in the number of local minima found for a given problem occurred in run number one on the CM-2 for problem 2.6 (Table 11.6). While the number of local minima found on the second (65) and third (63) run on the CM-2 correspond to the number found on all the runs for the Cray X-MP (Average = 64.6), the first run only found 19 local minima. Closer inspection of the results showed that on the 19th local minimum, the algorithm got "stuck." It proceeded to find that same local minimum 174 consecutive times until the stopping criteria indicated that 99 percent of the expected volume of the feasible region had been searched. This run demonstrates the potential downfall for a stochastic method. Since a stochastic method uses probability, there will always be that one set of skewed data which gives an abnormal solution while indicating that the problem has

been accurately solved. Although the correct global minimum was found in this case, clearly 99 percent of the feasible region was *not* searched since only 19 local minima were found compared with the average of approximately 64 local minima found by other runs.

This strange result could have been due to any number of problems. The random number generator may have gotten stuck in a repetitive series until the seed was reset in the next run. Perhaps rounding error caused the algorithm to traverse slightly outside the feasible region so that it found the same vertex each time it tried to conform to the linear constraints in the global phase. This problem could have revealed some obscure bug in the coding of the stochastic algorithm. For whatever reason, this data is obviously an anomaly and, while it highlights a potential downfall of the stochastic algorithm, the fact that it greatly deviates from other sample runs was identified and caused it not to be considered in the data averages for the CM-2 on problem 2.6 in Table 11.6.

The other significant conclusions that can be drawn from the test data regard the timed execution of the stochastic algorithm. The timing data of the runs on the problems with known solutions indicate that the coding of the stochastic algorithm on the Cray X-MP was very successful. Conversely, it also shows that using the present code, the CM-2 is not a viable platform for running larger problems.

Table 11.7 summarizes the efficiency for the stochastic algorithm on the test problems from Floudas and Pardalos.

When the test problems were solved on the Cray X-MP using the stochastic algorithm, all four processors were used in parallel (P = 4). This provided the opportunity for speedup of CPU time (the cumulative execution time of all four processors) over wall time (the concurrent execution time of all four processors) to be

	Average Processor Time (Seconds)	Average Wall Time (Seconds)	Efficiency
Problem 2.2	0.478	0.207	0.578
Problem 2.3	6.011	2.470	0.608
Problem 2.4	0.510	0.166	0.768
Problem 2.5	1.034	0.422	0.613
Problem 2.6	29.13	7.838	0.928

Table 11.7: Cray X-MP Average Time Statistics and Speedup for Test Problems

as much as 4.0, but overhead and single processor operations cause it to be less than ideal. The efficiency calculation is a measure of how effectively the method utilizes the parallel processors and is defined as the speedup divided by the number of processors employed. Our data shows that the algorithm is achieving a significant level of efficiency when running even the small test problems. Perhaps most encouraging is the efficiency of 0.928 achieved when solving problem 2.6 . Thus, the solution of this problem used 92.8 percent of the possible CPU time during the time the stochastic algorithm was executing. Ideally, this performance could be a trend that will carry over to large randomly generated problems. Generally, a small problem results in a larger percentage of the CPU time being taken up by interprocessor operations. The hope is that the small size of test problems 2.2, 2.3, 2.4, and 2.5 partially caused the efficiency data to be significantly smaller than will be realized for the larger, randomly generated problems.

One method used to analyze the performance data of the Cray X-MP was to perform a flowtrace analysis on the executable code. Table 11.8, on the following page, is the flowtrace analysis for the five runs of problem 2.3 as shown in Table 11.3.

Routine Name	Multi?	Tot Time	# Calls	Avg Time	Percentage	Accum%
FINDCOL	Y	5.86E+00	7343	7.98E-04	40.16	40.16
GETCOST	Y	24E+00	190918	1.70E-05	22.20	62.36
GETORIG	Y	2.52E+00	101249	2.49E-05	17.29	79.65
DOPHASE2	Y	6.41E-01	1553	4.13E-04	4.39	84.05
NEWTAB	Y	6.35E-01	5790	1.10E-04	4.35	88.40
CALCCOL	Y	5.95E-01	5790	1.03E-04	4.08	92.48
FINDROW	Y	1.55E-01	5790	2.68E-05	1.06	93.54
NEWBASIS	Y	1.50E-01	5790	2.59E-05	1.03	94.57
PARALG	Y	1.43E-01	20	7.15E-03	0.98	95.55
NEWMULT	Y	1.35E-01	1533	8.84E-05	0.93	96.48
ADLOCMIN	Y	1.21E-01	520	2.34E-04	0.83	97.32
SIMPLEX	Y	8.56E-02	1553	5.51E-05	0.59	97.90
RELERR	Y	7.78E-02	4760	1.63E-05	0.53	98.43
PHI	Y	5.04E-02	2073	2.43E-05	0.35	98.78
GETPOSX	Y	3.71E-02	1553	2.39E-05	0.25	99.03
GETPROB	N	3.47E-02	5	6.95E-03	0.24	99.27
PRNTPROB	N	2.46E-02	5	4.93E-03	0.17	99.44
GRADPHI	Y	1.99E-02	1033	1.93E-05	0.14	99.58
RANDVEC	Y	1.35E-02	520	2.60E-05	0.09	99.67
NORM	Y	1.35E-02	520	2.59E-05	0.09	99.76
STOPCRIT	Y	1.20E-02	540	2.21E-05	0.08	99.85
PRNTOBJ	N	7.90E-03	5	1.58E-03	0.05	99.90
PRNTSTAT	N	6.72E-03	5	1.34E-03	0.05	99.95
PRINTMIN	N	5.71E-03	5	1.14E-03	0.04	99.98
$MAIN	N	1.18E-03	1	1.18E-03	0.01	99.99
INITPROB	Y	7.05E-04	20	3.52E-05	0.00	100.00
DOSTATS	N	9.67E-05	5	1.93E-05	0.00	100.00
CHKPROB	N	7.07E-05	5	1.41E-05	0.00	100.00
NEWTAB@44	Y	7.04E-05	3	2.35E-05	0.00	100.00
GETCPU	N	5.72E-05	10	5.72E-06	0.00	100.00
GETWALL	N	4.15E-05	10	4.15E-06	0.00	100.00
INITPAR	N	1.06E-05	5	2.11E-06	0.00	100.00
Totals		1.46E+01	338932			

Table 11.8: Flowtrace Output for the Five Runs of Problem 2.3 on Cray X-MP

The data from this flowtrace illustrate a number of interesting features. First, since all initialization, overhead, and printing routines are not multitasked routines (as indicated by an N under "Multi?"), they were not included in the previous time and efficiency performance data. This means that only multitasked routines were

considered for efficiency calculations. In an attempt to determine when and where four processors are not operating at the same time, we must analyze the algorithm in reference to the flowtrace output.

There are only two procedures considered in the efficiency calculations of the stochastic algorithm which contain statements which could not be run on different processors at the same time. This is because they work on common data known as critical data. If these statements, in the critical section of the code, are allowed to work on the critical data at the same time, improper sequencing of the processors may cause the calculations to be performed incorrectly. Thus, semaphores were employed in the stochastic algorithm to keep the processors from working on the critical data at the same time.

One procedure which contains a critical section is ADLOCMIN. This is critical because it adds data to K, the set of local minima found by all processors. Also, it increments N, ω, and other global data variables (those variables which can be accessed by all processors). If semaphores were not used in ADLOCMIN then two processors could add the same local minimum to the set of unique local minima. While this will not affect the final answer of the stochastic algorithm, it will force more trials to be run since the stopping rules determine when to terminate the algorithm based on the number of unique local minima.

The flowtrace shows that there was a relatively low number of calls to ADLOCMIN and that it only accounted for 0.83 percent of all execution time. However, this percentage does not include waiting time to get into the critical region of the code. The worst case analysis would be that every time a processor was executing ADLOCMIN, the other three processors immediately would be waiting to get into this critical region. Thus, since the reported flowtrace time does not include

waiting time, the worst case total time spent in ADLOCMIN could have actually been four times the total time of 0.121 seconds or 0.484 seconds. When you sum the total time column for the multitasked procedures, but substitute the worst case total time of 0.484 seconds for the measured time of ADLOCMIN, you arrive at a total of 14.869 seconds. Then, assuming that 0.363 seconds of the worst case execution time in ADLOCMIN is time wasted by other processors waiting on the one executing processor, we can estimate a worst case efficiency of:

$$\frac{14.869 - 0.363}{14.869} = 0.976.$$

This indicates that even in the worst case, the procedure ADLOCMIN has little negative effect on the efficiency of the stochastic algorithm. In fact, since this procedure is called only once for every "global phase" trial run, (0.15 percent of all calls in Table 11.8), this worst case estimate is probably much too pessimistic.

The other procedure which manipulates critical data is RANDVEC. This procedure manipulates the random number seed which is used by all processors. However, since it is also run once for every "global phase" trial, it has the same small amount of procedure calls and it's execution time is even less than ADLOCMIN, (0.0135 seconds versus 0.121). Adding the effect of RANDVEC to that of ADLOCMIN generates a worst case efficiency of 0.972.

Two conclusions can be drawn from this data. First, even the worst case analysis demonstrates that the encoding of the stochastic algorithm is very efficient at preventing unnecessary slowdown in the parallel processing. However, the second determination is that there must exist some inherent slowdown when using multiple processors. This is based on the fact that the average efficiency when running

problem 2.3 was only 0.608 on the four processor machine (Table 11.7), much worse than the worst case slowdown caused by waiting inherent in the critical regions of the stochastic algorithm. This slowdown could be caused by communication between processors when operating on global data, competition with processes begun by other users (the most likely reason), and any of the other numerous synchronization features inherent in a multiprocessor machine. Thus, it appears that while efficiency approaching the ideal value of 1.00 is possible given the algorithm's encoding, its actual performance may be the result of factors outside of its control.

The final significant conclusion that can be drawn from the test problems run from Floudas and Pardalos regards the performance of the CM-2. The CM-2 execution times presented in Tables 11.2 - 11.6 are broken down into elapsed time and busy time. The elapsed time is the total amount of execution time for a process, including both front-end run time and CM-2 multiprocessor time. CM busy time is the amount of actual execution time of the multiprocessor CM-2. Thus, the difference between the elapsed time and busy time is the idle time. Idle time includes those clock cycles when the parallel processors are waiting for instructions from the front-end but not those where the parallel processors have received an instructic but are waiting for argument data.

Thus, given the characteristics of the timing categories for the CM-2, we can roughly compare the busy time of the CM-2 to the wall time of the Cray X-MP. These both represent the length of time over which the parallel processors were operating. Data from the test problems in Tables 11.2-11.6 show that the busy time of the CM-2 is between ten and forty times longer than the wall time of the Cray X-MP. This difference seems logical considering the reasons outlined in section 7.

However, this speed difference alone was not sufficient to exclude the CM-2 from consideration for the larger problems.

When CM idle time is factored in, it becomes apparent that our encoding of the stochastic algorithm on the CM-2 is infeasible for larger problems. The total observed CM-2 elapsed time on the presented test problems ranged from 20 to more than 100 times the wall time on the Cray X-MP. It was also clear that as the problem size increased, the performance of the CM-2 relative to the Cray X-MP decreased. Thus, randomly generated problems with hundreds of local minima which run in approximately sixty seconds on the Cray X-MP would take upwards of two hours to run on the CM-2. Likewise, since there is time-sharing of the processors on the CM-2 and the reported elapsed time does not include time you are waiting for other users, the actual observed difference in running time was even greater. This great disparity in the amount of real time it takes to solve a large problem led to the elimination of the CM-2 as a viable platform to test the larger, random problems presented in the next two sections.

The question then occurs as to how to increase the performance of the algorithm on the CM-2. There are two ways to do this. First, the implementation of the stochastic algorithm could have been more CM-2 specific. Second, machine characteristics which affect performance could be improved. Although these possibilities exist for increasing the execution speed of the parallel algorithm, a strong possibility also exists that the significant non-array data processing which is required is the main reason for its slow performance.

When coding the stochastic algorithm, a base program was initially developed in standard FORTRAN and then conversion to specific Cray X-MP and CM-2 constructs took place. The conversion for the Cray X-MP was very smooth. Since

the Cray utilizes coarse-grain parallelism, there were only a few parallel FORTRAN constructs to understand and implement. However, the CM-2 transition was much more difficult. Under the fine-grain parallelism of the CM-2, there were many constructs which had to be understood. In addition, these constructs relied on extensive knowledge of the architecture and workings of the CM-2. Only an expert in CM-2 programming could have coded the stochastic algorithm using the most time-efficient techniques. How much of an effect the inexperienced coding had on the slow execution speed of the stochastic algorithm on the CM-2 is unknown.

A faster, more dedicated front-end system could also reduce the execution time of the stochastic algorithm on the CM-2 system. In fact, since performing the test problem runs detailed in this section on the CM-2 in February of 1992, the Sun 4/490 front-ends have been replaced by higher performance Sun 4/690s. Also, since the front-end machine is a time-sharing system, there is competition among all users for the process execution. A dedicated machine would devote all the front-end processor time to a single process. Again, how much effect a dedicated or improved performance front-end would have on the execution of the stochastic algorithm on the CM-2 is unknown.

In fact, the implementation of either of the two presented possibilities for improved performance of the stochastic algorithm may not produce significantly better results. This is due to the fact that execution of the required data bookkeeping and the subsequent front-end to parallel processor communication makes the stochastic algorithm unsuitable for execution on a Connection Machine system. As previously discussed, all singular data elements and serial operations remain on the front-end. These processes are significantly slower (due to the SUN front-end) than the highly parallel array operations which can be spread across all of the parallel

processors. Also, observe, from the flowtrace output (Table 11.8) previously described, that the three procedures which take up the most time are FINDCOL, GETCOST, and GETORIG. Combined, these procedures account for nearly 80 percent of the program's execution time. Analysis of these procedures indicates that they are all very reliant on serial statements operating on singular data items. In fact, GETCOST, which is the procedure executed most often, accounting for 56.3% of all procedure calls, performs *no* CM-2 parallel operations whatsoever. This indicates that the stochastic algorithm is not very well suited to the CM-2 system of high-speed massively parallel array operations and low speed non-array operations and transmissions. It shows that it is much better suited to a Cray X-MP architecture which performs all operations at high speeds. Until a machine exists that can take the best features of both architectures, by running serial communication and operations at the high speed of the Cray X-MP and performing massively parallel operations when solving linear programs and other fine-grain operations, it appears that the architecture of the Cray X-MP is best for implementation of the stochastic algorithm.

Section 12
Results of Randomly Generated Quadratic Problems

When analyzing the computational results of the stochastic method, one of the biggest concerns is how "hard" a problem it can solve. Theoretically, the stochastic algorithm could solve a problem no matter how difficult. However, memory and time limitations force us to put a reasonable limit on certain characteristics of the problem. Therefore, when solving the randomly generated concave quadratic functions on the Cray X-MP, CPU time is used as a measure of problem difficulty. The CPU time required for a problem is a function of two problem characteristics: the number of local minima and the average number of pivots per trial.

It is readily apparent how the number of local minima affects the CPU time of the stochastic algorithm. Given (SR2), there is a linear relation between the number of trials which need to be run and the number of unique local minima which have been found. Since the stochastic algorithm consists of a repetition of "global phase" trials, CPU time has a direct linear relationship to the number of trials that are run. Thus, CPU time for any problem is linearly related to the number of local minima as well.

While the number of local minima is representative of the total number of trials which must be run, the average number of pivots per trial is a measure of the time it takes to run each trial. In the stochastic algorithm, steps 3 and 4 are two linear programs which are solved repeatedly (see section 3). As previously stated, these linear programs are solved using the revised simplex method. Each iteration of the revised simplex method is considered a pivot, because it moves from one vertex to another along a constraint of the feasible region. The repeated solution of these two

linear programs is the major portion of execution and thus the majority of CPU time. Therefore, the number of pivots the stochastic algorithm takes per trial is a measure of the CPU time per trial.

Since CPU time is dependent on the number of local minima and the number of pivots per trial, the next step is to explore what characteristics of the randomly generated concave quadratic problem affect these statistics.

As discussed in section 8, the location of the concave quadratic function's global maximum is expected to affect the number of local minima. If the maximum is interior to the feasible region, then, on average, more local minima are expected. If the maximum is unconstrained, then fewer local minima are expected on average.

Section 8 also contained an explanation of how the generation of the eigenvalues of the concave quadratic function affects the number of local minima. It is expected that if the eigenvalues are equal then there will be more local minima per problem. Likewise, if the eigenvalues are random, then, on average, fewer local minima are expected.

Table 12.1 summarizes the analysis of the effect of both the location of the global maximum and the mode of eigenvalue generation on the number of local minima. As the legend shows, the problems were categorized into four different types based on the two characteristics to be studied. Relative amounts for the average number of local minima are given for each type of problem in each of six different sizes. More precisely, for each size category the problem type with the least number of local minima is normalized to a value of 1.00. The number for every other type within that size is simply the average number of local minima for that type divided by the average number of local minima for the type whose value is 1.00. Thus a relative average is given for each problem type within a size category.

Legend for Table 12.1
A = Random Eigenvalues, Interior Global Maximum
B = Random Eigenvalues, Unconstrained Global Maximum
C = Equal Eigenvalues, Interior Global Maximum
D = Equal Eigenvalues, Unconstrained Global Maximum

	n = 20	n = 40	n = 60
m = 15	A = 1.25 B = 1.00 C = 2.28 D = 2.43	A = 1.20 B = 1.00 C = 1.67 D = 1.42	A = 1.08 B = 1.00 C = 1.37 D = 1.35
m = 25	A = 1.02 B = 1.00 C = 1.44 D = 1.64	A = 1.00 B = 1.05 C = 1.23 D = 1.37	A = 1.00 B = 1.21 C = 1.55 D = 1.54

Table Averages
A = 1.09
B = 1.09
C = 1.62
D = 1.60

Table 12.1: Relative Average Amounts of Local Minima Generated per Problem

The results from Table 12.1 are surprising. The difference in relative average amounts of local minima generated per problem is very small between problem types A and B. This difference is nearly zero between types C and D as well. Since type A differs from type B in the location of the function maximum and type C differs from type D in the location of the function maximum, it can be concluded that the location of the function maximum has *little* effect on the number of local minima! However, there is a jump in the relative average number of local minima between the A and B types, which have random eigenvalue generation, and the C and D type problems,

which have equal eigenvalues. Thus, the method of eigenvalue generation appears to have a significant effect on the average number of local minima. Tables 12.2 and 12.3 provide more evidence for these conclusions.

Eigenvalues	n = 20 m = 15	n = 40 m = 15	n = 60 m = 15	n = 20 m = 25	n = 40 m = 25	n = 60 m = 25	Avg.
Equal	59.69	10.80	6.91	34.72	21.22	5.07	23.07
Random	46.57	21.39	19.32	93.68	23.30	41.48	40.96

Table 12.2: Standard Deviation of the Average Number of Local Minima as a Percentage of the Average Number of Local Minima for Problems with Interior Global Maximum

Global Maximum	n = 20 m = 15	n = 40 m = 15	n = 60 m = 15	n = 20 m = 25	n = 40 m = 25	n = 60 m =25	Avg.
Interior	59.69	10.80	6.91	34.72	21.22	5.07	23.07
Unconstrained	56.82	22.24	1.53	63.79	6.04	6.64	26.18

Table 12.3: Standard Deviation of the Average Number of Local Minima as a Percentage of the Average Number of Local Minima for Problems with Equal Eigenvalues

Tables 12.2 and 12.3 show the standard deviation of the average number of local minima as a percentage of the number of local minima. For example if the average number of local minima found was 150 and the standard deviation was 30 then the table would read 20, since 30 is 20 percent of 150.

Table 12.2 demonstrates that different types of eigenvalues (equal vs. random) cause a great variation in the standard deviation. Specifically, random eigenvalues create problems which have a much wider range in the number of local minima generated than problems which have equal eigenvalues. This is because a problem with random eigenvalues could have approximately the average number of local minima of a problem with equal eigenvalues if all the randomly generated eigenvalues are roughly equivalent. Likewise, it could have much fewer local minima as a result of a great range of randomly generated eigenvalues. These possibilities combine for a standard deviation for random eigenvalue problems that is relatively much larger than the standard deviation of equal eigenvalue problems. Thus, Table 12.2 gives further evidence that the types of eigenvalues for the problem have a significant effect on the number of local minima.

If the location of the function global maximum had a significant effect on the number of local minima, then Table 12.3 would have similar trends as Table 12.2. We would expect that an unconstrained maximum would have a greater standard deviation since it could have an interior global maximum or an exterior one. However, Table 12.3 demonstrates that the unconstrained global maximum has a standard deviation which is only slightly larger than the interior global maximum. The fact that the standard deviation changes very little based on the location of the global maximum, coupled with the fact from Table 12.1 that the location of the global maximum has little effect on the number of local minima, demonstrates that the location of the global maximum has virtually no effect on the number of local minima for a randomly generated concave quadratic problem. Therefore, in further analysis of the number of local minima generated, problems with different locations of the global maximum will be grouped together.

The other characteristic of the randomly generated quadratic functions which has an effect on the number of local minima is the size of the problem. Tables 12.4, 12.5, and 12.6 summarize this relationship.

	n = 20	n = 40	n = 60
m = 15	60.14	269.0	370.6
m = 25	62.50	311.0	366.9

Table 12.4: Average Number of Local Minima per Problem with Random Eigenvalues

	n = 20	n = 40	n = 60
m = 15	125.6	392.0	484.9
m = 25	95.20	394.2	512.4

Table 12.5: Average Number of Local Minima per Problem with Equal Eigenvalues

	n = 20	n = 40	n = 60
m = 15	92.89	330.5	427.5
m = 25	78.85	352.6	439.6

Table 12.6: Average Number of Local Minima per Problem for all Problem Types

From these tables, it is obvious that the number of local minima is a function of the size of the problem. As the number of variables (n) increases, independent of other characteristics of the problem, the number of local minima increase. Also, the

number of constraints (m) has a similar effect on the number of local minima, although not nearly as definitive or pronounced. The basic logic behind this relationship was given in section 3 where the formula relating the maximum number of vertices to the number of variables and the number of constraints was presented. The average number of vertices increases as either dimension or the number of constraints increases. When more vertices exist, then there are more potential candidates for local minima and thus, on average, more local minima per problem. Therefore, increasing size, either in the number of variables or number of constraints, results, on average, in more local minima per problem.

The number of pivots per trial is another characteristic of a randomly generated concave quadratic problem that has been shown to have a linear relationship with CPU time. We will again explore how the location of the global maximum and the method of eigenvalue generation affect the number of pivots per trial in light of how they affect the number of local minima.

How these characteristics of the problem affect the number of pivots per trial is a function of the number of local minima. Specifically, assuming a fixed problem size, if there are more local minima, then the number of pivots to find a local minimum is *less*. This decrease in the number of pivots for the local phase decreases the overall average number of pivots per trial. Therefore, given the relatively insignificant effect of the location of the global maximum on the number of local minima, the location of the global maximum is expected to have a correspondingly small impact on the number of pivots per trial. Similarly, the types of eigenvalues generated can be expected to have a much greater effect on the number of pivots per trial. Table 12.7 employs the same technique as Table 12.1 to summarize the relative

impact that the location of the global maximum and the types of eigenvalues have on the average number of pivots per trial.

Legend for Table 12.7
A = Random Eigenvalues, Interior Global Maximum
B = Random Eigenvalues, Unconstrained Global Maximum
C = Equal Eigenvalues, Interior Global Maximum
D = Equal Eigenvalues, Unconstrained Global Maximum

	n = 20	**n = 40**	**n = 60**
m = 15	A = 1.09 B = 1.17 C = 1.00 D = 1.00	A = 1.14 B = 1.13 C = 1.00 D = 1.06	A = 1.13 B = 1.18 C = 1.01 D = 1.00
m = 25	A = 1.14 B = 1.19 C = 1.03 D = 1.00	A = 1.14 B = 1.15 C = 1.00 D = 1.06	A = 1.19 B = 1.14 C = 1.00 D = 1.03

Table Averages
A = 1.14
B = 1.16
C = 1.01
D = 1.03

Table 12.7: Relative Average Amounts of $\frac{\text{\# of Pivots}}{\text{Trial}}$

The results of Table 12.7 reiterate the results of Table 12.1. Since the location of the global maximum has almost no effect on the number of local minima, it also has a very small effect on the average number of pivots per trial. This is

evidenced by both the average A and B, and the average C and D values being equal. Therefore, since it has been shown that the location of the global maximum has almost no effect on any of the statistics which affect execution time of the stochastic algorithm, it will no longer be considered as a distinction for a randomly generated concave quadratic problem.

The averages from Table 12.7 also reinforce the impact of eigenvalue type on the number of pivots per trial. However, it appears that this effect is relatively much smaller than the effect on the number of local minima. This may be due to the fact that the number of local minima only influences the local phase portion of the number of pivots per trial calculation, not the global phase.

Each trial of the stochastic algorithm runs the global phase linear program to find a random vertex of the feasible region. This linear program has a number of pivots based on the number of vertices encountered when moving to that random vertex. Since the number of problem variables and the number of constraints affect the number of vertices on the feasible region, it is obvious that they also affect the number of pivots in the global phase. Since the number of global phase pivots is influenced, size affects the overall average number of pivots per trial. Table 12.8 summarizes the impact of size on the number of pivots per trial.

Eigenvalues	n = 20 m = 15	n = 40 m = 15	n = 60 m = 15	n = 20 m = 25	n = 40 m = 25	n = 60 m = 25
Random	26.45	40.91	50.03	41.84	76.03	98.55
Equal	23.51	37.21	43.42	36.37	68.49	85.95

Table 12.8: Average $\frac{\text{\# of Pivots}}{\text{Trial}}$ per Problem

Table 12.8 confirms the expectation that as size increases, the number of pivots per trial also rises. The effect is significant and this large impact may be the result of a less obvious relationship between problem size and the number of pivots in the local phase. More precisely, a case could be made for the theory that if the number of vertices increases while the number of local minima remains stable, it will also take more pivots per local phase portion of the trial to solve for a local minimum. Thus, the large effect of problem size most likely results from an overall increase in vertices, which increases the pivots necessary to find the global random vertex and the local minimum vertex.

Given that we have a linear relationship between CPU time and the number of local minima and another linear relationship between CPU time and the number of pivots per trial, the question exists as to whether CPU time is a function of the sum or the product of these characteristics. Obviously, since the number of local minima is a measure of the number of trials, and the number of pivots per trial is a measure of the execution time per trial, it is expected that CPU time is linearly related to the product of the number of local minima and the average number of pivots per trial. Table 12.9 summarizes the reliability of the linear regression calculations relating CPU time to the number of local minima, the number of pivots per trial, the sum of these two statistics, and the product of these two statistics.

	# of Local Minima	# of Pivots / Trial	Sum	Product
R-squared	0.639	0.585	0.704	0.960

Table 12.9: Statistical Reliability of Linear Models for CPU Time

The R-squared term is a standard statistical representation of the accuracy of a linear model to the actual points. A value of 0.00 represents no correspondence between the independent and dependent variables. Likewise, a value of 1.00 represents complete correspondence. Thus, the higher the R-squared term is, the more accurate the linear modeling. As expected, the most accurate model is that which linearly relates CPU time to the product of the number of local minima and the number of pivots per trial. The complete linear regression model for CPU time for randomly generated concave functions is:

$$\text{CPU Time} = 0.010466 * (\text{Number of Local Minima}) * \left\{\frac{\text{Number of Pivots}}{\text{Trial}}\right\}.$$

An interesting result of this function is the cumulative effect that the types of eigenvalues has on the CPU time of the stochastic algorithm. While increased size increases both components of the CPU time function, different types of eigenvalues move these components in opposite directions. More precisely, an equal eigenvalue problem is expected to have more local minima than a random eigenvalue problem of the same size. From Table 12.1, it is expected that an equal eigenvalue problem will have about 1.478 (1.61/1.09) times more local minima than a random eigenvalue problem. At the same time, an equal eigenvalue problem will have fewer pivots per trial. From Table 12.7, the equal eigenvalue problem will have about 0.887 (1.02/1.15) times fewer pivots per trial than a random eigenvalue problem. Thus, an equal eigenvalue problem is expected on average to take about 1.31 (1.478*0.887) times the CPU time of a same sized random eigenvalue problem. Table 12.10 summarizes the computational results of the comparison of CPU times for various sized problems.

Eigenvalues	n = 20 m = 15	n = 40 m = 15	n = 60 m = 15	n = 20 m = 25	n = 40 m = 25	n = 60 m = 25
Random	6.504	84.17	198.1	14.31	199.6	411.1
Equal	14.31	113.8	229.4	19.37	232.4	509.4
Factor	2.200	1.352	1.158	1.354	1.164	1.239

Table 12.10: Average CPU Time per Problem and Increase Factor of Equal Eigenvalues over Random Eigenvalues

Table 12.10 shows that the computational results support the formula estimation of the effect of eigenvalue generation on CPU time. Table 12.10 has an average multiplication "factor" of 1.411, a difference of less than eight percent from the value obtained from the linear regression formula and previous statistical results. This result supports the statistical analysis and linear regression formula relating CPU time to the product of the number of local minima and the number of pivots per trial.

Thus, through extensive statistical analysis, a formula which can accurately estimate the CPU time of a randomly generated concave quadratic problem solved by the stochastic algorithm on the Cray X-MP has been approximated. Likewise, observations have demonstrated to what degree certain problem characteristics influence the variables which affect CPU time, and even eliminated the location of the global maximum as an influence on CPU time. Through this extensive investigation of the CPU time of the stochastic algorithm, an accurate estimation of the "difficulty" of the randomly generated concave quadratic function problems has been generated.

Solution of random concave quadratic problems provides information on two other performance considerations of the stochastic algorithm: CPU efficiency and stopping criteria effectiveness.

In section 11, a trend of increasing CPU efficiency as the test problems got more difficult was observed. Ideally, that trend would have carried over to the solution of random problems and their much larger sizes would create very high efficiency in the use of the four processors of the Cray X-MP. However, it was observed that there was no correlation between problem difficulty and efficiency of the stochastic algorithm. The efficiency statistic was seemingly random with an average value of about 0.65, and a range of 0.20 to 0.95. Thus, the trend of increasing efficiency with greater number of local minima was not substantiated.

The primary reason for the random nature of CPU efficiency is that the random concave quadratic problems were not run on a dedicated machine. The execution of the stochastic algorithm was competing for time and processors with all of the other programs executing at the same time on the Cray X-MP. Any particular run could have received less than the requested four processors, and thus greatly decreased the efficiency calculation which assumes all four processors are allocated to the stochastic algorithm. To perform a more accurate test of the efficiency of the stochastic algorithm, time on a Cray X-MP devoted to running only that code would be necessary.

The performance of the stopping criteria is the final significant result obtained from testing the stochastic algorithm on randomly generated quadratic problems. Since these problems are randomly generated, there can be no guarantee that the stochastic algorithm arrived at the correct global minimum. However, from the accurate results presented in section 11 (where every minimum obtained to test problems was at least as good as the best known solution), and the data in Table 12.11, we see that stopping criteria of the stochastic algorithm were sufficiently strict. In Table 12.11, N^* represents the first trial number on which the global minimum

vertex was found and ω^* represents what unique local minima number was the global minimum.

Table 12.11 shows the relative speed of finding the global solution vertex for a randomly generated concave quadratic problem. Note that on average, the solution is found after only 1.17 percent of the trials have been run and 6.25 percent of the global minima have been found. More precisely, on average, the stochastic algorithm runs 98.8 percent of its trials and finds 93.75 percent of the other local minima *after* the global minimum vertex has been found! Given these values, we can be reasonably certain that the stopping criteria are sufficiently strict and that a correct solution vertex is being found by the stochastic algorithm.

			Avg. Case		**Worst Case**	
Eigenvalues	**n**	**m**	$\frac{N^*}{N}$	$\frac{\omega^*}{\omega}$	$\frac{N^*}{N}$	$\frac{\omega^*}{\omega}$
Random	20	15	.0162	.0971	.0390	.2791
Random	40	15	.0056	.0376	.0160	.1054
Random	60	15	.0064	.0415	.0196	.1160
Random	20	25	.0194	.0953	.0652	.2688
Random	40	25	.0026	.0155	.0046	.0354
Random	60	25	.0081	.0609	.0173	.1122
Equal	20	15	.0153	.0755	.0503	.2379
Equal	40	15	.0043	.0314	.0094	.0667
Equal	60	15	.0067	.0484	.0228	.1667
Equal	20	25	.0124	.0667	.0361	.2857
Equal	40	25	.0321	.1074	.2711	.6256
Equal	60	25	.0108	.0722	.0361	.2364
		Average	.0117	.0625	.0490	.2113
		Worst	.0321	.1074	.2711	.6256

Table 12.11: Relative Speed of Finding Global Minimum Vertex

The worst case data gives more information about the adequacy of the stopping criteria. For Table 12.11, the worst case occurs when the global minimum was found latest in the number of trials and number of local minima, thus putting the correct solution in jeopardy if a decrease in the strictness of the stopping rules is implemented. Considering that all of these problems stopped after satisfying (SR2) and that a value of $t = 0.99$ was used, a decrease in this expected volume of the feasible region to be searched could be warranted. In well over 100 test problems, the latest that the stochastic algorithm ever found the global minimum vertex was after 27 percent of the trials had been run. Thus, at least 73 percent of the trials for any problem were run to find local minima other than the global minimum and to satisfy the stopping criteria. The average of the worst case scenario across various sized problems is even more dramatic. Given any particular size problem, it is expected that, even in the worst case, where the global minimum is found late in the stochastic algorithm's execution, at least 95 percent of trials would be run after the global minimum was found. A decrease in t would decrease the number of unnecessary extra trials which are run by the stochastic algorithm. However, a decrease in t could have negative implications as well.

A decrease in t would decrease the strictness of (SR2), allowing the stochastic algorithm to terminate after fewer trials. While the data in Table 12.11 indicates that this decrease may not affect the final solution of the stochastic algorithm, it does not present the whole picture. A decrease in the expected volume of the feasible region to search could result in premature termination if the frequency of unique local minima location of the stochastic algorithm varies greatly. For example, if relatively few of the unique local minima are found in early trials, then a decreased t constant could cut off the search and declare that the global minimum

solution has been found. In fact, this would not represent the complete solution of the problem. Therefore, while it is apparent that a value of $t = 0.99$ for (SR2) is causing redundant searches to be performed, the accuracy it achieves justifies maintaining it at this level unless problem size dictates that less stringent stopping criteria be implemented.

Section 13

Results of Randomly Generated Fixed-Charge Problems

The formulation of randomly generated fixed-charge problems was given in section 10. Analysis of computational data from randomly generated fixed-charge functions solved using the stochastic algorithm on the Cray X-MP revealed no new information regarding the performance of the stochastic algorithm. It did, however, demonstrate the inherent difficulty of this type of problem.

Because of the formulation method, size is the only characteristic which affects the difficulty of the problem. Since the problem is not quadratic, there are no eigenvalues to generate. Likewise, the global maximum is at a fixed point just inside of the origin. Thus, the only important problem statistics are contained in Tables 13.1 and 13.2. These tables summarize the effect that size has on the number of local minima and CPU time for a fixed-charge problem.

	n = 10	**n = 20**
m = 10	64.00	746.6
m = 20	133.4	670.2

Table 13.1: Average Number of Local Minima per Problem

	n = 10	**n = 20**
m = 10	1.634	49.91
m = 20	6.340	90.45

Table 13.2: Average CPU Time per Problem

When the results from the fixed-charge problem are compared with similar results from the randomly generated concave quadratic function (Tables 12.6 and 12.10), it is obvious that the fixed-charge problem is much more difficult. The fixed-charge function generates a much greater number of local minima and, correspondingly, the CPU times were much larger than same-sized quadratic problems. In fact, fixed-charge problems of modest size proved intractable on the Cray X-MP.

To save memory and CPU time on the Cray X-MP, the number of unique local minima was limited to 1000 for any single run of the stochastic algorithm. After reaching 1000, the algorithm stopped and any obtained results were reported. However, this information would not represent a complete solution because neither of the stopping rules had been met. When solving randomly generated fixed-charge problems, the stochastic algorithm quickly obtained 1000 local minima and terminated for any problems larger than 10 constraints and 30 variables. Likewise, in problems of 25 constraints and 50 variables, which were difficult but still quite reasonable for quadratic functions, the stochastic algorithm discovered 1000 local minima in, on average, 1250 trials. Obviously, since the stochastic algorithm discovers such a high percentage of local minima per trial, many more local minima exist which have not yet been found. These statistics indicate that the randomly generated fixed-charge problem is inherently much more difficult than a randomly generated concave quadratic problem of the same size.

In addition to the randomly generated fixed-charge problems, two fixed-charge application problems with known solutions were tested on the Cray X-MP. The "Ghandi Cloth Company" and "J.C. Nickles Lockbox" problems from Winston (1987) were solved. Despite their relatively large size ($n = 20$, $m = 40$ for the "J.C.

Nickles" problem), each was relatively simple, containing only one local minimum (the solution vertex), and the stochastic algorithm accurately solved both in an insignificant amount of CPU time (0.040 seconds and 0.130 seconds, respectively). Other than substantiating the supposition that the stochastic algorithm is a reasonable technique for solving extremely small application problems, the solution of these example application problems provided no additional information regarding the implementation of the stochastic algorithm. It does show, however, that either these test problems are much easier than a standard fixed charge problem, or that the randomly generated fixed-charge problems we tested are much more difficult than standard application problems.

Section 14
Results of a Sample Application Problem

One potential application of the stochastic algorithm is in the design of VLSI chips. A VLSI chip is made up of components which need to be electrically wired. The objective is to minimize the distance between the components which need to be connected, in an effort to minimize the overall size of the chip. The constraints can be formulated to account for orientation of the components, enforce particular area requirements, and prevent overlapping of components. This is a very large problem and, as such, is extremely difficult to solve.

In an effort to simulate this type of problem on a much smaller scale, the two-dimensional bin-packing problem was considered. The two-dimensional bin-packing problem is: Given a collection of rectangles, and a bin with a fixed width and an unbounded height, pack the rectangles into the bin so that no two rectangles overlap and so that the height to which the bin is filled is as small as possible (Coffman, Garey, Johnson, and Tarjan 1980). For this problem, it is assumed that the rectangles are oriented, each having a specific horizontal and vertical side. Figure 14.1 is a typical rectangle (B_i) used in the bin-packing problem where (x_i, y_i) represents the location of the center of the block in the bin.

If W equals the width of the bin, and N equals the number of blocks, then the bin constraints (BC) are:

$$x_i - \tfrac{1}{2}L_i \geq 0$$

$$x_i + \tfrac{1}{2}L_i \leq W$$

$$y_i - \tfrac{1}{2}H_i \geq 0$$

$$\forall\, i = 1,\ldots,N.$$

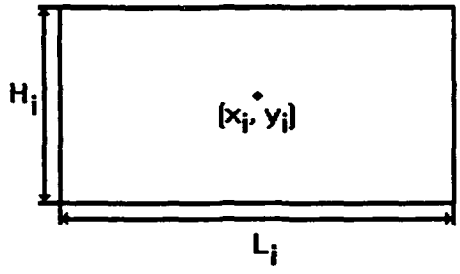


Figure 14.1: Bin-Packing Rectangle (B_i)

The bin constraints force all rectangles to lie within the edges of the bin, but do not prevent overlapping of rectangles. In order to prevent the rectangles from overlapping, at least one of the following must hold for each pair of i and j ($i \neq j$):

$$x_i + \tfrac{1}{2}L_i \leq x_j - \tfrac{1}{2}L_j \qquad (B_i \text{ to the left of } B_j)$$

$$x_i - \tfrac{1}{2}L_i \geq x_j + \tfrac{1}{2}L_j \qquad (B_i \text{ to the right of } B_j)$$

$$y_i + \tfrac{1}{2}H_i \leq y_j - \tfrac{1}{2}H_j \qquad (B_i \text{ below } B_j)$$

$$y_i - \tfrac{1}{2}H_i \geq y_j + \tfrac{1}{2}H_j \qquad (B_i \text{ above } B_j)$$

Note that two of these non-overlapping constraints *may* be satisfied, but at least one *must* be satisfied. L and H are now defined as:

$$L = \sum_{i=1}^{N} L_i$$

and

$$H = \sum_{i=1}^{N} H_i .$$

Clearly, it can be assumed that:

$$|x_i - x_j| \le L$$

and

$$|y_i - y_j| \le H$$

$$\forall\ i, j = 1,\ldots,N.$$

Therefore, the 0-1 integer variables x_{ij} and y_{ij} $\{\forall\ i = 1,\ldots,(N-1),\ \forall\ j = (i+1),\ldots,N\}$ are introduced and the non-overlapping constraints (NOC) for any two blocks B_i and B_j $(i < j)$ can be represented by:

$$x_i + \tfrac{1}{2}L_i \le x_j - \tfrac{1}{2}L_j + (x_{ij} + y_{ij})\,L$$

$$x_i - \tfrac{1}{2}L_i \ge x_j + \tfrac{1}{2}L_j - (1 + x_{ij} - y_{ij})\,L$$

$$y_i + \tfrac{1}{2}H_i \le y_j - \tfrac{1}{2}H_j + (1 - x_{ij} + y_{ij})\,H$$

$$y_i - \tfrac{1}{2}H_i \ge y_j + \tfrac{1}{2}H_j - (2 - x_{ij} - y_{ij})\,H$$

Notice that for:

$(x_{ij}, y_{ij}) = (0, 0)$, B_i is to the left of B_j ;

$(x_{ij}, y_{ij}) = (0, 1)$, B_i is to the right of B_j ;

$(x_{ij}, y_{ij}) = (1, 0)$, B_i is below B_j ;

$(x_{ij}, y_{ij}) = (1, 1)$, B_i is above B_j.

Since we want all $x_{ij}, y_{ij} \in \{0, 1\}$, we must use limiting constraints (LC):

$$0 \le x_{ij} \le 1$$

and

$$0 \le y_{ij} \le 1$$

$$\forall\ i\ j\ (i < j).$$

To account for the overall height of the bin, we introduce another variable y_0, which is subject to the following height constraints (HC):

$$y_0 \geq y_i + \tfrac{1}{2}H_i$$
$$\forall i = 1,\ldots,N$$

Thus, we want to minimize the overall height of the bin, y_0, subject to all the previous constraints. However, (LC) forced the x_{ij} and y_{ij} variables to be *between* 0 and 1. To get them to be *either* 0 or 1 at the global minimum vertex, we use the following "penalty" function formulation:

$$\min y_0 + \mu \sum_{\forall ij (i<j)} x_{ij}(1 - x_{ij}) + \mu \sum_{\forall ij (i<j)} y_{ij}(1 - y_{ij})$$

subject to: (BC)

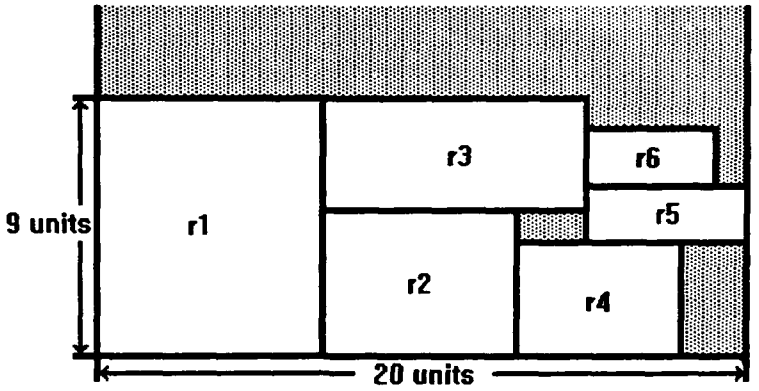
(NOC)

(LC)

(HC)

where: μ is *very large* ($\mu = H$ is sufficient).

Figure 14.2 shows the two-dimensional bin-packing problem from the literature that was tested on the Cray X-MP (Coffman, Garey, Johnson, and Tarjan 1980). Although this appears to be a very small problem (N = 6) which could be solved by inspection, it actually requires a large number of variables and constraints. For N = 6, there are 42 variables (6 x_i, 6 y_i, 15 x_{ij}, and 15 y_{ij}) and 114 constraints (18 (BC), 60 (NOC), 30 (LC), and 6 (HC)). Thus, it was expected that this problem would be reasonably difficult to solve.



r1 = 7 by 9 rectangle
r2 = 6 by 5 rectangle
r3 = 8 by 4 rectangle
r4 = 5 by 4 rectangle
r5 = 5 by 2 rectangle
r6 = 4 by 2 rectangle

Figure 14.2: Two-Dimensional Bin-Packing Problem Tested on the Cray X-MP.

In fact, the problem displayed in Figure 14.2 proved extremely difficult to solve on the Cray X-MP. In order to find the proper global minimum, we were forced to raise t to 0.9999. Once this was done, the correct solution was found, but only after a great deal of computation. The stochastic algorithm found 1071 local minima, ran over 100,000 "global phase" trials, and took 3.31 *hours* of CPU time. Some of the reasons which explain why the bin-packing problem is so difficult to solve can be seen in the graphical representation of the global minimum solution (Figure 14.3).

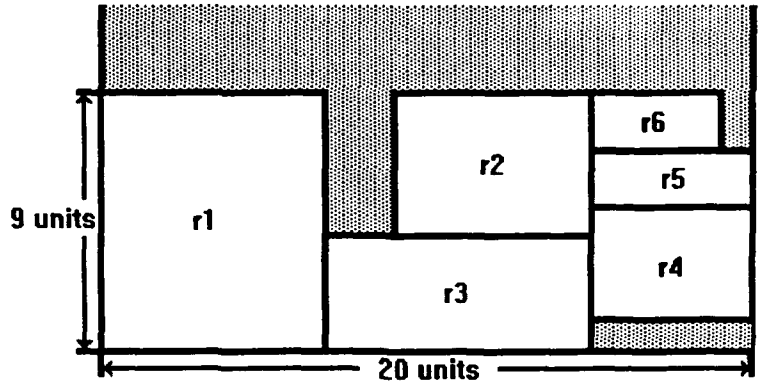


Figure 14.3: Realized Solution to the Bin-Packing Test Problem

There are four main reasons why so many local minima are developed for a problem of this size. First, consider the achieved solution (Figure 14.3) in relation to the problem statement (Figure 14.2). Note that while both have the same minimum height, the location of the blocks is different. It is obvious that two distinct global solutions of the same height are going to be different local minimum vertices of the feasible region. More precisely, consider a "reflection" of Figure 14.3 about the axis $x = 10$. This would result in different arrangement of blocks, occurring at a different vertex. However, it would still have the same global minimum height, therefore it must be a local minimum. It is easy to see that there are many combinations of the rectangles' positions which achieve the same global minimum height and, as a result, many global minima. This same theory can be applied to all the local minima combinations of blocks which are not the global minimum. Hence, this further increases the effect of multiple block arrangements on the number of local minima.

A second factor of the bin-packing problem which greatly increases the number of local minima is the fact that the blocks can "defy gravity." That is, as

evidenced by block r_4 in Figure 14.3, they are not forced to be directly on top of another rectangle or the bottom of the bin. Therefore, the empty space below r_4 can be regarded as another rectangle. It can go either below r_4, above r_4 but below r_5, above r_5 but below r_6, or above r_6. Again, this greatly increases the number of local minimum vertices by allowing more combinations of the positions of the rectangles.

The third reason that the bin-packing problem generates so many local minima involves the formulation of the non-overlapping constraints. Consider blocks 3 and 6 in Figure 14.3. Block 3 is both to the left of and below block 6. Thus, with all other variables remaining the same, this arrangement of rectangles is a local minimum with either $(x_{3\,6}, y_{3\,6}) = (0, 0)$ or $(1, 0)$. Therefore, when any two blocks in a local minimum arrangement could meet two of the non-overlapping constraints, two distinct local minima vertices are created.

Lastly, the method of using $(x_{i\,j}, y_{i\,j})$ variables also increases the number of local minima. Since there is no way to generate constraints which force these variables to be 0 *or* 1, we must formulate the *function* to drive them to be either 0 or 1 at the *global minimum*. However, this does not prevent them from being between 0 and 1 at other local minimum vertices. In fact, when running the presented bin-packing problem, the most often found local minimum had $(x_{i\,j}, y_{i\,j})$ variables which were not at 0 or 1. Therefore, by allowing these variables to range between 0 an 1, many more local minimum vertices are created, making the problem significantly more difficult.

In conclusion, the formulation of the two-dimensional bin-packing problem causes many local minima to be generated and is extremely difficult to solve using the stochastic algorithm. However, different formulation techniques could greatly reduce the number of local minima and make it more suitable to the stochastic

algorithm. The development of a lower solution bound, which stops the algorithm when a certain global minimum function value is achieved, could also significantly increase the stochastic algorithm's performance on this problem. Despite the difficulty, the formulation and solution of the two-dimensional bin-packing problem demonstrates a potential real-world problem to which the stochastic algorithm can be applied.

Section 15
Conclusion

In conclusion, a number of interesting computational results were discovered when analyzing the implementation of the stochastic algorithm for solving linearly constrained concave global minimization problems on the Cray X-MP and Connection Machine CM-2 supercomputers. First, it was obvious that the implementation of the algorithm on the CM-2 is *not* efficient for the problem size tested. Although the computational results from the test problems with known solutions (Floudas and Pardalos 1990) were always at least as good as (and in one instance better than) the "best known solution", execution data indicates that perhaps the algorithm does not contain enough fine-grain parallelism to efficiently use a massively parallel SIMD machine. However, the Cray X-MP implementation of the stochastic algorithm was much faster. While achieving the same correct results for every test problem, the high instruction speed, coarse-grain architecture of the Cray X-MP proved to be hundreds of times faster in solving (GP) than the CM-2.

When the computational analysis verified the efficiency of the algorithm on small sized test problems with known solutions, we then focused on the generation and solution of larger, randomly generated problems. From these results, several additional conclusions were reached. First, the question of problem difficulty was explored. CPU time on the Cray X-MP was modeled as a linear relationship of the product of the number of local minima and the number of pivots per trial. We saw that the stochastic algorithm was capable of solving reasonably large problems in a tolerable amount of execution time. For example, problems with 60 variables, 25 constraints, and over 500 local minima were solved in four to eight minutes of CPU

time. Thus, the stochastic algorithm can solve *significantly* larger problems of the form (GP) than previous deterministic methods.

It has previously been asserted that the two characteristics of quadratic functions of the form (GP) which affect problem difficulty are the location of the global maximum and whether the eigenvalues are equal or unequal (Floudas and Pardalos 1990). The computational results from the randomly generated quadratic test problems supported the theory about eigenvalues. However, from the random concave quadratic test problems which were generated, virtually *no* difference in difficulty was noted between those where the global maximum was located within the feasible region and those where the global maximum of the quadratic function was unconstrained.

Finally, the new stopping rule based on the bayesian estimate of the expected volume of the feasible region was much more efficient than previous stopping rules. However, it appears that the technique of solving for local minima in the "local phase" is finding the global minimum very early. Therefore, although (SR2) requires significantly fewer trails to be run before termination than (SR1), a large majority of these are occurring *after* the global minimum is found and are simply repetitions to satisfy (SR2).

The stochastic algorithm could be a particularly useful solution technique for linearly constrained concave global minimization problems if it is desired that a large percentage of the local minima for a problem be found. Likewise, the method works for any concave function, and does not require any special structure or other stipulations in order to be applied. In conclusion, the implementation of the stochastic algorithm appears to be a viable method for solving problem (GP) for modest sized ($\omega \leq 500$) problems.

Section 16
References

Boender, C.G.E., and A.H.G. Rinnooy Kan. 1987. Bayesian stopping rules for global optimization methods. *Mathematical Programming* 37(1):59-80.

Byrd, R.H., C.L. Dert, A.H.G. Rinnooy Kan, and R.B. Schnabel. 1990. Concurrent stochastic methods for global optimization. *Mathematical Programming* 46(1):1-29.

Cabot, A.V., and R.L. Francis. 1970. Solving certain nonconvex quadratic minimization problems by ranking the extreme points. *Operations Research* 18(1):82-86.

Chvátal, V. 1983. Linear Programming. New York: W.H. Freeman and Company.

Coffman, E.G., M.R. Garey, D.S. Johnson, and R.E. Tarjan. 1980. Performance Bounds for Level-Oriented Two-Dimensional Packing Algorithms. *Society for Industrial and Applied Mathematics Journal on Computing* 9(4):808-826.

Connection Machine Model CM-2 Technical Summary. 1989. Version 5.1. Thinking Machines Corporation, Cambridge, MA.

Connection Machine User Guide. 1991. Army High Performance Computing Research Center Document UG-0002. Minneapolis, MN.

Falk, J.E., and K.R. Hoffman. 1976. A successive underestimation method for concave minimization problems. *Mathematics of Operations Research* 1(3):251-259.

Falk, J.E., and R.M. Soland. 1969. An algorithm for separable nonconvex programming problems. *Management Science* 15(9):550-569.

Floudas, C.A. and P.M. Pardalos. 1990. A collection of test problems for constrained global optimization algorithms. In *Lecture Notes in Computer Science 455*, ed. G. Goos and J. Hartmanis. Berlin: Springer-Verlag.

Hillis, W.D. 1986. The Connection Machine. Cambridge: The Massachusetts Institute of Technology Press.

Law, A.M., and W.D. Kelton. 1991. Simulation Modeling and Analysis. New York: McGraw-Hill.

Lazou, C. 1988. Supercomputers and their Use. Oxford: Clarendon Press.

Martin, J.C. 1991. Introduction to Languages and the Theory of Computation. New York: McGraw-Hill.

Minkoff, M. 1981. Methods for Evaluating Nonlinear Programming Software. In *Nonlinear Programming 4*, ed. O.L. Mangasarian, R.R. Meyer, and S.M. Robinson. New York: Academic Press.

MSC User Guide - MSI Edition. 1991. University of Minnesota Supercomputer Institute, Minneapolis, MN.

Phillips, A.T. 1988. Parallel Algorithms for Constrained Optimization. PhD dissertation, University of Minnesota, Minneapolis, MN.

Phillips, A.T., J.B. Rosen, and M. van Vliet. 1991. A parallel stochastic method for solving linearly constrained concave global minimization problems. University of Minnesota Supercomputer Institute Research Report UMSI 91/192, University of Minnesota, Minneapolis, MN.

Quinn M.J. 1987. Designing Efficient Algorithms for Parallel Computers. New York: McGraw-Hill.

Thieu, T.V. 1980. Relationship between bilinear programming and concave minimization under linear constraints. *Acta Mathematica Vietnamica* 5:106-113.

Tui, H. 1964. Concave programming under linear constraints. *Doklady Akademii Nauk SSSR* 159:32-35. English translation in *Soviet Mathematics Doklady* 5:1437-1440.

Winston, W.L. 1987. Operations Research: Applications and Algorithms. Boston: Duxbury Press.

Van Dam, W.B., J.P.G. Frenk, and J. Telgen. 1983. Randomly generated polytopes for testing mathematical programming algorithms. *Mathematical Programming* 26:172-181.