A new parallel algorithm for global optimization with application to the molecular cluster problem

Aiping Liao[1]

Advanced Computing Research Institute
Cornell Theory Center
Cornell University
Ithaca, NY 14853

July 25 1994

Abstract

In this paper we present a simple algorithm for global optimization. This algorithm combines random searches with efficient local minimization algorithms. The proposed algorithm begins with an initial "local minimizer." In each iteration, a search direction is generated randomly, along which some points are chosen as the initial points for the local optimization algorithm and several "local minimizers" are obtained. The next iterate is determined by comparing these local minimizers. We will discuss the expected number of iterations for finding a global minimizer with this algorithm. Several variants of the algorithm that take advantage of the partially separable structure are proposed for the Lennard-Jones cluster problem and tested on the IBM SP1 parallel computer. Our numerical results show that our algorithms are promising.

Key words. global optimization, partially separable structure, Lennard-Jones potential function.

[1] This research was conducted using the resources of the Cornell Theory Center, which receives major funding from the National Science Foundation, and New York State. Additional funding comes from the Advanced Research Projects Agency, the National Institutes of Health, IBM Corporation and other members of the center's Corporate Research Institute.

1 Introduction

In practice we are often confronted with the global optimization problem

$$\min_{x \in \mathcal{R}^n} f(x) \tag{1}$$

where $f(x) : \mathcal{R}^n \to \mathcal{R}$ is a continuous function. Examples are, structural optimization, engineering design, allocation and location problems, and protein folding.

There are many efficient algorithms for finding a local minimizer of f, e.g., Newton's method, the quasi-Newton method, and the trust region method (see, for example, Fletcher [5], Dennis and Schnabel [4], and Gill, Murray and Wright [7]). However, these local algorithms make use of local information and cannot predict the behavior of the objective function far away from the current iterate. Once the iterate falls into an absorbing neighborhood of a local (but not global) minimizer it cannot escape. On the other hand it seems that random search (or random walk) is a generally acceptable technique for overcoming this locality barrier. The random search method usually needs many trials, even if the iterate is in the absorbing neighborhood of a global minimizer. It is thus desirable to combine random search with the local minimization procedure and yet avoid the difficulties of both methods. We also note that for many problems the local minimizers (including the global one) are not wide apart. In particular, if we solve the large-scale problem by solving a sequence of smaller sized subproblems whose global solutions approach a global solution of the original problem (i.e. the build-up technique), the global solutions of these subproblems are usually clustered. Therefore, it may be important to pay more attention on the near neighborhood of the current iterate (e.g., a global solution of the subproblem) than that far away. This is the central philosophy of our method and this philosophy is incorporated into the probability P_1 defined later which characterizes our algorithm. This probability P_1 distincts our method from others as we note that the simulated annealing is based on the Boltzmann distribution [10] and the classical Monte Carlo method is based on the uniform distribution.

The basic idea of our algorithm is that we take a "local minimizer" as the current iterate and cover it with a "search region." This search region can be, for example, a large ball. Then we randomly generate a search direction along which a point is picked up randomly within the search region and apply the local minimization procedure with this point as the initial point. If the output of the minimization procedure is better than the current iterate we then take it as the

the next iterate. Otherwise, we do the random search again until a better point is found.

We will present the conceptual algorithm in section 2 and discuss the convergence properties of this algorithm; in section 3 some variants of the conceptual algorithm are given and parallel implementations are discussed; in section 4 we tailor our algorithms for solving the Lennard-Jones type molecular conformation problems by exploiting the special structure of the problem, i.e., the partial separability, and present the numerical results of the tailored algorithm on SP1 parallel computer; we give the conclusion and some discussions in section 5.

2 The model algorithm

In this section we will describe a very simple algorithm for solving problem (1). We assume the availability of a local minimization procedure called LOCMIN. For any point x the output of LOCMIN with x as the initial point is given by LOCMIN(x).

Algorithm 1

Initialization. Given $x_c(=0) \in \mathcal{R}^n$ as the center of a ball, let the initial approximate solution be $x_* = x_c$.

Until convergence, do

(1). Generate randomly a search direction d with $\|d\| = 1$ (throughout this paper $\|\cdot\|$ denotes the l_2 norm).

(2). Select "randomly" a point $\bar{x}$ in the half-line $\{x = x_c + \alpha d : \alpha \geq 0\}$.

(3). Get $\bar{x}_* = \text{LOCMIN}(\bar{x})$. If $f(\bar{x}_*) < f(x_*)$ set $x_* = \bar{x}_*$, otherwise, x_* remains unchanged.

End

In step (2) of the above algorithm, the point $\bar{x}$ can be chosen according to some distribution. For example, the uniform distribution $\mathcal{U}(0, R)$, if we know that the global minimizer is located within a ball of radius R, or an F-distribution if no such

R is available. The point $\bar{x}$ can also be chosen on the line $\{x = x_c + \alpha d : \alpha \in \mathcal{R}\}$ according to a Gaussian distribution. In order to simplify the analysis, we assume that $x_c = 0$ and the global minimizer x_{opt} exists and $x_{opt} \in B(0, R)$, where $B(0, R)$ is a ball of radius R centered at 0. In step (2), $\bar{x}$ is chosen according to the distribution $\mathcal{U}(0, R)$. We note that, in practice, it is often the case that the best solution within a given ball $B(0, R)$ is desired or a ball which contains the global minimizer x_{opt} can be provided.

Let D be the set of all possible outputs of the local procedure LOCMIN, i.e.,

$$D := \{x : x = \text{LOCMIN}(x_{input}), \text{for some } x_{input} \in B(0, R)\}.$$

We will call D the station set and any point in D a station point. Obviously, $x_{opt} \in D$. For each $x_s \in D$ there is an associated absorbing set defined by

$$A(x_s) := \{x \in D : x_s = \text{LOCMIN}(x)\}.$$

We assume that each $A(x)$ is measurable. Any point x can be expressed uniquely as (σ, r) with $\sigma = x/\|x\| \in B(0, 1)$ and $r = \|x\|$. Thus for a set A, the probability that a point x, chosen randomly from $B(0, R)$, belongs to A is

$$\begin{aligned} P_0(A) &:= \frac{\text{volume}(A)}{\text{volume}(B(0, R))} \\ &= \int_{S_0} \int_r \phi(\sigma, r) r^{n-1} d\sigma dr / \text{volume}(B(0, 1)) \end{aligned} \tag{2}$$

where $S_0 = \{x : \|x\| = 1\}$ is the surface of the ball $B(0, 1)$ and $\phi(\sigma, r)$ is the index function of set A defined by

$$\phi(\sigma, r) = \begin{cases} 1 \text{ if } (\sigma, r) \in A \\ 0 \text{ otherwise.} \end{cases}$$

Proposition 2.1 *Suppose that $x \in B(0, R)$ is chosen in the following way: we first choose randomly $d \in S_0$ according to a uniform distribution and then choose $r \in (0, R)$ according to the uniform distribution $\mathcal{U}(0, R)$ and set $x = r \cdot d$. Let $P_1(A)$ denote the probability that x belongs to A, then $P_1(A) \geq n^{-1} P_0(A)$; and $P_1(A) \geq n^{-1} \rho^{1-n} P_0(A)$ if $A \subset B(0, \rho R)$ where $\rho \in (0, 1]$.*

Proof. Consider a small volume element at (σ, r):

$$E_{(\sigma,r)} := \{(\sigma, r) + (\sigma', r') : \sigma' \in d\sigma, r' \in dr\}$$

where $d\sigma$ is a small set in S_0 and dr is small set in $(0, R)$. Then

$$P_1(E_{(\sigma,r)}) = \frac{d\sigma}{\text{area}(S_0)} \cdot \frac{dr}{R} \tag{3}$$

where area(S_0) is the area of the surface of the ball $B(0,1)$. Integrating (3) over all possibilities, we have

$$P_1(A) = \frac{\int_{S_0} \int_r \phi(\sigma, r) d\sigma dr}{\text{area}(S_0) R}. \tag{4}$$

Suppose $A \subset B(0, \rho R)$. Noting that volume$(B(0,1)) = n^{-1}\text{area}(S_0)R^n$ it follows from (2) that

$$\begin{aligned} P_0(A) &:= \frac{\int_{S_0} \int_r \phi(\sigma, r) r^{n-1} d\sigma dr}{\text{volume}(B(0,1))} \\ &= \frac{\int_{S_0} \int_r \phi(\sigma, r) (\frac{r}{R})^{n-1} d\sigma dr}{n^{-1}\text{area}(S_0)R} \\ &\leq n\rho^{n-1} P_1(A). \end{aligned}$$

Taking $\rho = 1$ we have $P_1(A) \geq n^{-1} P_0(A)$. □

This proposition shows that Algorithm 1 puts more weight on the area that is closer to the center than that far from the center. For example, if $\rho = 1/2$ then P_1 is greater than P_0 for any set that is in the smaller ball $B(0, R/2)$. We will call the integration

$$\text{volp}(A) := \int_{S_0} \int_r \phi(\sigma, r) d\sigma dr$$

the projected volume of set A. From Proposition 2.1 we can derive the expected number of iterations for Algorithm 1 to find x_{opt}. If the projected volume of the absorbing set of x_{opt} is known a priori, then the expected number of iterations is

$$N_1 = \frac{1}{P_1} = \frac{\text{area}(S_0) R}{\text{volp}(A_{opt})}.$$

However volp is usually hard to obtain, we usually can only get some information about volume(A_{opt}). For example, suppose we use Newton's method as the local minimization algorithm. If the function f is simple enough to find positive constants $\kappa_0, \kappa_1, \kappa_2$ such that for all x with $\|x - x_{opt}\| < \kappa_0$ we have $\|H(x)\| < \kappa_1$ and

$$\|g(x_{opt}) - g(x) - H(x)(x_{opt} - x)\| \leq \kappa_2 \|x - x_{opt}\|^2$$

where H is the Hessian and g denotes the gradient, then $B(x_{opt}, r_{opt}) \subset A(x_{opt})$ with $r_{opt} = \min\{\kappa_0, 1/(\kappa_1\kappa_2)\}$. Thus

$$\begin{aligned} N_1 &= \frac{1}{P_1} \\ &\leq \frac{n}{P_0} = \frac{n\text{volume}(B(0,R))}{\text{volume}(A_{opt})} \\ &\leq \frac{n\text{volume}(B(0,R))}{\text{volume}(B(x_{opt}, r_{opt}))} \\ &\leq n(\frac{R}{r_{opt}})^n. \end{aligned}$$

2.1 The proposed algorithm

We now describe our algorithm based on the model algorithm in the previous section. The basic idea of our algorithm is that during each iteration we use Algorithm 1 with the current iterate as the center to find a better point and set it to be the next iterate.

Algorithm 2

Initialization. Given $\bar{x}_0 \in \mathcal{R}^n$, take $x_0 = \text{LOCMIN}(\bar{x}_0)$ as the initial iterate. Let R_0 be the initial radius. Set $k = 0$.

Until convergence, do

(1). Use Algorithm 1 with center $x_c = x_k$ and radius R_k to find a better point, x_{k+1}.

(2). Update the radius to get R_{k+1} according to the current information. Set $k \leftarrow k + 1$.

End

We note that if we know a priori an R such that $x_{opt} \in B(0, R)$, then we can simply take $R_k = 2R$ for all k, or take $\{R_k\}$ to be a monotonically increasing sequence. For convenience, we assume such an R exits and $R_k = 2R$ for all k in the following discussion.

We note that, from Proposition 2.1, Algorithm 1 pays more attention on the set near the center. Therefore, Algorithm 2 represents a conservative policy: do not

move away from the current area unless a better point is found. Once the iterate is in the absorbing neighborhood of a global minimizer, x_{opt}, all the following iterates will remain at x_{opt}. This is difference from the simulated annealing algorithm ([10]) in which the global minimizer could be missed.

Let N denote the expected number of iterations that is needed for Algorithm 2 to find x_{opt}. Let $q = P_0(A_{opt})/(n2^n)$. Then $P_1^{(k)}(A_{opt}) \geq q$ for each k (noting that since the center is not fixed, $P_1(A_{opt})$ varies with k). A bound on the expected number of iterations N is given below.

$$\begin{aligned} N &= P_1^{(1)} + 2(1 - P_1^{(1)})P_1^{(2)} + 3(1 - P_1^{(1)})(1 - P_1^{(2)})P_1^{(3)} + \cdots \\ &\leq 1 + 2(1 - q) + 3(1 - q)^2 + \cdots \\ &= q^{-2}. \end{aligned}$$

We now discuss possible parallel implementations of this algorithm. Since the main computation is performed in Algorithm 1, we will focus on Algorithm 1 only. We suppose there are n_p processors with labels $p(0), p(1), \ldots, p(n_p - 1)$. There are two obvious ways to parallelize this algorithm (we do not discuss here the parallel local minimization procedure though it is important for the overall efficiency of the algorithm). First consider parallelism at the high level (or the σ-level): for each of these processors, Algorithm 1 is carried out concurrently and then the best one of these n_p outputs is taken as the new center for the next iteration. The other possible way to exploit parallelism is at the low level (or the r-level): for each direction $d \in S_0$, these n_p processors work concurrently on finding a best point along d. Since these processors work independently during each step, these two parallel procedures are equivalent to choosing n_p random points in S_0 and $(0, R)$ respectively. The probability that at least one of these n_p points is in A_{opt} for the high level procedure is

$$\begin{aligned} PP_h(A) &= Part1 + Part2 \\ &=: (n_p \int_r \int_{S_r : area(S_r) < \text{area}(S_0)/n_p} d\sigma dr)/(\text{area}(S_0)R) + \int_{r : area(S_r) \geq \text{area}(S_0)/n_p} R^{-1} dr \end{aligned}$$

where $S(r) = \{\sigma \in S_0 : \phi(\sigma, r) = 1\}$. Similarly, for the low level procedure we have

$$\begin{aligned} PP_l(A) &= Part1 + Part2 \\ &=: (n_p \int_{S_0} \int_{r : area(L_\sigma) < R/n_p} dr d\sigma)/(\text{area}(S_0)R) + \int_{\sigma : area(L_\sigma) \geq R/n_p} \text{area}(S_0) d\sigma \end{aligned}$$

where $L_\sigma = \{r : \phi(\sigma, r) = 1\}$. Obviously, if n_p is very large then the expected efficiency will decrease to 0 in which case Part1 will disappear. It is also clear that in general the high level procedure can use more processors before the 0-efficiency is reached than the low level one. These two methods can be mixed to yield the third method: we divide these n_p processors into two groups, one group works on the high level parallel procedure and the other works on the low level procedure.

Our implemented algorithm is the high level one. Besides, we employ another local procedure LOCMIN_1 to enhance the performance where

$$x_{out} = \text{LOCMIN}_1((x, d), [\alpha_1, \alpha_2])$$

is a local minimizer of f on the interval $[x + \alpha_1 d, x + \alpha_2 d]$. The following is a detailed description of the proposed parallel algorithm.

Algorithm 3

For each processor $p(0), \ldots, p(n_p - 1)$ **do**

Initialization. Given $\bar{x}_0 \in \mathcal{R}^n$, take $x_0 = \text{LOCMIN}(\bar{x}_0)$ as the initial iterate. $k = 0$.

Until convergence, do

(1). Generate randomly a search direction d with $\|d\| = 1$.

(2). Take $r_d \in \mathcal{U}(0, 1)$ and choose s intervals $[\alpha_j, \alpha_{j+1}]$, $j = 1, \ldots, s$ such that $\alpha_1 = 0$ and $\alpha_{s+1} \geq 2R$. For each $j = 1, \ldots, s$, find

$$\pm y_j = \text{LOCMIN}_1((x_k, \pm d), [\alpha_j r_d, \alpha_{j+1} r_d])$$

and let $y_j = \arg\min\{f(x) : x = \pm y_j\}$ then get $\bar{x}_j = \text{LOCMIN}(y_j)$. Take $\bar{x} = \arg\min\{f(\bar{x}_j) : j = 1, \ldots, s\}$. Send $(\bar{x}, f(\bar{x}))$ to processor $p(0)$.

(3). For processor $p(0)$: get pairs $(\bar{x}, f(\bar{x}))$ from all processors and choose the best one as the next iterate x_{k+1}, send the pair $(x_{k+1}, f(x_{k+1}))$ to all other processors.

(4). Get pair $(x_{k+1}, f(x_{k+1}))$ from processor $p(0)$, set $k \leftarrow k + 1$ and go to step (2).

End

We note that in step (2) the s intervals $[\alpha_j, \alpha_{j+1}], j = 1, \ldots, s$ can be chosen in various ways. In our actual implementation we choose them as $[2^{j-1}, 2^j]$ ($[3^{j-1}, 3^j]$ for large problems), for $j = 1, 2, \ldots, J$ where J is the smallest integer such that $2^{J-1} \geq 2R$ ($3^{J-1} \geq 2R$). This strategy also reflects the "nearsighted" policy. Actually, the proposed algorithm seems to be suitable for problems whose local minimizers (including the global minimizer) are not far apart. Also this algorithm depends on efficient local minimization procedures.

3 Numerical test with the molecular cluster problem

In this section we propose some implementation techniques for the cluster problem. However we note all these methods are applicable to any problems with the partially separable structure:

$$f(x) = \sum_{i=1}^{ne} f_i(x), \tag{5}$$

where each of the ne element functions f_i depends only on a few variables. Since every function f with a sparse Hessian is partially separable (Griewank and Toint [8]), our methods have a broad application domain.

The cluster problem is of interest in many applications in biochemistry and physics. This problem can be characterized as finding the global minimizer of a potential energy function. There are many algorithms for solving this problem (see, for example, Coleman, Shalloway, and Wu [2], Xue [17], Wille [16], and Byrd et al. [1] to name a few). Following Coleman, Shalloway, and Wu [2], we will consider a special potential function called the Lennard-Jones potential function:

$$f(x) = \sum_{i=1, j>i}^{n_{atom}} h(\|x_i - x_j\|) \tag{6}$$

where n_{atom} is the number of atoms whose locations are $x_j \in \mathcal{R}^3$, $j = 1, \ldots, n_{atom}$ and $x \in \mathcal{R}^n$ with $n = 3n_{atom}$ and $x(3(j-1)+k) = x_j(k), k = 1, 2, 3, j = 1, \ldots, n_{atom}$, function h is given by

$$h(z) = \frac{1}{z^{12}} - \frac{2}{z^6}.$$

Noting that f is function of the distances between these atoms, we can fix one atom at the origin, say $x_1 = 0$. Thus, n, the dimension of x, can be reduced to $3(n_{atom} - 1)$. The problem is thus

$$\min_{x \in \mathcal{R}^n} f(x) \tag{7}$$

where $n = 3(n_{atom} - 1)$.

We note that this problem has two good properties that can be used to reduce the sampling space needed for Algorithm 3: one is the boundedness of all the global minimizers; the other is its separability. We first study the boundedness of its global solutions which is stated in the following proposition.

The following proposition provides a bound for x_{opt} of problem (7).

Proposition 3.1 *The global minimizers for problem (7) exist and for any global minimizer x_{opt} we have $\|x_{opt}\| \leq R$ where $R = (n_{atom} - 1)\sqrt{n_{atom} - 1}$.*

Proof. It can be verified that $h(z)$ is monotonically increasing for $z \in (1, \infty)$ and monotonically decreasing for $z \in (0, 1)$. We now claim that for any $\bar{x}$ with $\|\bar{x}_J\| > (n_{atom} - 1)$ where $J = \arg\max\{\|\bar{x}_j\| : j = 1, \ldots, n_{atom} - 1\}$ we can find a new point $\hat{x}$ such that $\|\hat{x}\| < \|\bar{x}\|$ and $f(\hat{x}) < f(\bar{x})$. Consider $n_{atom} - 1$ "intervals"

$$\{y \in \mathcal{R}^3 : \|y\| \leq \|\bar{x}_J\|, y^T \bar{x}_J / \|\bar{x}_J\| \in (k-1, k]\}$$

$k = 1, \ldots, n_{atom} - 1$. Since $\|\bar{x}_J\| > n_{atom} - 1$ at least one of these intervals, say the k'-th, is empty. Thus the hyperplane

$$H_s := \{y \in \mathcal{R}^3 : y^T \bar{x}_J / \|\bar{x}_J\| = k'\}$$

separates the n_{atom} atoms into two groups: P_0, one contains 0 and P_1, the one contains $\bar{x}_J$ and the distance between these two groups is greater than 1. For a given small number $\epsilon > 0$ we define $\hat{x}$ as follows. $\hat{x}_j = \bar{x}_j$, if $\bar{x}_j \in P_0$ and $\hat{x}_j = \bar{x}_j - \epsilon \bar{x}_J / \|\bar{x}_J\|$, if $\bar{x}_j \in P_1$. We can choose ϵ very small so that the distance between these two groups is still greater than 1. It is easy to verify that $\|\hat{x}\| < \|\bar{x}\|$. It follows from the property of the function h that $f(\hat{x}) < f(\bar{x})$. The claim is thus true. Noting that function f can be extended to be a continuous function by defining $f(x) = +\infty$ if there are some i, j such that $x_i = x_j$, it follows from the claim that the global minimizers exist and the associated atoms' positions $(x_{opt})_j, j = 1, \ldots, n_{atom} - 1$ satisfy $\|(x_{opt})_j\| \leq n_{atom} - 1$. Thus $\|x_{opt}\| \leq (n_{atom} - 1)\sqrt{n_{atom} - 1} =: R$. □

We note that problem (7) may have several global minimizers. Since the function depends only on the distances between these atoms we can shift the origin atom of one global minimizer to any other atom and get a different global minimizer. The above lemma guarantees that all of these global minimizers are within a big ball $B(0,R)$. We will use this bound in our algorithm. Note that if two atoms are very close then the value of f can be very large. To avoid this situation we take the initial point $\bar{x}_0$ to be

$$\bar{x}_0(i) = \begin{cases} l & \text{if } i = 9(l-1) + 4(j-1) + 1,\ j = 1,2,3 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

for $i = 1, \ldots, n$, i.e., the associated atoms' positions are

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix}, \ldots .$$

Our search direction d is chosen as follows. We first generate a vector, d, of normally distributed numbers, with a mean of 0 and a standard deviation of 1. Then scale d so that $\|d\| = 1$. It can be shown that the point d, produced in this way, is of a uniform distribution over the sphere $\|x\| = 1$. Our code was written in FORTRAN and implemented in double precision arithmetic.

We tested the algorithm on an SP1 parallel computer with PVM as the message passing library. SP1 – IBM's Scalable POWERparallel system 9076 – is a distributed-memory, Multiple Instruction Multiple Data (MIMD), high-performance computer. It consists of a cluster of IBM RS6000 processors connected by a high performance switch. PVM – the Parallel Virtual Machine library – is a message passing library developed at Oak Ridge National Laboratories for a variety of platforms. PVM is very flexible and easy to use. We include 10 RS6000's in our hostfile, i.e., the computations are carried out by 10 machines. We use the optimization subroutines (BFGS method) of the IMSL library as the local minimization procedures and use their default parameters. In step (2) of the algorithm the s intervals are chosen as: $[2^{j-1}r_d, 2^j r_d]$, $j = 1, 2, \ldots, J_2$, if $n_{atom} \leq 16$ and $[3^{j-1}r_d, 3^j r_d]$, $j = 1, 2, \ldots, J_3$, if $n_{atom} \geq 17$, where $r_d \in \mathcal{U}(0,1)$ and J_2 (J_3) is the smallest integer such that 2^{J_2-1} (3^{J_3-1}) $\geq 2R$. The results are presented in Table 1. The algorithm terminates if an energy within 10^{-4} of the lowest energy cited in [2] is found. *nproc* is number of processors. *nproc* can be any number less than 262,144. The system will automatically load these processors into the physical machines. The time is the wall-clock time.

In our numerical experience we find that without the bound the algorithm may take a large number of iterations before the global minimizer is obtained. For example, when solving the problem with $n_{atom} = 8$ without the bound limitation, 6 atoms of the iterate approaches the solution of the problem with $n_{atom} = 6$ with the other two atoms being far away from the origin. We also note that it can probably improve the simulated annealing algorithm if the bound information is incorporated.

We now consider the parallel performance of our algorithm. Since our algorithm is nondeterministic, the standard definitions of speedup and efficiency do not apply. Nevertheless, we can measure the parallel performance of a nondeterministic algorithm by the average-speedup, S_{av}, and average-efficiency, E_{av}, defined as follows.

$$S_{av} = \frac{\text{average elapsed time of serial program}}{\text{average elapsed time of parallel program}},$$

$$E_{av} = \frac{S_{av}}{nproc}.$$

To highlight the parallel performance of our algorithm, we solve the problem with 20 atoms using $nproc$ processors with $nproc = 1, \ldots, 10$. For each $nproc = 1, \ldots, 10$ we solve the problem 11 times using different seeds for the random number generator and taking the average computation time as the average elapsed time in our definitions of the average-speedup and the average-efficiency. Loadleveler batch system from IBM was used for accurately measuring the running time. The average-speedup and the average-efficiency are represented in Figure 1 and Figure 2 respectively which indicate that our algorithm is worthy to be parallelized.

We also note that for our algorithm the average-efficiency can exceed 1.0 which is impossible for a deterministic algorithm. This is mainly due to facts that there is not much communication among processors, and the more processors the bigger the chance that the solution is found by one of them. We do not know if the average-efficiency will not exceed 1 if the sample size is sufficiently large (not just 11).

Now we exploit the structure of the problem. Note that for the problem with n_{atom} atoms if one of these atoms is removed or the distance between this atom and the rest of the cluster is large, then the original problem becomes one with $n_{atom} - 1$ atoms. This structure has been used in many previous works, such as [3], [9], [14], [1], [13], and [17]. Here we give an approach based on a continuity argument which applies to any problem with such a separability. Consider a

Table 1: Numerical results

n_{atom}	iter/time(in sec)/nproc
3	1/0.01/1, 1/0.02/2
4	1/0.06/1, 1/0.07/2
5	1/0.07/1, 1/0.06/2
6	2/0.50/1, 2/0.51/2, 2/0.56/3
7	1/0.38/1, 1/0.39/2
8	2/1.30/1, 2/1.32/2, 1/0.69/3
9	6/4.03/1, 2/1.65/2, 2/1.80/3
10	3/2.95/1, 15/13.38/2, 1/1.00/5
11	3/3.77/1, 1/1.48/2
12	10/16.52/1, 6/10.56/2, 5/9.07/5
13	17/31.98/1, 5/10.14/2, 18/34.14/5
14	4/9.81/1, 5/11.68/2, 2/6.01/10
15	24/63.35/1, 32/81.54/2, 2/6.01/10
16	17/50.84/1, 11/33.64/5, 2/6.19/20
17	26/60.39/5, 3/6.61/10
18	48/133.61/5, 10/28.3/10
19	4/13.73/5, 3/10.41/10
20	134/464.12/5, 20/70.21/10
21	21/81.43/5, 6/24.48/10
22	40/172.75/5, 22/92.76/10, 9/41.07/20
23	26/124.51/5, $>$ 51/$>$ 242.00/10, 11/53.14/20
24	$>$ 51/$>$ 264.94/10, 8/42.49/20
25	6/37.63/20
26	21/131.96/20, 13/84.65/40
27	21/170.40/40
36	58/898.04/60
54	41/1487.28/60

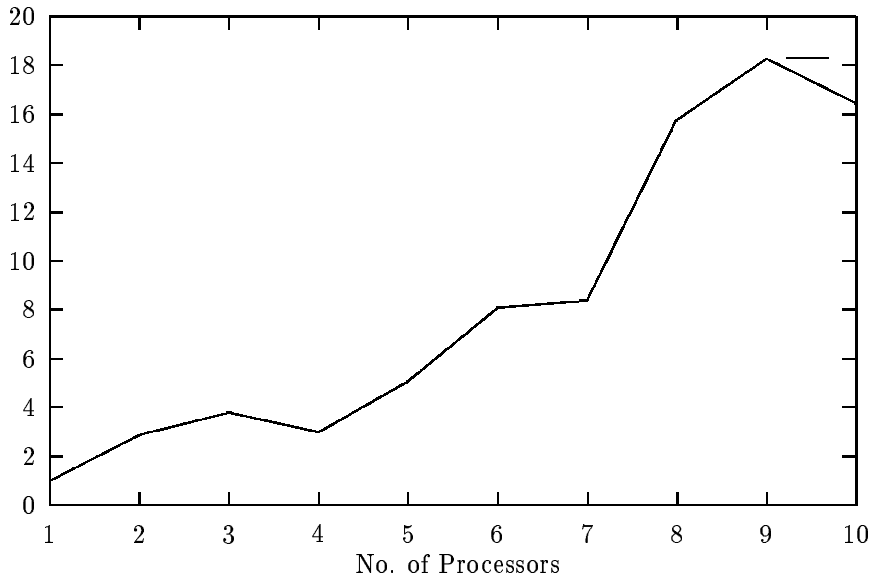

Figure 1: Average-speedup for the problem with 20 atoms.

parametered function

$$f_\varepsilon(x) := \sum_{i=1, j>i}^{n_{atom}-1} h_{\varepsilon_{ij}}(\|x_i - x_j\|) \tag{9}$$

where $\varepsilon = \{\varepsilon_{ij} : i = 1, \ldots, n_{atom} - 1; j = i+1, \ldots, n_{atom} - 1\}$ and each $\varepsilon_{ij} \in [0, 1]$. By the continuity of function f (or a modified version of f truncated from above) the global solutions vary continuously with the parameter ε. We thus can start with a global solution of a problem with fewer atoms and let the rest atoms "join in" graduately by increasing the corresponding $\varepsilon'_{ij}s$ from 0 to 1. The optimal solution x_ε forms a continuous trajectory and we call it an optimal trajectory. In [17] it is observed that "in some cases, the relaxation of a lattice local minimizer with a worse potential function value may lead to a local minimizer with a better potential function value." This observation can be justified with the continuity argument: the minimizer with a worse potential function may be close to a global optimal trajectory.

To make the optimal trajectory more attractive we employ another technique:

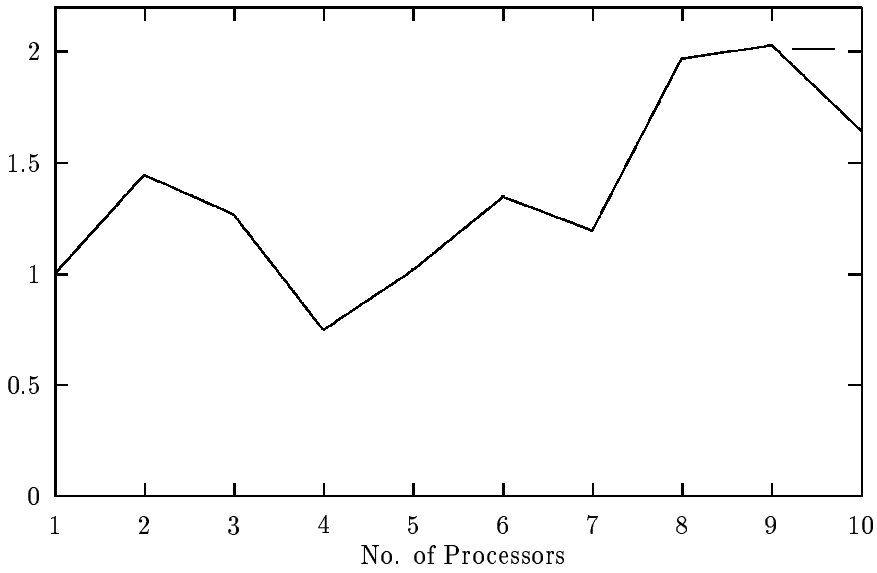

Figure 2: Average-efficiency for the problem with 20 atoms.

we take the objective function for next iteration to be

$$\bar{f}_\varepsilon(x) := \sum_{i=1,j>i}^{n_{atom}-1} h_{\varepsilon_{ij}}(\|x_i - x_j\|) + \delta(x - x_\varepsilon^c)^T(x - x_\varepsilon^c) \tag{10}$$

where δ is a positive constant and x_ε^c is the current iterate. The underlying idea of this technique is the following: if x_ε is an optimal trajectory for function f_ε then it is also an optimal trajectory for function

$$\hat{f}_\varepsilon(x) = f_\varepsilon(x) + \delta(x - x_\varepsilon)^T(x - x_\varepsilon).$$

Function $\hat{f}_\varepsilon$ is usually more convex than function f_ε.

We now state a detailed algorithm based on these techniques. We denote the objective function with i atoms ($i \leq n_{atom}$) by $f_{(i)}$.

Algorithm 4

Initialization. Let $\bar{x}_0$ be given by (8) and $x_0^{(3)} \in \mathcal{R}^6$ a global minimizer of function $f_{(3)}$ (whose value is 3). Set $k = 0$.

Until convergence, do

(1). Apply one iteration of Algorithm 3 with

$$((x_k^{(3+k)})^T, (\bar{x}_0(3(2+k)+1:3(3+k)))^T)^T$$

as the initial point and $\bar{f}_\varepsilon(x)$ as the objective function where ε is chosen as follows.

$$\varepsilon_{i,j} = \begin{cases} 0 & \text{if } j > 3+k \\ l/10 & \text{if } j+l = k+n_{atom}-2, l = 1,\ldots,9 \\ 1 & \text{otherwise.} \end{cases}$$

Denote the output as $x_{k+1}^{(3+k+1)}$, set $k \leftarrow k+1$.

End

If each output of step (1) is a global minizer of function $\bar{f}_\varepsilon$ then Algorithm 4 finds a global minimizer of f in $n_{atom}+10$ iterations. However since one iteration of Algorithm 3 may fail to find a global solution more outer iterations might be needed. We also note that if δ is large then it is hard to pull the iterate back to an optimal trajectory once it is out. We call Algorithm 4 the full-range algorithm since it starts with an optimal solution of the objective function associated with 3 atoms. Table 2 shows how far the full-range algorithm (with $\delta = 0.0001$ and $n_{atom}+10$ iterations being performed) can go before it fails to keep the optimal trajectory. This result shows that the more accurate the solution obtained by Algorithm 3, the further Algorithm 4 can go along the optimal trajectory.

One variant of the full-range algorithm is that we first get an accurate solution $x^{(4)}$ of f_4 using Algorithm 4 with the $x_0^{(3)}$ as the initial point, then using Algorithm 4 again but with $x^{(4)}$ as the initial point to obtain $x^{(5)}$, an solution of f_4, and so on until a solution of $f_{n_{atom}}$ is found. We call this variant the single-range algorithm. Table 3 presents the numerical results of the single-range algorithm. In this single-range algorithm we choose $\delta = 0$ and the initial point for Algorithm 3 is given by $((x^{(n_{atom}-1)})^T, (x^a)^T)^T$ where $x^a \in \mathcal{R}^3$ is given by $x_1^a = x_1^l + 1,$, $x_2^a = x_2^l$, and $x_3^a = x_3^l$ where x^l is the one of x^j with the largest l_2-norm and $x^j, j = 1, n_{atom}-2$ are the positions of the atoms with $x^{(n_{atom}-1)}$.

We note that Algorithm 4 can be strengthened by either adding in the new atom more slowly or taking more inner iterations with Algorithm 3 to get more accurate intermediate solutions.

Table 2: How far the full-range algorithm can go.

n_{atom}	f [2]	f/time(in sec)/nproc
4	-6.000000e+00	-6.000000e+00/2.35/10
5	-9.103852e+00	-9.103852e+00/3.25/10
6	-1.271206e+01	-1.271206e+01/5.00/10
7	-1.650539e+01	-1.650538e+01/6.69/10
8	-1.982149e+01	-1.982149e+01/9.67/10
9	-2.411336e+01	-2.411336e+01/11.66/10
10	-2.842254e+01	-2.842253e+01/12.97/10
11	-3.276597e+01	-3.276597e+01/16.42/10
12	-3.796761e+01	-3.796760e+01/18.89/10
13	-4.432681e+01	-4.432680e+01/23.16/10
14	-4.784517e+01	-4.784516e+01/30.71/10
15	-5.232265e+01	-5.232263e+01/33.14/10
16	-5.681575e+01	-5.681574e+01/34.46/10
17	-6.131801e+01	-5.681574e+01/50.10/10 -6.131800e+01/60.06/60

Table 3: the results of the single-range algorithm.

n_{atom}	f [2]	f/time(in sec)/nproc	iter
4	-6.000000e+00	-6.000000e+00/2.01/10	15
5	-9.103852e+00	-9.103852e+00/2.20/10	15
6	-1.271206e+01	-1.271206e+01/3.75/10	15
7	-1.650539e+01	-1.650538e+01/6.78/10	15
8	-1.982149e+01	-1.982149e+01/6.84/10	15
9	-2.411336e+01	-2.411336e+01/8.71/10	15
10	-2.842254e+01	-2.842253e+01/11.53/10	15
11	-3.276597e+01	-3.276597e+01/13.83/10	15
12	-3.796761e+01	-3.796760e+01/17.08/10	15
13	-4.432681e+01	-4.432680e+01/19.53/10	15
14	-4.784517e+01	-4.784516e+01/26.33/10	15
15	-5.232265e+01	-5.232263e+01/29.32/10	15
16	-5.681575e+01	-5.681574e+01/31.05/10	15
17	-6.131801e+01	-6.131800e+01/33.46/10	15
18	-6.653097e+01	-6.653095e+01/38.03/10	15
19	-7.265979e+01	-7.265978e+01/41.06/10	15
20	-7.717707e+01	-7.717704e+01/53.32/10	15
21	-8.168460e+01	-8.168457e+01/60.32/10	15
22	-8.680981e+01	-8.680978e+01/64.31/10	15
23	-9.284451e+01	-9.284447e+01/76.78/10	15
24	-9.734884e+01	-9.734882e+01/77.81/10	20
25	-1.023727e+02	-1.023727e+02/96.15/10	20
26	-1.083156e+02	-1.083156e+02/126.20/10	20
27	-1.128736e+02	-1.128736e+02/222.34/10	20

Table 4: Numerical results for constrained problems

problem	n	$f_{bestknown}$ ([6])	f_{opt}	x_{opt}
4.3 [6]	4	-4.5142	-4.5142	$(4/3, 4, 0, 0)^T$
4.4 [6]	4	-2.07[1]	-2.2168	$(4/3, 4, 0, 0)^T$
4.5 [6]	6	-11.96	-13.4019	$(1/6, 2, 4, 1/2, 0, 2)^T$

4 Concluding remarks

We have proposed a simple algorithm for global optimization. The expected number of iterations is considered and parallel implementations are discussed. Some variants of the basic algorithm are described and discussed. Numerical results with the Lennard-Jones potential function are promising. It seems that this algorithm is suitable for parallel machines such as SP1 and should be efficient for problems whose local minimizers are not far apart.

The efficiency of the local minimization algorithm is the crucial part of the proposed algorithm. Theoretically, if the number of processors is sufficiently large, the algorithm can find the global minimizer in a few iterations and thus the time spent on the local minimization will be the dominant one. If we use the limited memory BFGS method (see, for example, [12] and [11]), instead of the standard BFGS method, the wall-clock time for the calculations for problems with $n_{atom} \geq 25$ can be reduced by almost half.

We also note that our algorithm can be used for constrained optimization in an obvious way: simply replace LOCMIN with a local algorithm for constrained problem. To highlight the behavior of the algorithm for constrained problem we solved 3 simple problems (problems 4.3, 4.4, and 4.5) of Floudas and Pardalos [6]. The results are presented in Table 4. All runs were performed on one RS6000 and 150 iterations were carried out for each run.

Finally, to conclude this paper, we give an open problem of the theoretical complexity of the Lennard-Jones potential function: what is the theoretical complexity of finding a global solution of the Lennard-Jones potential function with n atoms if a global solution of the potential function with $n-1$ atoms is giving. A more challenge problem is proposed by Vavasis [15].

[1] This value is obviously wrong since it does not fit the associated x^*.

Acknowledgment: I would like to thank Professor Thomas F. Coleman and Dr. S. Chinchalkar for many discussions relating to this work and for their helpful comments and suggestions on the manuscript. I would also like to thank Professor J. Nocedal for providing Fortran code of the limited memory BFGS method.

References

[1] R. Byrd, T. Derby, E. Eskow, K. Oldenkamp, and R. Schnabel. A new stochastic/perturbation method for large-scale global optimization and its application to water cluster problems. Technical Report cu-cs-652-93, Department of Computer Science, University of Colorado at Boulder, 1993.

[2] T. F. Coleman, D. Shalloway, and Z. Wu. Isotropic effective energy simulated annealing searches for low energy molecular cluster states. *Computational Optimization and Applications*, 2:145–170, 1993.

[3] T. F. Coleman, D. Shalloway, and Z. Wu. A parallel build-up algorithm for global energy minimizations of molecular clusters using effective energy simulated annealing. Technical Report CTC93TR130, ACRI, Cornell University, 1993.

[4] J. E. Dennis, Jr. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983.

[5] R. Fletcher. *Practical Methods of Optimization.* John Wiley & Sons, Ltd, 1981.

[6] C. Floudas and P. Pardalos. *A Collection of Test Problems for Constrained Global Optimization Algorithms.* Springer-Verlag, 1990.

[7] P. Gill, W. Murray, and M. Wright. *Practical Optimization.* Academic Press, 1981.

[8] A. Griewank and Ph.L. Toint. On the constrained optimization of partially separable objective functions. In M. J. D. Powell, editor, *Nonlinear Optimization*, pages 301–312, London, 1982. Academic Press, London.

[9] B. Hingerty, S. Figueroa, T. Hayden, and S. Broyd. Prediction of DNA structure from sequence: a build-up technique. *Biopolymers*, 28:1195–1222, 1989.

[10] J. Kirkpatrick, Jr. C. D. Gellat, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[11] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Math. Program.*, 45:503–528, 1989.

[12] J. Nocedal. Updating quasi-Newton matrices with limited storage. *Math. Comput.*, 35:773–782, 1980.

[13] J. A. Northby. Structure and binding of Lennard-Jones clusters: $13 \leq n \leq 147$. *Journal of Computational Chemistry*, 87:6166–6178, 1987.

[14] M. Pincus, R. Klausner, and H. Scheraga. Calculation of the three-dimensional structure of the membrane-bound portion of melittin from its amino acid sequence. *Proceedings of National Academy of Science, USA*, 79:5107–5110, 1982.

[15] S. Vavasis. Open problems. *J. Global Optimization*, 4:343–344, 1994.

[16] L. T. Wille. Minimum-energy configuration of atom clusters: New results obtained by simulated annealing. *Chemical Physics Letters*, 133:405–410, 1987.

[17] G. Xue. Improvement on the Northby Algorithm for Molecular Conformation: Better Solutions. *J. Global Optimization*, 4:425–440, 1994.