Journal of Global Optimization, 4, 187–208 (1994)

Molecular Conformation on the CM-5 by Parallel Two-Level Simulated Annealing

GUOLIANG XUE*
Department of Computer Science and Electrical Engineering, University of Vermont, Burlington, VT 05405

Abstract. In this paper, we propose a new kind of simulated annealing algorithm called *two-level simulated annealing* for solving certain class of hard combinatorial optimization problems. This two-level simulated annealing algorithm is less likely to get stuck at a non-global minimizer than conventional simulated annealing algorithms. We also propose a parallel version of our two-level simulated annealing algorithm and discuss its efficiency. This new technique is then applied to the Molecular Conformation problem in 3 dimensional Euclidean space. Extensive computational results on Thinking Machines CM-5 are presented. With the full Lennard-Jones potential function, we were able to get satisfactory results for problems for cluster sizes as large as 100,000. A peak rate of over 0.8 giga flop per second in 64-bit operations was sustained on a partition with 512 processing elements. To the best of our knowledge, ground states of Lennard-Jones clusters of size as large as these have never been reported before.

Keywords: Molecular conformation, global optimization, simulated annealing, parallel algorithms

1. Introduction

Simulated annealing is a general purpose combinatorial optimization technique that has been proposed by Kirkpatrick et al. [19]. This method is an extension of a Monte Carlo method developed by Metropolis et al. [23], to determine the equilibrium states of a collection of atoms at any given temperature T. Since the method was first proposed in [19, 20], much research has been conducted on its use and properties. Some relevant references are: [3, 7, 9, 10, 16, 17, 24, 26].

In this paper, we present a new kind of simulated annealing algorithm called two-level simulated annealing which is less likely to get stuck at a non-global minimizer. Instead of moving from one solution to another, the two-level simulated annealing algorithm moves from one catchment basin of a local minimizer to the catchment basin of another (or the same) local minimizer. Advantages of this new algorithm over conventional simulated annealing algorithms are discussed. A new method of parallelizing a sequential simulated annealing (conventional or two-level) algorithm is presented next. The parallel algorithm is usually different from the sequential algorithm, but follows the same philosophy of simulated annealing algorithms and is very efficient. These techniques are then applied to the Molecular Conformation problem and implemented in F77 and message passing mode on the Thinking

*ALSO A RESEARCHER AT THE ARMY HIGH PERFORMANCE COMPUTING RESEARCH CENTER, UNIVERSITY OF MINNESOTA, MINNEAPOLIS, MN 55415

Machines CM-5. We first reduce the time complexity of an algorithm proposed by Northby [25] for Molecular Conformation. Then the improved Northby algorithm is combined with the two-level simulated annealing algorithm to get satisfactory results for Molecular Conformation problems of sizes much larger than ever reported before. These computational results demonstrate that the parallel two-level simulated annealing algorithm is a very powerful tool for solving certain class of hard combinatorial optimization problems.

The rest of the paper is organized as follows. In section 2, we present the two-level simulated annealing algorithm and discuss its advantage over conventional simulated annealing algorithms. In section 3, we present a paradigm for parallelizing sequential simulated annealing algorithms. In section 4, we present the Molecular Conformation problem, the combinatorial optimization problem associated with it, Northby's algorithm and its improvement, and a parallel two-level simulated annealing algorithm for solving the combinatorial optimization problem. In section 5, computational results on the CM-5 are presented. In section 6, we make some conclusion remarks.

2. Two-Level Simulated Annealing

Given a real-valued function $f(x)$ defined on a feasible domain $\mathcal{D}$, the general global optimization (minimization) problem is to find a point $x^* \in \mathcal{D}$ such that $f(x)$ is globally minimized at x^*, i.e., $f(x^*) \leq f(x)$ for all $x \in \mathcal{D}$. The general global optimization problem is stated as follows:

$$(P) \qquad \begin{array}{ll} \text{global min} & f(x), \\ \text{subject to} & x \in \mathcal{D}. \end{array}$$

A point $x^* \in \mathcal{D}$ is called a global minimizer of (P) if $f(x^*) \leq f(x)$ for all $x \in \mathcal{D}$. A point $x^* \in \mathcal{D}$ is called a local minimizer of (P) if $f(x^*) \leq f(x)$ for all x in a neighborhood of x^*. Here we are interested in finding a global minimizer of (P). There have been many approaches to solving (P), e.g., Multistart with Bayesian Stopping Rules [2], Genetic Algorithms [17], and Simulated Annealing [19, 20]. In this paper, we are interested in simulated annealing like algorithms.

The basic simulated annealing algorithm as proposed by Kirkpatrick et al. [19, 20] for solving (P) is stated in Figure 1, where α is a given constant in the interval (0, 1), *number_of_iterations* is a given positive integer, and *perturbation*() is a procedure which generates a new trial point be making a small perturbation from the current solution.

Algorithm_1: Simulated Annealing
Set T to a positive number (initial temperature). Set x_{old} to an initial feasible solution and compute $f_{old} := f(x_{old})$. Set $x_{best} := x_{old}$ and $f_{best} := f_{old}$.
repeat

```
    for i := 1 to number_of_iterations
        x_new := perturbation(x_old);
        f_new := f(x_new);
        generate a random number rand ∈ (0, 1);
        if ((f_new < f_old) or (rand < exp((f_old − f_new)/T))) then
            x_old := x_new; f_old := f_new;
            if (f_new < f_best) then
                x_best := x_new; f_best := f_new;
            endif
        endif
    endfor
    T := αT;
until (stopping criterion is met)
Output x_best and f_best as the best known solution and objective function value.
```

Figure 1. **Basic simulated annealing algorithm**

A simulated annealing algorithm differs from a conventional iterative improvement algorithm in that it not only accepts a solution with better objective function value but also accepts a solution with a worse objective function value conditionally. When the temperature T is high, the probability of accepting a solution with a worse objective function value is relatively large, making it relatively easy for the simulated annealing algorithm to go from the catchment basin corresponding to one local minimizer to the catchment basin corresponding to another local minimizer. It is expected that the algorithm will arrive at the catchment basin of the global minimizer before the temperature T gets very low. When the temperature T gets very low, the algorithm will essentially accept a solution only if the new solution has a better objective function value. Therefore, if the algorithm has already arrived in the catchment basin of a (global or local) minimizer, it will eventually converge to that minimizer.

This method has been proved quite useful in solving hard problems in combinatorial optimization. However, we observe that the simulated annealing algorithm as stated above suffers from certain drawbacks as described in the following scenario: After a large number of iterations (the temperature has already been very low), the algorithm arrives at a strictly local minimizer. Unfortunately, this local minimizer is not a global minimizer. A small perturbation of this local minimizer produces a new solution still in the catchment of this local minimizer. Since the local minimizer is a strict one and that the temperature is low, this move will be rejected. The algorithm tries many times to move in vain, and finally stops at this non-global solution!

In certain optimization problems, it might be relatively cheap to perform a local minimization from any given feasible point. These are the problems which we are interested in. We will propose a two-level simulated annealing algorithm for solving this kind of optimization problems. It will be seen that the new algorithm is less

likely to get stuck in the above scenario than conventional simulated annealing algorithms.

Suppose that we are given problem (P) and a cheap local minimization algorithm. For any given feasible point $x \in \mathcal{D}$, define $\underline{x}$ to be the local minimizer which the given local minimization algorithm will lead to from x. Suppose that a small perturbation from x produces y and that we want to determine whether to accept the move or reject it. The conventional simulated annealing algorithm will compare $f(x)$ and $f(y)$ to make this decision. We think that it is is more meaningful to compare $f(\underline{x})$ and $f(\underline{y})$ in order to determine whether to accept or reject the move, because we are interested in finding local minimizers and therefore a point lying on a hill is not of our interest. This idea leads to the following two-level simulated annealing algorithm.

Algorithm_2: Two-Level Simulated Annealing
Set T to a positive number (initial temperature). Set x_{old} to an initial feasible solution and compute $\underline{x_{old}}$. Set $f_{old} := f(\underline{x_{old}})$; $x_{best} := x_{old}$; and $f_{best} := f_{old}$.
repeat
  for i := 1 to *number_of_iterations*;
    $x_{new} := perturbation(x_{old})$;
    $f_{new} := f(\underline{x_{new}})$;
    generate a random number $rand \in (0,\ 1)$;
    if $((f_{new} < f_{old})$ or $(rand < exp((f_{old} - f_{new})/T)))$ then
      $x_{old} := x_{new}$; $f_{old} := f_{new}$;
      if $(f_{new} < f_{best})$ then
        $x_{best} := x_{new}$; $f_{best} := f_{new}$;
      endif
    endif
  endfor
  $T := \alpha T$;
until (stopping criterion is met)
Output $\underline{x_{best}}$ and f_{best} as the best known solution and objective function value.

Figure 2. **Two-level simulated annealing algorithm**

Our two-level simulated annealing algorithm differs from a conventional simulated annealing algorithm in that it operates on two sequences of iterative points $\{x_k\}$ and $\{\underline{x_k}\}$. We call $\{x_k\}$ the upper level and $\{\underline{x_k}\}$ the lower level for the simple reason that the objective function value of $\underline{x_k}$ is lower than or equal to that of x_k. The perturbation (move) is made on the upper level while the decision of accepting or rejecting the move is based on the comparison of the objective function value on the lower level. Therefore the algorithm is named *two-level simulated annealing.*

Why is a two-level simulated annealing algorithm necessary? Suppose that a small perturbation of x produces y. $f(y)$ may be greater than, equal to, or less than $f(x)$. This information is useful, but not enough. Even more useful information is the answer to the following question: Starting from y, will a local minimization

algorithm arrive at a better (lower) local minimizer than $\underline{x}$, a same local minimizer as $\underline{x}$ or a different local minimizer with same objective function value as $\underline{x}$, or a worse local minimizer than $\underline{x}$? The two-level simulated annealing algorithm looks ahead for the latter information before making any decision. In the first two cases, the two-level simulated annealing algorithm will accept the move. In the third case, it will accept the move conditionally, depending on the random number *rand*, the temperature T, and the difference in $f(\underline{x})$ and $f(\underline{y})$. When the temperature T gets very low, the two-level simulated annealing will still accept moves which lead to worse function values, but these moves essentially all lead to better (or same) local minimizers. In other words, the two-level simulated annealing algorithm can easily climb up the hill of a catchment basin of a minimizer at any temperature, but is very careful in moving into the catchment basin of a worse local minimizer when the temperature T is low.

3. Parallel Simulated Annealing

How to implement a (two-level) simulated annealing algorithm efficiently on a given parallel machine?

The easiest way to implement a simulated annealing algorithm on a parallel machine is to parallelize the function evaluation phase of a sequential algorithm. In this case, the parallel algorithm will be the same as the sequential algorithm, except that the function evaluation phase is speeded up. This kind of implementation is not efficient unless the function evaluation is the most significant part of the algorithm and that the function evaluation can be parallelized efficiently.

A second kind of parallel simulated annealing algorithms has been proposed in [7]. In [7], the authors present a method for parallelizing the simulated annealing algorithm by mapping the algorithm onto a dynamically structured tree of processors. They have studied the SA Decision Tree and designed three parallel simulated annealing algorithms, namely the Static PSA, the Dynamic Balanced PSA, and the Dynamic Unbalanced PSA. However, the speedup one can hope in the worst case for the Static PSA on a P processor machine is $\log_2 P$. For the other two PSA's, various assumptions are required to guarantee a reasonable speedup. There are many versions of parallel simulated annealing algorithms. We will not try to mention all of them here. In what follows, we will present a parallel (two-level) simulated annealing similar to the one proposed in [10].

We will present our parallel (two-level) simulated annealing algorithm on a EREW MIMD multiprocessors (or tightly coupled machines). This algorithm will be implemented in a master-slave mode on the Thinking Machines CM-5, a SPMD supercomputer (which can be think of as a loosely coupled machines), to solve the Molecular Conformation problem. For terminologies of parallel computers, readers are referred to [1].

The parallel algorithm that we are going to describe is a nondeterministic algorithm in the sense that the parallel algorithm is doing different work than the sequential algorithm and that different runs of the program on the same parallel

computer will perform slightly different work (especially in a timesharing mode like the CM-5). By no means, we claim that determinacy is not important. We do think, however, that our parallel algorithm is efficient and that it follows the philosophy of (two-level) simulated annealing.

In the following description of our parallel algorithm, T, x_{old}, x_{best}, f_{old}, f_{best}, i, and *done* are global variables. All other variables are local variables, although each processor also has a local variable with the same name for each of the global variables.

Algorithm_3: Parallel Two-Level Simulated Annealing

{Initialize global variables}
Set T to a positive number (initial temperature). Set x_{old} to a initial feasible solution and compute $\underline{x_{old}}$. Set $f_{old} := f(x_{old})$, $x_{best} := x_{old}$, $f_{best} := f_{old}$, $i := 1$, $done := FALSE$.
repeat_in_parallel
    $lock(done,\ i,\ T,\ f_{old},\ x_{old})$;
    $read(done,\ i,\ T,\ f_{old},\ x_{old})$;
    $unlock(done,\ i,\ T,\ f_{old},\ x_{old})$;
    if $(done = TRUE)$ exit repeat_in_parallel loop
    if $(mod(i, number\_of\_iterations) = 0)$ then
        $lock(T)$; $T := \alpha T$; $unlock(T)$;
    endif
    $lock(i)$; $i := i + 1$; $unlock(i)$;
    $x_{new} := perturbation(x_{old})$; $f_{new} := f(x_{new})$;
    generate a random number $rand \in (0,\ 1)$;
    if $((f_{new} < f_{old})$ or $(rand < exp((f_{old} - f_{new})/T)))$ then
        $lock(x_{old})$; $x_{old} := x_{new}$; $unlock(x_{old})$;
        $lock(f_{old})$; $f_{old} := f_{new}$; $unlock(f_{old})$;
        $lock(f_{best})$; $read(f_{best})$; $unlock(f_{best})$;
        if $(f_{new} < f_{best})$ then
            $lock(x_{best})$; $x_{best} := x_{new}$; $unlock(x_{best})$;
            $lock(f_{best})$; $f_{best} := f_{new}$; $unlock(f_{best})$;
        endif
    endif
    if (stopping criterion is met) then
        $lock(done)$; $done := TRUE$; $unlock(done)$;
    endif
endrepeat
Output x_{best} and f_{best} as the best known solution and objective function value.

Figure 3. **Parallel two-level simulated annealing**

It should be clear from the description of the algorithm that two runs of the same algorithm on same input problem and same parallel machine may or may not produce the same iterative sequence, depending on whether or not all of the processors work at the same speed in the two runs. Therefore, in a timesharing mode, the iterative sequence obtained from two different runs may be different. However, the parallel algorithm still follows the move/evaluate/decide idea followed by the sequential (two-level) simulated annealing algorithm. It tries to move from one iteration point to another, accepting a better solution, and accepting a worse solution with some probability. The parallel algorithm in [7] works the same as the sequential algorithm, but at a cost of a lot of wasted computation time. In our parallel algorithm, all of the computation provides useful information. There is no waste of computer time except the communication time if it is implemented on a loosely coupled machine.

4. Molecular Conformation on the CM-5

The minimization of potential energy functions of clusters of atoms is known as the molecular conformation problem. The global minima of potential energy functions are of great interests to researchers in chemistry, biology, physics, and optimization. One of the fundamental problems in molecular conformation is the minimization of the pure Lennard-Jones potential function [12]. Even this problem has been proven to be very hard. Hoare has claimed that the number of local minimizers of a cluster of n atoms grows as fast as the function $O(e^{n^2})$. Nontheless, many papers have been published on computational methods [4, 5, 11, 13, 14, 15, 18, 25, 27, 28, 29, 32, 33, 34] and putative global minima for cluster sizes as large as $n = 150$ have been reported [14, 25, 29].

The most successful algorithm for minimizing Lennard-Jones clusters has been Northby's algorithm which first finds a set of lattice local minima and then relaxes those lattice minima by continuous minimization. With this algorithm, Northby is able to publish putative global minima for cluster sizes ranging from 13 to 150 [25]. In Northby's algorithm, the lattice search part is a discrete optimization problem (actually a combinatorial one). Therefore in [22], we call this algorithm a Discrete-Continuous algorithm. So far, the most computing intensive part in Northby's algorithm is in the discrete optimization. And the most computing intensive part in Northby's lattice search algorithm is in the search for lattice local minima where it pivots from one configuration to another with a better function value. We call each of these pivots a *move* (or just a pivot). In Northby's implementation [25], each move takes $O(n^{\frac{5}{3}})$ for a cluster with n atoms. In [22], we have reduced the time complexity of each move to $O(n^{\frac{4}{3}})$ for a cluster with n atoms. Therefore, with the supercomputer CM-5, we have been able to get computational results for n as large as 1000.

It should be noted that in [5, 6] general purpose global optimization algorithms have been proposed which can, without knowledge on the lattice structure, find

minimizers as good as the ones reported by Northby for the Lennard-Jones clusters of size in the range $n \leq 147$, with only a few exceptions, where minimizers almost as good as the ones reported by [25] are found. To the best of our knowledge, these are the most successful applications of general purpose global optimization algorithms on the Lennard-Jones clusters.

In this paper, we further reduce the time complexity of each move in Northby's lattice to $O(n^{\frac{2}{3}})$, apply our two-level simulated annealing algorithm to the lattice search problem, and implement the algorithm on Thinking Machines CM-5. Because of the reduced time complexity of the pivot algorithm and the efficient implementation of the two-level simulated annealing algorithm, we are able to get satisfactory results for the discrete minimization problem for $n = 100,000$ on the CM-5 in a relatively short time. These lattice minimizers are then relaxed to obtain ground states for the Lennard-Jones clusters. Particularly, for $n \leq 1000$, we have found lower energies than the ones reported in [22].

4.1. Lennard-Jones Potential and the *IC* and *FC* Lattices

Given n atoms (points), $p_1, p_2, \cdots, p_n$, in 3 dimensional Euclidean space, the total 2-body potential energy function is defined as

$$V_n(p) = \sum_{j=2}^{n} \sum_{i=1}^{j-1} v(||p_j - p_i||_2) \ , \tag{1}$$

where $v(r)$ is the Lennard-Jones potential function ([12]) defined as

$$v(r) = \frac{1}{r^{12}} - \frac{2}{r^6} \ . \tag{2}$$

The problem is to find a configuration (positions for the n points) such that the total potential energy function $V_n(p)$ is minimized.

For each pair, the Lennard-Jones potential function $v(r) = r^{-12} - 2r^{-6}$ is plotted in Figure 4. It has only one local minimizer at $r = 1$ (which is also the global one) with function value -1. As r approaches 0, $v(r)$ approaches $+\infty$. As r approaches $+\infty$, $v(r)$ saturates to 0. Note that the function $v(r)$ is a unimodal nonconvex function.

Finding a global minimizer of $V_n(p)$ is extremely difficult except for very small cluster sizes. The difficulty is due to the fact that while it is always possible with a supercomputer and a local minimization algorithm (e.g. quasi-Newton method) to relax any initial configuration to some local minimizer, unless the starting configuration is in the catchment basin of the global minimizer, the minimizer found may not be the global minimizer. Hoare has shown that the number of local minima in the potential energy surface of an n-atom Lennard-Jones cluster is about $O(e^{n^2})$. Thus, it is impractical to perform an undirected search for all local minima of the potential function in order to find the global minimizer, except for very small clusters.

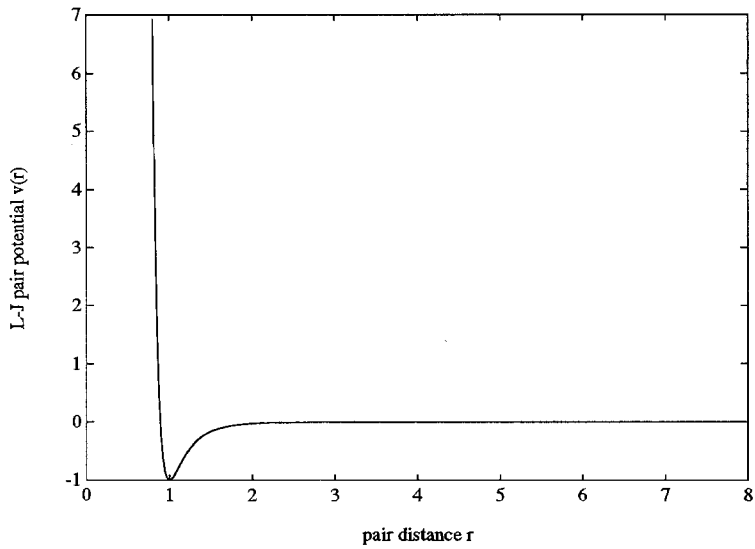

Figure 4. **Lennard-Jones pair potential function**

Very often, one can better solve a problem if he/she has some physical insight into the problem. Here again, it is the case. Chemical physicists have learned from previous research that the "ground states" of Lennard-Jones clusters exhibit certain kind of lattice structures. So far, the most successful algorithms for computing ground states of Lennard-Jones clusters are based on lattice search followed by local minimization from the lattice minima, represented by the Northby algorithm [25]. As stated in [22], a critical assumption for lattice search based algorithms is that *a well-defined set of lattice structures contains at least one initial cluster configuration which relaxes to the ground state.* As described and supported by computational results in [25], the IC and FC lattices to be described below are well-defined lattice structures for the pure Lennard-Jones clusters. We believe that in *most of the cases*, the relaxation of a global lattice minimizer will result a configuration with a lower energy than the relaxation of a non-global lattice local minimizer.

The icosahedral lattice [8, 14, 25] introduced by Mackay can be described as 20 slightly flattened tetrahedrally shaped fcc units with 12 vertices on a sphere centered at the origin. The ratio between the interatomic spacing in the 20 equilateral outer faces and the radial lines connecting the 12 vertices with the origin is $\sqrt{\frac{2}{1+\cos(\frac{\pi}{5})}}$, which is approximately 1.05146.

For the IC lattice, the total number of lattice on each layer is 1, 12, 42, 92, $\cdots$, $10i^2+2$, $\cdots$. Therefore the number of lattice points in the sequence of closed shell IC lattice is 1, 13, 55, 147, $\cdots$, $1+(10i^3+15i^2+11i)/3$, $\cdots$.

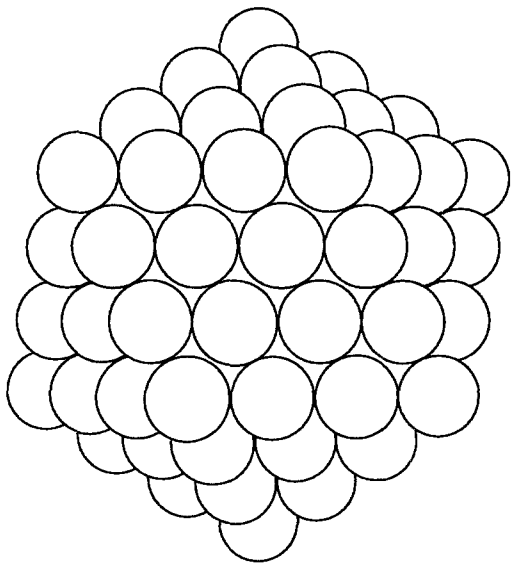

Figure 5. **An *IC* lattice with 147 lattice points**

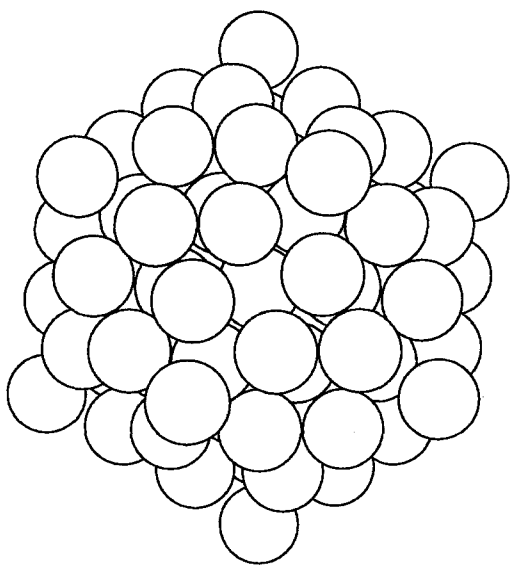

Figure 6. **An *FC* lattice with 127 lattice points**

The FC lattice consists of a smaller IC lattice enclosed by a layer of *stacking fault* icosahedral shell. This shell has 12 vertices and 20 facets as described above. However, it has fewer filling lattice points on each facet. These lattice points are located at the stacking fault positions of the IC lattice shell. The number of lattice points on the outer layer of an FC lattice is 1, 12, 32, 72, 132, $\cdots$, $10i(i-1)+12$, $\cdots$. Therefore the number of lattice points in the sequence of closed shell FC lattice is 1, 13, 45, 127, $\cdots$, $11+(10i^3+15i^2-19i)/3$, $\cdots$.

Figure 1 of [25] best describes how each of the facets are filled with other lattice points for both the IC shell and the FC shell. An IC lattice with 147 points is illustrated in Figure 5 and an FC lattice with 127 points is illustrated in Figure 6. [22] also describes how to generate these lattices.

4.2. Pivoting on the Lattice

In this subsection, we describe Northby's pivot algorithm for finding a lattice local minimizer. We will also introduce a simple data structure which reduces the time complexity of a single pivot from $O(n^{\frac{5}{3}})$ to $O(n^{\frac{2}{3}})$.

Suppose that we want to find a lattice local minimizer of an n-atom cluster. Let us assume that we have chosen one of the two types of lattices for the lattice minimization. First, find the largest IC lattice which contains fewer than n points (if one of the IC lattices has exactly n points, we simply put all n atoms on the lattice points of that lattice and quit). Call this IC lattice the *core* and let N_{core} and I_{core} be the number of points in this IC lattice and the index set of this IC lattice, respectively. Next, find the next layer of IC (or FC) lattice which contains N_{surf} (surface) points. Let I_{surf} be its index set. If $N_{core}+N_{surf}=n$, we simply put all n atoms on the lattice points of the lattice and quit.

An initial configuration can be constructed by filling N_{core} atoms into the core lattice and randomly put the remaining $(n-N_{core})$ atoms onto the N_{surf} surface lattice sites. This is equivalent to partitioning the index set I_{surf} to two subsets I_{surf}^{filled} and I_{surf}^{vacant} such that $|I_{surf}^{filled}|=n-N_{core}$ and that site $i\in I_{surf}$ is filled with an atom if and only if $i\in I_{surf}^{filled}$.

Northby [25] computes the interaction matrix $VP(i,j)$, the pair interaction between an atom on site i and one on site j at the very beginning of the algorithm and stores it as a lookup table. After this is done, Northby's pivot algorithm for finding a lattice local minimizer can then be summarized as follows.

Algorithm_4: Northby's Pivot Algorithm for Lattice Minimization

1. {*Find the most loosely bound atom*}
   Find $i_{loose}\in I_{surf}^{filled}$ such that

$$i_{loose} = arg \max_{i \in I_{surf}^{filled}} \{ \sum_{j \in I_{core}} VP(i,j) + \sum_{\substack{j \in I_{surf}^{filled} \\ j \neq i}} VP(i,j) \}. \quad (3)$$

Site i_{loose} is called the most loosely bound filled site and the atom at that site is called the most loosely bound atom. Let $gain_{loose}$ be the maximum function value that the maximization problem in (3) achieves at i_{loose}. Apparently, this is the total contribution that the atom at site i_{loose} has towards the total potential energy.

2. {*Find the most tightly binding vacant site*}
Find $i_{tight} \in I_{surf}^{vacant}$ such that

$$i_{tight} = arg \min_{i \in I_{surf}^{vacant}} \{ \sum_{j \in I_{core}} VP(i,j) + \sum_{\substack{j \in I_{surf}^{filled} \\ j \neq i_{loose}}} VP(i,j) \}. \quad (4)$$

Site i_{tight} is called the most tightly binding vacant site. Let $gain_{tight}$ be the minimum function value that the minimization problem in (4) achieves at i_{tight}. This is the new contribution that the atom at site i_{loose} has towards the total potential energy when moved to site i_{tight}.

3. {*Pivot on the Lattice*}
If $gain_{tight} - gain_{loose} < 0$ then move the atom at site i_{loose} to site i_{tight} and goto step 2. Otherwise, the current configuration is a lattice local minimizer.

Figure 7. **Northby's pivot algorithm**

Moving an atom from one lattice site to another is called a *move* (or just a *pivot*) in the Northby algorithm. Each time an atom is added or removed from a site the program recalculates from VP the total potential V, and the energy change $DV(i)$ associated with adding or removing another atom at each site. Although Northby does not specify clearly the time complexity of each move, we can easily deduce from the above description that the time complexity for choosing the most loosely bound atom is $O(n(n - N_{core}))$ and the time complexity for choosing the most tightly binding vacant site is $O(n(N_{surf} - (n - N_{core})))$. Therefore the time complexity for each move is $O(nN_{surf})$. Since $N_{surf} = O(n^{\frac{2}{3}})$ and that $N_{core} < n < N_{core} + N_{surf}$, the time complexity of each move in Northby's implementation is $O(n^{\frac{5}{3}})$. It should also be noted that in Northby's implementation, $O(n^2)$ storage is required to store the interaction matrix VP.

In [22], we have carefully studied Northby's algorithm. We note that the interaction matrix VP only speeds up the computation by a constant factor (of about 4) at

the cost of $O(n^2)$ storage. Therefore, we have dropped the interaction matrix. Instead, we have introduced a double precision array $CORE(0 : N_{surf})$ which stores the value $\sum_{i,j \in I_{core}, i \neq j} VP(i,j)$ in $CORE(0)$ and the values $\sum_{j \in I_{core}} VP(i,j)$ in $CORE(i)$ for each $i \in I_{surf}$. With the aid of this simple data structure, we have reduced the time complexity of each move to $O(n^{\frac{4}{3}})$.

In this paper, we further reduce the time complexity of each move to $O(n^{\frac{2}{3}})$ by introducing a new $O(n^{\frac{2}{3}})$ storage data structure.

This new data structure is a double precision array $SURF(N_{surf})$. Given an initial configuration represented by I_{surf}^{filled}, the array $SURF$ is initialized in $O(n^{\frac{4}{3}})$ time so that $SURF(i) = \sum_{j \in I_{surf}^{filled}, j \neq i} VP(i,j)$ for each $i \in I_{surf}$. The value $CORE(i) + SURF(i)$ is the contribution of the atom which is placed at the ith surface lattice point(if $i \in I_{surf}^{filled}$) or the amount that will be added to the total potential energy if a new atom is to be placed at site i of the surface lattice(if $i \in I_{surf}^{vacant}$). After this initialization is done, the most loosely bound atom can be found in $O(n^{\frac{2}{3}})$ time; if we delete i_{loose} from I_{surf}^{filled} and put it in I_{surf}^{vacant}, it requires $O(n^{\frac{2}{3}})$ time to update the array $SURF$; then the most tightly binding vacant site can be computed in $O(n^{\frac{2}{3}})$ time; to insert i_{tight} into I_{surf}^{filled} and update the array $SURF$ again takes $O(n^{\frac{2}{3}})$ time. Therefore the time complexity per move is reduced to $O(n^{\frac{2}{3}})$. Our new pivot algorithm can now be described as follows.

Algorithm_5: Modified Northby Pivot Algorithm

1. {*Find the most loosely bound atom*}
   Find $i_{loose} \in I_{surf}^{filled}$ such that

$$i_{loose} = arg \max_{i \in I_{surf}^{filled}} \{CORE(i) + SURF(i)\}. \tag{5}$$

   Let $gain_{loose}$ be the maximum function value that the maximization problem in (5) achieves at i_{loose}.

2. {*pick up the most loosely bound atom*}
   Drop i_{loose} from I_{surf}^{filled} and insert it into I_{surf}^{vacant}. Update the array $SURF$ in the following way: For each $i \in I_{surf}$ and $i \neq i_{loose}$, decrease $SURF(i)$ by $v(r_{i_{loose},i})$, where $v(\bullet)$ is the Lennard-Jones pair potential and $r_{i_{loose},i}$ is the Euclidean distance between site i_{loose} and site i on the surface lattice.

3. {*Find the most tightly binding vacant site*}
   Find $i_{tight} \in I_{surf}^{filled}$ such that

$$i_{tight} = arg \min_{i \in I_{surf}^{vacant}} \{CORE(i) + SURF(i)\}. \tag{6}$$

   Let $gain_{tight}$ be the minimum function value that the minimization problem in (6) achieves at i_{tight}.

4. {*put the atom at the most tightly binding vacant site*}
   Drop i_{tight} from I_{surf}^{vacant} and insert it into I_{surf}^{filled}. Update the array $SURF$ in the following way: For each $i \in I_{surf}$ and $i \neq i_{tight}$, increase $SURF(i)$ by $v(r_{i_{tight},i})$.

5. {*Check for stopping rule*}
   If $gain_{tight} - gain_{loose} \geq 0$ then stop, this is a lattice local minimizer; otherwise, goto step 1.

Figure 8. **Modified Northby's pivot algorithm**

4.3. Lattice Search by Two-Level Simulated Annealing

In the previous subsection, we have made the pivot very cheap and thus finding a lattice local minimizer has also been made less costly. Therefore, the lattice search problem in Molecular Conformation is an ideal application of our two-level simulated annealing algorithm which has been introduced in section 2. The algorithm is described as follows.

Algorithm_6: Lattice Search by Two-Level Simulated Annealing

1. {*Initialization*}
   Find the largest IC lattice which contains at most n points and call this the *core* lattice. Let I_{core} be the index set of the core and define $N_{core} = |I_{core}|$. If $N_{core} = n$, put the n atoms on the core lattice and stop.

   Find the next IC or FC lattice shell (depending on the lattice we are using), let I_{surf} be its index set and define $N_{surf} = |I_{surf}|$. Define $N = N_{core} + N_{surf}$ as the total number of points in the lattice. If $N = n$, put the n atoms on the N lattice points and stop.

   Fill the N_{core} sites in the core with N_{core} atoms. Assign the remaining $n - N_{core}$ atoms randomly onto the surface sites. This assignment partitions the index set I_{surf} into two subsets I_{surf}^{filled} and I_{surf}^{vacant} which corresponds to the filled surface sites and the vacant surface sites, respectively.

   In $O(n^2)$ time, compute the values of the array $CORE$ In $O(n^{\frac{4}{3}})$ time, compute the values of the array $SURF$. Define the 0-1 array x_{old} with index set I_{surf} according to the above partition: $x_{old}(i) = 1$ if and only $i \in I_{surf}^{filled}$. Define $f(x_{old})$ to be the total potential energy function of the cluster resulted by filling N_{core} atoms in the core sites and the rest $n - N_{core}$ atoms in the surface sites determined by x_{old}.

Set $T = 10$, $\alpha = 0.5$, $x_{best} = x_{old}$. Apply Algorithm_5 to find the lattice local minimizer $\underline{x_{old}}$ starting from x_{old}. Set $f_{old} = f(\underline{x_{old}})$. Set $f_{best} = f_{old}$.

2. {*Two-Level Simulated Annealing*}
repeat
for $i := 1$ to 1530 {number_of_iterations}
$x_{new} := perturbation(x_{old})$;
Compute $\underline{x_{new}}$ and $f_{new} := f(\underline{x_{new}})$;
generate a random number $rand \in (0,\ 1)$;
if $((f_{new} < f_{old})$ or $(rand < exp((f_{old} - f_{new})/T)))$ then
$x_{old} := x_{new}$; $f_{old} := f_{new}$;
if $(f_{new} < f_{best})$ then
$x_{best} := x_{new}$, $f_{best} := f_{new}$;
endif
endif
endfor
$T := \alpha T$;
until (stopping criterion is met)
Output $\underline{x_{best}}$ and f_{best} as the best known solution and objective function value.

Figure 9. **Lattice search by two-level simulated annealing**

The stopping criterion we have used here is that the best known function value has not been improved for two consecutive for-loops. Of course, other stopping criterion can also be used here.

We would like to point out here that the value $CORE(0)$ is only necessary if we want to get the total potential energy while pivoting. Without it, the algorithm works correctly to locate the (local or global) lattice minimizers.

4.4. Implementation on the CM-5

The lattice search by two-level simulated annealing algorithm described in the previous subsection has been implemented on the Thinking Machines CM-5.

The CM-5 extends TMC's existing Data Parallel programming model from Single Instruction-Multiple Data (SIMD) to Single Program-Multiple Data (SPMD). It can support both a highly synchronized Data Parallel/SIMD paradigm and a message passing paradigm which a MIMD machine would provides.

The CM-5 allows a control processor (CP) to control a large number of processing elements (PE's) by down loading to each PE an identical copy of the same program. The PE's then either execute the same code in a SIMD mode, or take different branches in that code, thus effectively emulating MIMD.

Interprocessor communications are supported through two networks: the Data Network (DN) and the Control Network (CN). The message passing library (CMMD) is used to control and coordinate program streams running on different PE's. For

more detailed information about the CM-5, readers are referred to the Thinking Machines publications [30, 31].

The machine that we have used at the Army High Performance Computing Research Center/Minnesota Supercomputer Center is a 544 PE machine. The system can be configured into two or three partitions of either 32 and 512 PE's, or 32, 256, 256 PE's. The PE's are addressed from 0 to 31, or 255, or 511, depending on the different partitions. Each PE is a 33 MHz SPARC-2 chip with 16 MB local memory. Interprocessor communication has a bandwidth of 20 MB/sec within a group of 4 nearest neighbors (e.g., PE's 0, 1, 2, 3, or PE's 4, 5, 6, 7, etc.), 10 MB/sec within a group of 16 of second nearest neighbors (e.g., PE's 0-15, or PE's 16-32, etc.), and 5 MB/sec between any two PE's on the system. The machine is running under CMOST 7.1.1. In the future, each PE will be upgraded to 32-MB of memory and the SPARC will be augmented with 4 vector units.

Our two-level simulated annealing algorithm has been implemented on the CM-5 in a master-slave mode. One PE (number 0 in this case) serves as the master PE, and all other PE's serve as slave PE's. The master PE is used to emulate the sheared memory assumed in the description of Algorithm_3. Each slave PE asks for a job from the master, performs a perturbation, lattice search, and decides whether to accept or reject the new solution. It then sends the computation results back to the master and asks for a new job until it is told to stop. The master PE checks incoming messages from any node. sends out new jobs to and receives computation results from the slaves, Whenever a new solution is accepted by the corresponding slave, the master also accepts that solution and makes it the new current solution. The ten (or fewer) best solutions are stored in the master node. If stopping criterion is met, it signals the slaves to stop and sends the computed results to the host.

This turns out to be very efficient both in solving the problem and in achieving a good flop rate on the machine. In particular, we have got satisfactory results for the lattice minimization problem for cluster sizes as large as 100,000 and achieved a 0.8 giga flop/sec in double precision operations which is about one third of the theoretical peek performance of the machine as it is. Computational results are presented in the next section.

5. Computational Results

Computational results on lattice minimization are obtained on the CM-5 at the AHPCRC/MSC operated under the CMOST 7.1.1 operating system. The programs are written in F77 and employ the CMMD message passing library. The results are presented in three different ranges of cluster sizes: 100 – 1000, 1000 – 10000, and 10000 – 100000. The lattice minima in the first two cluster ranges are then relaxed on the Cray-XMP supercomputer at the AHPCRC/MSC using a Minpack2 subroutine: the Limited Memory BFGS code [21]. Since our main interested here is in the lattice minimization, timing results are reported only for the CM-5.

Lattice minimization results for cluster ranges in [10000, 100000] are obtained on the 512 partition in dedicated mode. These results are presented in Table 1.

Table 1. **Computational results for cluster size in [10000, 100000]**

n	$Latt$	$func_{lat}$	$nmoves$	$seconds$	$mflop/sec$
10000	IC	-72803.2969	2105762	90.5828	692
20000	IC	-147903.7031	11764754	815.4856	704
30000	IC	-223614.1094	8762607	842.2309	690
40000	IC	-299660.7188	14371260	1615.5211	708
50000	IC	-376019.0312	6472805	743.7345	818
60000	IC	-452480.1250	14383802	2162.8548	676
70000	FC	-529128.5000	8425552	1424.6277	674
80000	FC	-605586.3125	14027870	2473.0903	692
90000	IC	-682341.1875	14713350	3471.4590	573
100000	FC	-759190.1250	21269830	4979.7554	597

Note that the lowest lattice potential energy values are obtained on the IC lattice except for cluster sizes 70000, 80000, and 100000, where the lowest lattice potential energy values are obtained on the FC lattice. Column 3 of the table reports the lowest lattice potential energy values computed. Column 4 reports the total number of moves (pivots) required by the algorithm. Column 5 reports the maximum elapsed seconds on the master PE. Column 6 reports the mega flops (in 64-bit operations) per second sustained in executing the program. Note that a rate of 818 mega flop per second is sustained on the 50000-cluster problem.

Table 2. **Computational results for cluster size in [100, 1000]**

n	$Latt$	$func_{lat}$	$func_{opt}$	$nmoves$	$seconds$	$mflop/sec$
100	IC	-522.2946	-557.0398	85114	9.22	14
200	IC	-1147.5015	-1229.1848	178713	10.06	46
300	IC	-1809.5105	-1942.1068	49346	14.07	10
400	IC	-2465.6689	-2650.4315	272438	14.81	73
500	IC	-3144.3364	-3382.6935	312777	17.26	72
600	FC	-3825.7654	-4119.2441	161639	17.85	45
700	IC	-4513.9102	-4862.3946	686757	14.57	264
800	IC	-5206.5977	-5609.7262	274074	2.44	628
900	IC	-5916.9136	-6377.4914	88209	11.90	44
1000	IC	-6604.6631	-7121.8967	240448	15.71	117

Table 3. **Computational results for cluster size in [1000, 10000]**

n	$Latt$	$func_{lat}$	$func_{opt}$	$nmoves$	$seconds$	$mflop/sec$
1000	IC	-6604.6631	-7121.8967	239510	17.72	103
2000	IC	-13741.7510	-14837.5681	204325	14.74	139
3000	IC	-20960.8281	-22652.0670	462310	15.85	446
4000	FC	-28267.5215	-30562.3114	1073377	56.52	323
5000	IC	-35650.5781	-38549.4018	328537	38.18	161
6000	IC	-42977.4766	-46475.4712	3106811	174.32	388
7000	IC	-50367.2109	-54479.7056	1501368	82.04	468
8000	IC	-57881.3906	-62611.9024	2048383	134.16	391
9000	IC	-65230.0547	-70567.5777	3268319	179.46	539
10000	IC	-72803.2969	-78773.5292	2105762	90.58	692

Computational results for the [100, 1000] and the [1000, 10000] ranges are presented in Tables 2 and 3. These results are obtained in a timesharing mode, with

variable number of users (from one to three) sharing the machine during the execution of the program. Therefore only the first 5 columns are important in these two tables. The last two columns are included for readers who are interested in the performance of the algorithm/machine in a time sharing environment. The 4th column in Table 2 and Table 3 reports the lowest potential energy function values obtained after the relaxation from the lattice minima. Note that for cluster sizes 700, 800, and 1000, we have obtained lower energy values than the ones reported in [22].

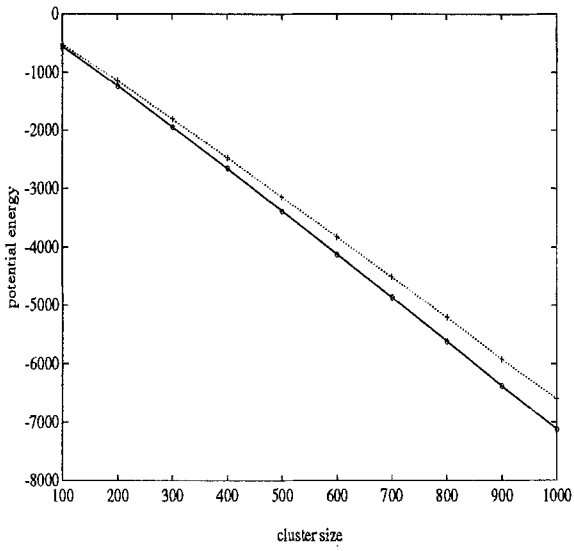

Figure 10. **Best known function value as a function of cluster size: 100-1000, dotted line plots function values before relaxation, solid line plots function values after relaxation**

The lowest potential energy function values for the different cluster size ranges are illustrated in Figures 10-12. The minima before relaxation (lattice minima) are plotted in dotted lines. The function values after the relaxation are plotted in solid lines. Although the dependence of the lowest energy values found on the cluster size looks like linear, we believe (from study of the tables) that the dependence is somewhat superlinear. This suggests that the minimum inter-atom distance could be very small in the ground state configuration if the cluster size becomes very large because otherwise the minimum energy will have a linear lower bound [35].

The configuration of the putative global minimizer for the 200-cluster is illustrated in Figure 13. The potential energy function value for this configuration is -1229.1848. Artificial inter-atom bonds between nearest neighbors are added to increase visibility.

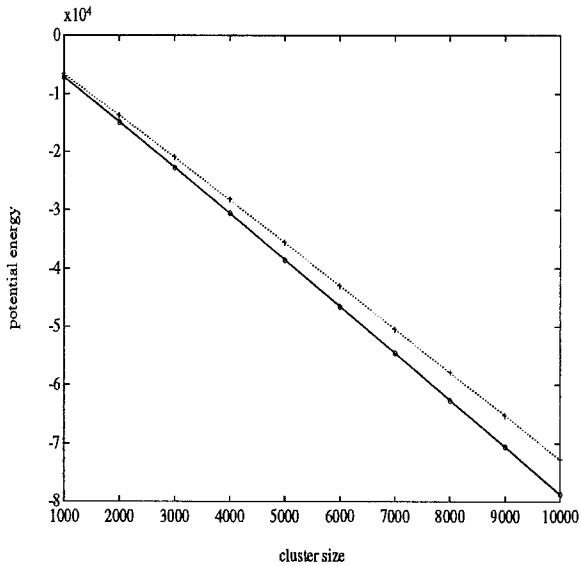

Figure 11. **Best known function value as a function of cluster size: 1000-10000, dotted line plots function values before relaxation, solid line plots function values after relaxation**

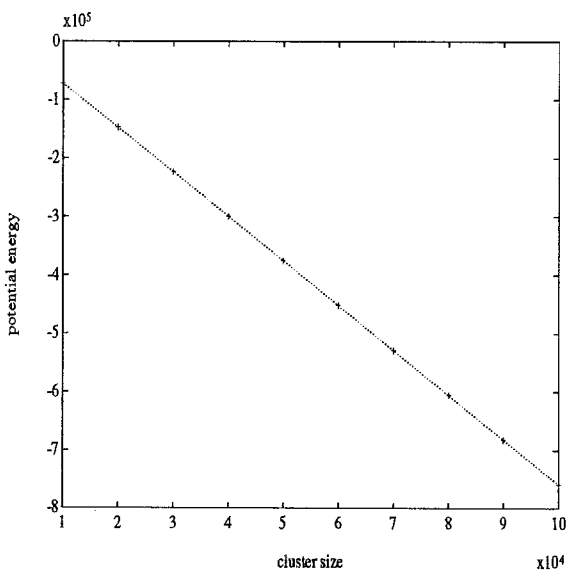

Figure 12. **Best known function value as a function of cluster size: 10000-100000, before relaxation**

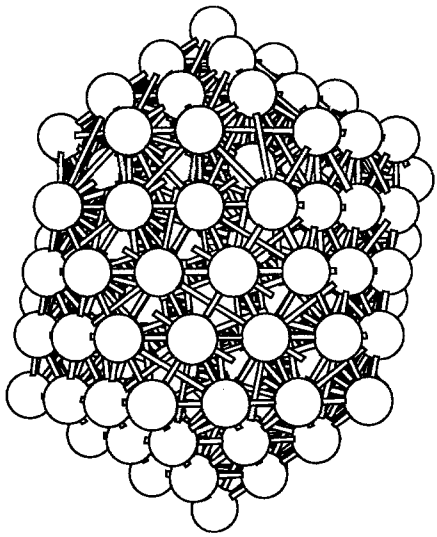

Figure 13. **Putative global minimizer for the 200-cluster**

6. Conclusions

In this paper, we have presented a new kind of simulated annealing algorithm – the two-level simulated annealing algorithm for solving certain class of hard combinatorial optimization problems. A parallel version of this algorithm has also been presented. These algorithms are then applied to the Molecular Conformation problem and implemented on the Thinking Machines CM-5. We have been able to get better results and get satisfactory results for much larger problems with our parallel two-level simulated annealing algorithm. We believe that our parallel two-level simulated annealing algorithm will find more and more applications in other models of Molecular Conformation problems and in hard combinatorial optimization problems from Operations Research and Computer Aided Design.

Acknowledgement

This research was supported in part by the Army Research Office contract number DAAL03-89-C-0038 with the University of Minnesota Army High Performance Computing Research Center. I am grateful to Dr. Jorge Moré from Argonne National Laboratory for introducing me to the wonderful field of Molecular Conformation. I would like to thank Dr. D.G. Vlachos for stimulating discussions and for giving me a copy of reference [25]. Thanks are due to Drs. Panos Pardalos, Juan Maza, Jill Mesirov, Gorge Wilcox, and David Ferguson for helpful discussions.

Finally, I would like to thank Drs. Gorge Sell and Don Austin for their consistent support and encouragement.

References

[1] S.G. Akl, *The Design and Analysis of Parallel Algorithms*, Prentice-Hall International, Inc., 1989.

[2] C.G.E. Boender and A.H.G. Rinooy Kan, Bayesian Stopping Rules for Multistart Global Optimization Methods, *Mathematical Programming*, Vol. 37(1987), pp. 59-80.

[3] R.J. Brouwer, P. Banerjee, A Parallel Simulated Annealing Algorithm for Channel Routing on a Hypercube Multiprocessor, *Proceedings of 1988 IEEE International Conference on Computer Design*, pp. 4-7.

[4] J.P. Brunet, A. Edelman, J.P. Mesirov, An Optimal Hypercube Direct N-body Solver on the Connection Machine, *Proceedings of Supercomputing'90*, pp. 748-752, IEEE Computer Society Press 1990.

[5] R.H. Byrd, E. Eskow, R.B. Schnabel, and S.L. Smith, Parallel Global Optimization: Numerical Methods, Dynamic Scheduling Methods, and Application to Molecular Configuration, *Technical Report CU-CS-553-91, University of Colorado at Boulder, Department of Computer Science, Boulder, CO.*, October 1991.

[6] R.H. Byrd, E. Eskow, R.B. Schnabel, Global Optimization Methods for Molecular Configuration Problems, Presented at the *Fourth SIAM Conference on Optimization*, May 11-13, 1992, Chicago, IL.

[7] R.D. Chamberlain, M.N. Edelman, M.A. Franklin, E.E. Witte, Simulated Annealing on a Multiprocessor, *Proceedings of 1988 IEEE International Conference on Computer Design*, pp. 540-544.

[8] J.H. Conway and N.J.A. Sloane, *Sphere Packings, Lattices and Groups*, Springer-Verlag, 1988.

[9] A. Corana, M. Marchesi, C. Martini, and S. Ridella, Minimizing Multimodal Functions of Continuous Variables with the "Simulated Annealing" Algorithm, *ACM Transactions on Mathematical Software*, Vol. 13(1987), pp. 262-280.

[10] F. Darema, S. Kirkpatrick, V.A. Norton, Parallel Techniques for Chip Placement by Simulated Annealing on Shared Memory Systems, *Proceedings of 1987 IEEE International Conference on Computer Design*, pp. 87-90.

[11] J. Farges, M.F. De Feraudy, B. Raoult and G. Torchet, Cluster Models Made of Double Icosahedron Units, *Surface Science*, Vol. 156(1985), pp. 370-378.

[12] I.Z. Fisher, *Statistical Theory of Liquids*, University of Chicago Press, 1964.

[13] D.G. Garrett, K.D. Kastella, D.M. Ferguson, New Results on Protein Folding from Simulated Annealing, submitted to *Journal of the American Chemistry Society*, 1992.

[14] M.R. Hoare, Structure and Dynamics of Simple Microclusters, *Advances in Chemical Physics*, Vol. 40(1979), pp. 49-135.

[15] J. Danna Honeycutt and Hans C. Andersen, Molecular Dynamics Study of Melting and Freezing of Small Lennard-Jones Clusters, *Journal of Physical Chemistry*, Vol. 91(1987), pp. 4950-4963.

[16] L. Ingber, Very Fast Simulated Reannealing: A Comparison, *Mathematical and Computer Modeling*, Vol. 12(1989), pp. 967-973.

[17] L. Ingber, Genetic Algorithms and Very Fast Simulated Reannealing: A Comparison, To appear in *Mathematical and Computer Modeling*, 1992.

[18] R.S. Judson, M.E. Colvin, J.C. Meza, A. Huffer, and D. Gutierrez, Do Intelligent Configuration Search Techniques Outperform Random Search for Large Molecules?, *Sandia Report SAND91-8740, Sandia National Laboratories, Center for Computational Engineering, Livermore, CA.*, December 1991.

[19] S. Kirkpatrick, C.D. Gelatt, Jr., M.P. Vecchi, Optimization by Simulated Annealing, *Science*, Vol. 220(1983), pp. 671-680.

[20] S. Kirkpatrick, Optimization by Simulated Annealing: Quantitative Studies, *Journal of Statistical Physics*, Vol. 34(1984), pp. 975-986.

[21] D.C. Liu and J. Nocedal, On the Limited Memory BFGS Method for Large Scale Optimization, *Mathematical Programming*, Vol. 45 (1989), pp. 503-528.

[22] R.S. Maier, J.B. Rosen, G.L. Xue, A Discrete-Continuous Algorithm for Molecular Energy Minimization, in *Proceedings of Supercomputing'92, Minneapolis, November 16-20, 1992*, pp. 778-786.

[23] N. Metropolis, A. Rosenbluth, A. Teller, E. Teller, Equation of Several State Calculations by Fast Computing Machines, *Journal of Chemical Physics*, Vol. 21(1953), pp. 1087-1892.

[24] S. Nahar, S. Sahni, E. Shragowitz, Experiments with Simulated Annealing, *22nd Design Automation Conference*, 1985, pp. 748-752.

[25] J.A. Northby, Structure and Binding of Lennard-Jones Clusters: $13 \leq n \leq 147$, *Journal of Chemical Physics*, Vol. 87(1987), pp. 6166-6178.

[26] C.P. RaviKumar, L.M. Patnaik, Parallel Placement by Simulated Annealing, *Proceedings of 1987 IEEE International Conference on Computer Design*, pp. 91-94.

[27] D.R. Ripoll, S.J. Thomas, A Parallel Monte Carlo Search Algorithm for the Conformational Analysis of Proteins, *Proceedings ACM/IEEE Supercomputing'90*, pp. 94-102.

[28] T. Schlick and M. Overton, A Powerful Truncated Newton Method for Potential Energy Minimization, *Journal of Computational Chemistry*, Vol. 8(1987), pp. 1025-1039.

[29] David Shalloway, Packet Annealing: A Deterministic Method for Global Minimization, Application to Molecular Conformation, *Recent Advances in Global Optimization*, C. Floudas and P. Pardalos, eds. Princeton University Press: Princeton, NJ, 1991.

[30] Thinking Machines Corporation, *CMMD Reference Manual*, Version 1.1, 1992.

[31] Thinking Machines Corporation, *CMMD User's Guide*, Version 1.1, 1992.

[32] D.G. Vlachos, L.D. Schmidt, and R. Aris, Structures of Small Metal Clusters: Phase Transitions and Isomerization, *Army High Performance Computing Research Center Preprint 91-69, University of Minnesota, Minneapolis*, 1991.

[33] D.G. Vlachos, L.D. Schmidt, and R. Aris, Structures of Small Metal Clusters: Low Temperature Behavior, *Army High Performance Computing Research Center Preprint 91-70, University of Minnesota, Minneapolis*, 1991.

[34] L.T. Wille, Minimum-Energy Configurations of Atomic Clusters: New Results Obtained by Simulated Annealing, *Chemical Physics Letters*, Vol. 133(1987), pp. 405-410.

[35] G.L. Xue, R.S. Maier, J.B. Rosen, Minimizing the Lennard-Jones Potential Function on a Massively Parallel Computer, in Proceedings of *1992 ACM International Conference on Supercomputing*, pp. 409-416, ACM Press, 1992.