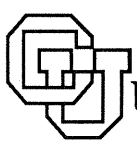
Parallel Global Optimization: Numerical Methods, Dynamic Scheduling Methods, and Application to Molecular Configuration

Richard H. Byrd
Elizabeth Eskow
Robert B. Schnabel
Sharon L. Smith

CU-CS-553-91 October 1991



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

The views, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

Abstract

Global optimization problems are computationally intensive problems that arise in many important applications. The solution of very large practical global optimization problems, which may have thousands of variables and huge numbers of local minimizers, is not yet possible. It will require efficient numerical algorithms that take advantage of the properties of the particular application, as well as efficient utilization of the fastest available computers, which will almost certainly be highly parallel machines. This paper summarizes our research efforts in this direction. First, we describe general purpose adaptive, asynchronous parallel stochastic global optimization methods that we have developed, and our computational experience with them. Second, we describe several alternative dynamic scheduling algorithms that are required to control such dynamic parallel algorithms on distributed memory multiprocessors, and compare their performance in the context of our parallel global optimization methods. Third, we discuss the application and refinement of these methods to global optimization problems arising from the structural optimization of chemical molecules, and present preliminary computational results on some problems with between 15 and 100 variables. This work includes the development of new algorithmic features that are motivated by the molecular configuration problem but are applicable to a wider class of large scale, partially separable global optimization problems.

1 Introduction

This paper summarizes the current status of a broad research program into parallel algorithms for the global optimization problem, and applications of these algorithms. The global optimization problem is to find the lowest minimizer of a nonlinear function that has multiple local minimizers. We denote the problem by

$$\min_{x \in D} f : R^n \rightarrow R \tag{1.1}$$

where D is some closed region in R^n. In this paper, we assume that the region D is specified by a set of upper and lower bounds on the variables x_i, and that the global minimizer is in the interior of D. That is, the problem is assumed to be essentially unconstrained with a region given that is known to contain the solution. We also assume that $f(x)$ is twice continuously differentiable, but do not assume any further properties of $f(x)$.

Global optimization problems arise in a variety of applications, including molecular chemistry and many uses of parameter estimation. In general they are very difficult and computationally expensive to solve. Partly for this reason, there has been a relatively small amount of research into global optimization methods. Advances in both optimization methods and computer technology are making the solution of practical global optimization problems more tractable, however, thus encouraging research in this area.

Of particular importance, it is becoming increasingly clear that the most powerful computers will be highly parallel computers, or networks of computers. Thus it is crucial that research into methods for computationally intensive problems, such as global optimization, includes consideration of the performance of these methods in highly parallel computational environments. As will be discussed in this paper, this involves issues in both numerical algorithms and computer science.

Our global optimization research has been based on a particular class of methods, *stochastic methods*. Overall, a large and diverse array of approaches have been investigated for solving the global optimization problem (1.1). First, many approaches exist for cases where $f(x)$ has special mathematical properties, such as concavity, but we are concerned with the general case where $f(x)$ can be any continuously differentiable nonlinear function. Approaches that have been developed for this general case include trajectory methods (e.g. [4, 1]), deflation and tunneling methods (e.g. [14, 25]), interval arithmetic methods (e.g. [16]), piecewise approximation methods (e.g. [33]), genetic algorithms (e.g. [19, 3]), simulating annealing methods (e.g. [24]), and stochastic methods (e.g. [28]). Stochastic methods, as we use the term, combine random sampling of the parameter space and the use of local minimization techniques. We have chosen this class of methods because their modern versions appear to be at least as efficient as other methods in the few comparisons that have been reported, because they are one of the few types of the methods that possess some mathematical (probabilistic) guarantee of success, and because they appear well suited for adaptation to highly parallel environments. The stochastic approach is described in the Section 2. An excellent survey of global optimization methods for general nonlinear functions is contained in [29].

Our research in parallel global optimization began with the development of a parallel version of existing stochastic methods. The limitations of these algorithms on both se-

quential and parallel computers led us to develop adaptive and asynchronous algorithms that appear to be more efficient in both sequential and parallel computing environments. The challenges regarding the efficient implementation of these adaptive parallel algorithms on highly parallel computers led to accompanying research into dynamic scheduling strategies for adaptive, asynchronous parallel algorithms, with the global optimization problem used as the primary example. Finally, we have recently begun to consider the application of stochastic methods to large scale global optimization problems, say problems with 20 to 1000 variables. This includes the development of new numerical algorithms. This research has been largely motivated by the molecular configuration problem.

The remainder of this paper summarizes the results to date of this ongoing research program. Section 2 discusses the research into parallel stochastic global optimization methods, focusing on the adaptive, asynchronous approach. Section 3 describes the work on scheduling strategies for dynamic parallel algorithms. Section 4 discusses the adaptation of parallel stochastic methods to solve large scale molecular configuration problems, including preliminary test results on some problems with 15 to 100 variables. More detailed descriptions of most of the work discussed in Sections 2 and 3 can be found in [6, 35, 36, 34].

2 Parallel Stochastic Global Optimization Methods

The parallel global optimization methods that we have developed derive from the stochastic methods of Rinnooy Kan and Timmer [28]. The basic framework of these stochastic methods is as follows. First they sample the objective function at a number of randomly chosen points in the feasible region (typically 100-1000 points per iteration). Next they select a subset of these sample points to be start points for local minimizations. Then they perform a local minimization algorithm from each start point. Finally they decide whether to terminate the algorithm, in which case the local minimizer with the lowest function value is considered to be the global minimizer, or they perform another iteration of this process, adding the new sample points to the previous ones. Rinnooy Kan and Timmer have proven that if such a procedure uses properly chosen rules for choosing start points, then with probability one it will find the global minimizer while doing only a finite amount of work.

Several aspects of the sequential method merit description in more detail. First, after conducting the random sampling, the methods generally discard all but a fixed percentage (say 20%) of the sample points, keeping only those points that have function values below a correspondingly selected "cutoff level". This refinment does not affect the theoretical properties of the method but significantly improves its computational performance. It is important to mention because it has important ramifications to the parallel methods. Second, the method for selecting start points for the local minimizations is crucial to the theoretical properties of the methods and also has important consequences to the parallel methods. Only the sample points below the cutoff level are considered in this phase. From them, a sample point is selected as a start point if it has a lower function value than all sample points within a prescribed "critical distance" from it. This critical distance is given by a formula related to the probabilistic analysis of the method, and is a monotonically decreasing function of the iteration number. Finally, the local

minimization method is a state-of-the-art algorithm; our implementations use the BFGS method from the UNCMIN package of Schnabel, Koontz, and Weiss [31].

Byrd, Dert, Rinnooy Kan, and Schnabel [6] developed a static, synchronous parallel version of this algorithm. It divides the domain space into P equal subregions, where P is the number of available processors. Each processor independently conducts sampling in the subregion it has been assigned, and then selects candidate start points from among its sample points, using the procedure described above but considering only the sample points in its subregion. The processors then synchronize and check, for any candidate start point within the critical distance of a subregion boundary, whether there is a lower sample point within the critical distance of it in some neighboring subregion. If so, the candidate start point is eliminated as a start point. Local minimizations are performed from the remaining start points. These start points may not be equally distributed among the subregions, so they are collected centrally and then distributed to the processors, which perform the local minimizations. If there are more start points than processors, initially one start point is distributed to each processor, and the remaining start points are distributed to processors as they finish their current local minimizations. When all the local minimizations have completed, the processors synchronize again in order to determine whether the stopping conditions have been satisfied and if not, the process is repeated. We refer to the methods as *static* because the method of subdividing the domain is fixed, and as *synchronous* because of the synchronization points at each iteration.

Byrd, Dert, Rinnooy Kan, and Schnabel [6] report test results of this parallel algorithm on a network of 4 or 8 computer workstations, using the standard set of very small test problems from [7]. While many of the speedup results were reasonably good, these experiments illuminated some important performance problems with the parallel approach. In particular, two aspects of the algorithm commonly led to load balancing problems, situations where some processors were idle while waiting for other processors to complete their tasks and reach a common synchronization point. The first was in the start point selection computation. The elimination of all sample points with function values above the cutoff level often left widely varying numbers of sample points in different subregions, causing the time for candidate start point selection to vary greatly between processors. This caused some processors to wait for others at the synchronization point that immediately followed this stage. Secondly, the local minimizations often had widely varying computation times, and the number of minimizations conducted often was too small to spread the load evenly among the processors. Consequently different processors often had widely differing amounts of work in the local minimization part of the computation.

The development and testing of the static, synchronous parallel algorithm also heightened our awareness of another problem with the underlying stochastic method that is independent of whether sequential or parallel computation is used. It is that the static algorithm can be inefficient for problems that have low local minimizers unevenly concentrated in the domain space. This is because the algorithm always does an equal amount of sampling in all areas of the domain space, even though there may appear to be a greater potential for finding the global minimizer in some portions of the domain than others. While this problem is common to sequential and parallel versions of the stochastic method, it was the consideration of decomposing the domain into subregions, which

only is done in the parallel version, that led us to consider alternatives.

To address both of these problems, namely lack of adaptivity by the algorithm to the problem being solved, and poor load balance in the parallel version, we have developed adaptive, asynchronous parallel (and sequential) stochastic global optimization methods. By *adaptive* methods we mean methods that react to the current state of the computation by attempting to focus computation on the part of the problem where this seems most fruitful. By *asynchronous* parallel algorithms we mean algorithms that allow each processor to proceed, as much as possible, independently of the other processors.

The general framework of our adaptive, asynchronous stochastic methods is given in Algorithm 2.1. There are two key distinctions between this algorithm and the previous sequential or parallel methods. First, rather than taking a global view of the computation, the algorithm is viewed as a set of separate subregion and minimization tasks that mainly operate independently of each other. This leads to a local, subregion-based view of the algorithm, and also leads naturally to an asynchronous parallel implementation. This change also requires that mechanisms be provided for scheduling these tasks and terminating the algorithm. Secondly, in step 3, each subregion determines adaptively at each iteration how much attention should be devoted to this subregion at the next iteration. The composition of these decisions forms the overall adaptive algorithm. In the next few paragraphs we describe some of the key aspects of these adaptive and asynchronous portions of the algorithm. More detailed discussions are given in [36] and [34].

Algorithm 2.1 – Framework of an Adaptive, Parallel Global Optimization Algorithm

Given $f : R^n \rightarrow R$, feasible region D, p processors

Partition D into $q \geq p$ subregions

For each subregion :

1. **Sampling :** Generate the coordinates of the new random sample points in the subregion, and evaluate $f(x)$ at each new sample point. Discard all sample points whose function value is below the global "cutoff level".

2. **Start Point Selection :** Select a subset of the sample points to be start points for local minimizations. (A sample point is selected to be a start point if it has the lowest function value of all sample points within the "critical distance" from it; special techniques are used for sample points near subregion boundaries.)

3. **Adaptive Decisions :** Decide whether to split this subregion into smaller subregions, what the new density of sample points for the subregion(s) should be, and the relative priority of continuing to process this subregion. Then apply this algorithm recursively to each of the new subregions as processors become available and as its priority prescribes. (Generally this will be done after the local minimizations in step 4.)

4. **Local Minimizations :** As processors become available, perform, on some processor, a local minimization from each start point selected in step 2.

Note : For each subregion and iteration, steps 1-3 constitute a single task, called a "subregion task", and each local minimization in step 4 constitutes a single "local minimization task". The scheduling of the tasks in parallel may be controlled in a centralized or distributed manner. A central process generally controls termination of the entire algorithm.

There are two key issues regarding the adaptive decisions in Algorithm 2.1. First, how does one identify subregions that should be given more emphasis in the computation? The goal is to emphasize subregions that are likely to contain undiscovered low minimizers. We have experimented with several heuristics, and among these, the most effective has been to emphasize subregions where the fraction of sample points with function values below the cutoff level is high. We call these "productive subregions"; regions where this fraction is particularly low are called "unproductive subregions". The second issue is, how does one redistribute the work so that more of the computational power is concentrated upon productive subregions? Here we have found that a combination of three heuristics is effective. First, we increase the sample size per iteration in productive subregions and decrease it in unproductive subregions, but not below some fixed lowest density. Second, if the sample size exceeds some upper bound we split the subregion into two subregions. Third, if the region is unproductive but the sample size can not be further decreased, we delay processing this subregion for one or more global iteration. Note that the net effect of these heuristics is to emphasize productive subregions, and to keep the work in the subregion tasks reasonably even.

There are also several important issues related to the asynchronous nature of the algorithm, and we only briefly describe the two main ones here. They are how tasks are scheduled among processors, and how the entire algorithm is terminated. One key aspect of scheduling is determining priorities among all the subregion and local minimization tasks. We note that each task still contains an iteration number that is determined in analogy to the sequential method. If multiple tasks are available to be processed, tasks from a lower numbered iteration are scheduled before tasks from a higher numbered iteration, and subregion tasks from a given iteration are scheduled before minimization tasks from the same iteration. The other key aspect of scheduling is how information regarding available tasks and processors is maintained and utilized. This turns out to be of great importance to the scalability of these methods, and is a considerable research topic of its own. It is discussed slightly later in this section, and is then the subject of Section 3.

Determining when to stop the asynchronous algorithm is somewhat complex, but the basic idea is that a central process determines when all tasks from an iteration have been completed, and then applies the termination criteria. Note that some tasks from succeeding iterations may already have executed at the point the algorithm is stopped. This appears to be the main penalty one pays in return for the improved load balancing in the asynchronous method.

We have implemented an adaptive, asynchronous parallel stochastic algorithm, based upon Algorithm 2.1 and the above discussion, on a network of Sun workstations. The initial version of this algorithm used a centralized scheduling mechanism that operates as follows. A single, separate scheduler process exists, and is informed of all tasks that are ready to be scheduled, and of all processors that have completed their last-assigned task. It then assigns the tasks to the processors using the criteria described above.

Smith and Schnabel [36] and Smith [34] report results using this algorithm. They use eight workstations for the worker processes (subregions and minimizations), and place the scheduler on a ninth workstation. Their experiments are on a single, artificial test problem that is constructed to be indicative of difficult problems with uneven concentrations of minimizers. The test problem has 2 variables and 46 local minimizers; the regions of attraction of the two highest (i.e. worst) local minimizers are contiguous and each constitute 25% of the domain D, whereas the regions of attraction of the 32 lowest local minimizers also form a contiguous subregion and each constitute 0.39% of the domain. The test problem was run with several different algorithmic configurations, corresponding to differing numbers of initial subregions and different initial sampling densities. The tests were run on all four combinations of static and adaptive, synchronous and asynchronous methods. Due to the stochastic nature of the algorithm, each problem-algorithm pair was run ten times.

Table 1 summarizes the results of these experiments for two typical versions of the test problem. In summary, the results indicate that the adaptive algorithm is considerably faster than the static algorithm on this type of problem, while the improvement in going from a synchronous to an asynchronous method is smaller, especially for the adaptive methods. Overall, the adaptive, asynchronous algorithm was between 1.5 and 7 times as fast as the static, synchronous algorithm in these tests. In addition, it achieved a speedup between of 6 and 7 over a sequential version of the adaptive algorithm in all of these cases. While 9 processors were used in the parallel method, the processor dedicated to the centralized scheduler never had a utilization of more than 12%, indicating that the overall parallel efficiency of the parallel algorithm was over 70%.

These experiments indicate that the adaptive versions of the stochastic methods can lead to significant improvements over the static stochastic methods, and that the adaptive, asynchronous parallel algorithm that was implemented can be expected to achieve good parallel efficiency for a small number of processors. Our experiences with these methods also indicated two problems with them that motivate the remainder of this paper. First, it seems clear that the centralized scheduler that was used in the initial parallel implementation will become a bottleneck as the number of processors is increased. The research described in Section 3 confirms this, and investigates alternative scheduling mechanisms within the context of the same adaptive, asynchronous global optimization method. Second, it seems likely that these methods will not be sufficient to solve large scale problems efficiently. In particular, it seems likely that their adaptive heuristics may be too general to focus attention on small, productive subregions of large dimensional domains quickly and tightly enough, and that the costs of sampling and minimizing in large dimensional spaces may become prohibitive. Section 4 describes some initial research that is aimed at modifying these methods to overcome these problems.

Table 2.1: Performance Comparison of Parallel Algorithms

density = 1000, 8 subregions	Static Synch	Static Asynch	Adapt Synch	Adapt Asynch
total number of iterations	8.6	7	6.5	6.1
total sample density	8383	8532	4026	3963
total number of local searches	39.6	46	47	53
total function evals	9186	9474	4966	5055
fnc eval improvement over static synch	-	-3%	46%	45%
computation time	23:19	15:01	4:10	3:25
time improvement over static synch	-	35%	82%	85%

density = 1000, 16 subregions	Static Synch	Static Asynch	Adapt Synch	Adapt Asynch
total number of iterations	5.6	5.7	5.4	5.7
total sample size	5891	5010	3048	3369
total number of local searches	69.9	81.5	45	69
total function evals	7341	6697	3976	4788
fnc eval improvement over static synch	-	9%	46%	35%
computation time	5:04	5:10	3:33	3:23
time improvement over static synch	-	-2%	30%	33%

3 Dynamic Scheduling of Parallel Global Optimization Algorithms

As discussed in Section 2, effective parallel stochastic global optimization algorithms generate tasks dynamically, and these tasks take varying amounts of time to execute. Thus it is not possible to determine a priori how to distribute the tasks among the processors to achieve good load balance, something that is possible for many simpler parallel numerical algorithms. Instead, some scheduling strategy must be used to keep track of the available tasks and processors as the algorithms proceeds, and to determine which tasks are to be executed by which processors in what order. We will refer to this as a dynamic scheduling strategy. The dual goals of any such strategy are to distribute the workload fairly evenly among the processors, and to keep the overhead cost of the scheduling computations as low as possible. Ideally, the dynamic scheduling strategy should be effective for a wide range of problem and parallel machine sizes.

In the case when the tasks are independent, the problem of dynamic scheduling has been addressed by a number of research projects in the computer science community (e.g. [10], [20], [26], [30]). On the other hand, little work has been done concerning dynamic scheduling of tasks that arise from the same application. We have investigated

this problem extensively, in the context of the parallel stochastic global optimization methods discussed in Section 2. This research has considered several dynamic scheduling approaches, including a *centralized scheduler*, two *distributed scheduling* strategies, and an approach that we term *centralized mediator*. In this section we will discuss these strategies briefly, and summarize the results of using them in the context of our parallel stochastic global optimization method. This research is described in far more detail in [34], and in preliminary form in [36].

The *centralized scheduler* strategy is the most straightforward. A master scheduling process keeps a priority queue of the tasks that are ready to execute, and sends tasks one at a time to the other, *worker* processes. Whenever a worker process finishes executing a task, it informs the master scheduler of this and in conjunction sends it any new tasks that it has created. The master scheduling process then sends it a new task to execute.

Centralized scheduling has been used in a variety of contexts on shared and distributed memory parallel computers, such as [8] and [22]. Its main advantages are the simplicity of implementation, and that the complete information possessed by the scheduler allows for good load balancing. The obvious disadvantage is that the scheduler can become a bottleneck as the number of processors increases or the granularity of tasks decreases.

Distributed scheduling strategies take essentially the opposite approach to centralized scheduling. Each processor maintains a local queue of tasks that are ready to execute, and schedules tasks to run from this queue if possible. If its workload becomes too heavy or too light, however, it tries to distribute tasks to other processors or obtain tasks from other processors. This is done in one of two ways. In a *receiver-initiated* strategy, processors that need more work request it from other processors. In a *sender-initiated* strategy, processors that have too much work offer excess tasks to other processors. In either case, the interaction is directly between processors, with no centralized control.

Distributed scheduling strategies have been investigated both in the context of systems where the tasks are independent (e.g. [10]), and where they are dependent (e.g. [12], [21]). In comparison to centralized strategies, they possess less complete information and thus may result in poorer load balancing, but the cost of scheduling should be far lower as the scheduling workload increases. On the other hand, there is a potential for $O(P^2)$ communication, where P is the number of processors, which may also become a problem as P becomes large.

The *centralized mediator* strategy combines aspects of both the centralized scheduler and distributed scheduling strategies. As in the distributed scheduling strategies, each processor maintains a local queue of tasks that are ready to execute, and schedules tasks to run from this queue if possible. When its workload becomes too heavy or light, however, rather than interacting directly with other processors, it sends requests or extra tasks to a centralized mediator process. This process then matches tasks to requests and sends tasks to the processors that requested them.

The centralized mediator strategy is a new contribution of our research, although related strategies have been used in some considerably different contexts such as [2] and [23]. In comparison to the centralized scheduler strategy, the scheduler should be far less of a bottleneck since it only handles a far number of tasks, namely those that the processors can not handle locally. In comparison to distributed scheduling strategies, it may be able to distribute tasks more evenly since there is some central information, and

it appears easier to implement. On the other hand, the centralized mediator process can still be expected to become a bottleneck eventually as the number of processors grows, and it is not clear how the scheduling overhead will compare to distributed scheduling strategies.

Smith [14] has conducted an extensive study of these dynamic scheduling strategies in the context of the adaptive, asynchronous, parallel stochastic global optimization methods described in Section 2. This study utilized a combination of analytic modeling, discrete event simulation, and parallel implementation. Analytic models of the centralized scheduler and centralized mediation strategies were used to identify likely bottleneck points to be investigated in the subsequent experiments. Discrete event simulations, using traces from the parallel centralized scheduler implementation discussed in Section 2, were used to examine the performance of all three scheduling strategies under a variety of workloads and for various numbers of processors. Parallel implementations of both the centralized scheduler and centralized mediator strategies on a network of 8 Sun workstations (plus one for the scheduler) were used to validate the results of the simulations and to perform further comparisons. They demonstrated that the differences between the implementation and the simulation results were small (ranging from 3% to 13%) and not statistically significant. In this section we summarize the simulation results, since they provide the broadest comparison of the dynamic scheduling strategies.

The simulations examined cases when there were 8, 16, 32, and 64 processors, and where the numbers of tasks created are small, medium, and large. They considered three versions of the global optimization problem used in the experiments discussed in Section 2, and used the adaptive asynchronous parallel algorithm described in that section. For each combination of number of processors, task workload, and initial algorithm configuration, 10 traces corresponding to ten different runs of the implemented method again were used, to minimize the effects of the stochastic nature of the algorithm.

In comparing the centralized scheduler strategy to the centralized mediator, the centralized scheduler was advantageous only when there were 8 processors and a small number of tasks. When there were 32 or 64 processors, the global optimization algorithm using the centralized mediator strategy always executed faster than the algorithm using the centralized scheduling strategy, for all workloads, with performance gains in the ranges of 22-64% and 53-77% respectively. In these cases, the centralized scheduler clearly became a bottleneck; for 64 processors the utilization of the scheduler process generally was 94%-99%. The utilization of the centralized mediator process was generally in the range of 50%-70% for 64 processors, indicating that it too would become a bottleneck for even larger numbers of processors.

In the comparison between the centralized mediator and distributed scheduling strategies, we utilized the received-initiated version of distributed scheduling since it appeared to be more efficient than the sender-initiated version in our preliminary tests. In all processor number, task workload, and initial algorithm configuration scenarios, the mean computation time of the centralized mediator version of the algorithm was less than the mean computation time of the distributed scheduling version. The differences in mean computation time were only statistically significant in fewer than half of these cases, however, mainly involving 32 and 64 processors. The gains in mean computation time by the centralized mediator ranged up to 28%.

These experiments indicate that, for the situations studied, the centralized mediator strategy is a robust and efficient dynamic scheduling strategy. It generally led to the lowest overall run time, and it never was far from optimal. For sufficiently large numbers of processors, however, it likely would be necessary to consider hierarchical versions of this strategy.

4 Solving Molecular Configuration Problems by Parallel Stochastic Global Optimization Methods

Recently, we have begun applying parallel stochastic global optimization methods to problems that arise from determining the structure of chemical systems. Many problems of this type lead to optimization problems because the naturally occurring structure minimizes, or nearly minimizes, the potential energy of the system or more complex energy measures. Commonly these energy functions have many local minimizers, so that it is necessary to solve a global optimization problem to determine the structure of interest. For general references on this subject, see e.g. [27] or [5].

The class of problems we have considered so far is to determine the configuration of molecules whose potential energy function is given by the sum of all the pairwise interactions between the atoms in the molecule, where these interactions are assumed to be VanderWaal's forces that are given by the Lennard-Jones 6-12 potential energy function. That is, if we define

$$x = (x_1, x_2, ..., x_m)$$

where each x_i is a vector denoting the coordinates of the i^{th} atom, then the potential energy function is

$$f(x) = \sum_{i=1}^{m} \sum_{j=i+1}^{m} \left[\frac{1}{d_{ij}^{12}} - \omega * \frac{1}{d_{ij}^{6}} \right] \tag{4.1}$$

where d_{ij} is the weighted Euclidean distance between atoms i and j. The weighted Euclidean distance is the standard Euclidean distance divided by a scalar that depends upon the sizes of atoms i and j. In the simplest case of the problem where all the atoms are identical, d_{ij} is the unweighted Euclidean distance. The potential energy function (4.1) is spherically symmetric, and thus is reasonable for inert atoms. The constant ω can be chosen arbitrarily and determines the scaling of the problem; [9] uses $\omega = 1$ (and multiplies (4.1) by 4) whereas [32] uses $\omega = 2$. We will use the same values as they do when we compare to their respective test problems and results.

Problems of the above form that are of practical interest may involve from hundreds to tens of thousands of atoms in three dimensional space. From the study of such problems with relatively small numbers of atoms, it is known that the number of local minimizers grows very rapidly with m. In fact, [17] has conjectured that the number of local minimizers is $O(e^{m^2})$. Furthermore, many of the local minimizers have function values that are near the global minimum function value (these minimizers come from repositioning of one or a few of the outermost atoms of the molecule), and each of these minimizers

has a small basin of attraction within the space of all possible configurations. For these reasons, these are very challenging global optimization problems. To our knowledge, only problems with relatively small numbers of atoms (typically 10-50 atoms in three dimensional space) have been considered so far, generally by methods that combine knowledge of chemical structures and optimization techniques. Even this number of parameters is one to two more orders of magnitude greater than the 2-6 parameter test problems that have been used in the general global optimization literature.

Our approach to solving these molecular configuration problems has been to use the general stochastic global optimization approach discussed in Section 2, with substantial modifications that make these algorithms suitable for solving large scale problems that have a special structure. In particular, (4.1) is an example of a *partially separable* function, one where the objective function is the sum of many terms, each of which involves only a small subset of the variables. Partially separable problems have been studied extensively in local optimization (see e.g. [15]), and it is felt that many, possible most large scale optimization problems have this form. Our approach is to develop a class of global optimization methods that is applicable to any large scale, partially separable function. Thus we do not use specific knowledge of molecular chemistry within our algorithms, rather we use such knowledge to aid us in constructing algorithmic techniques that are appropriate for any partially separable problem.

When one considers applying the class of methods described in Section 2 to find the global minimizer of functions of the form (4.1) with reasonably large m, there are two immediate challenges. These challenges are likely to be present for many other large scale problems as well. The first is how to efficiently locate sample points within the large-dimensional parameter space that will be good starting points for local minimizations. This is particularly difficult for the molecular configuration problem (4.1) because the value of $f(x)$ at randomly chosen sample points is usually 10-20 orders of magnitude higher than the optimal function values. (Again, we are assuming we do not use knowledge from molecular chemistry in determining good initial configurations.) The second challenge is how to find the global minimizer from among the many low local minimizers. In particular, one would like to be able to progress efficiently from one low local minimizer to an even lower one.

The main new technique that we have used so far to address these challenges is to *restrict the stochastic global optimization algorithm to carefully chosen small dimensional subproblems at key junctures in the overall full dimensional method.* We currently do this in two places, sampling and improvement of local minimizers. These correspond to the two main challenges described above.

The modified sampling phase begins as usual by sampling in the full dimensional parameter space, and discarding all but the lowest small fraction of the sample points. For each remaining sample point we then do the following. First we determine which atom contributes the most to the objective function value (4.1). This determination is easily made by summing the terms involving each atom, and it generalizes to other partially separable functions although perhaps in a application specific manner. Then we sample on just this atom (around its current value) and replace the current value of this atom with the new sample value of the atom that leads to the lowest new function value for this sample point. Finally this process is repeated a number of times, i.e. we select the atom

that contributes the most to the function value of the revised sample point (generally this will be a different atom), sample on this atom, replace it, and continue this process. We stop when the function value of the molecule falls below some predetermined threshhold or when the energy contribution of the worst atom falls below a second threshhold.

Generally this process reduces the function values of the sample points by many orders of magnitude. From these sample points, we determine start points for local minimizations as usual. The most important consequence of using this modified sampling procedure is that the local minimizations from the start points selected from these revised sample points generally lead to far better local minimizers than minimizations from start points selected from the original sample points. The second key point about the modified sampling procedure is that its cost is very low. The cost of sampling on one atom is $\frac{1}{m}$ times the cost of sampling in the full space, and the cost of re-evaluating (4.1) when one atom is perturbed is about $\frac{2}{m}$ times the cost of evaluating (4.1) in general. Finally, these techniques and savings generalize to any partially separable problem, although the determination of which variable(s) contributes most to the objective function may be application dependent.

The second place where we apply the technique of considering small dimensional subproblems is in the improvement of local minimizers. First we produce local minimizers as usual from the start points generated from the modified sample points. Then we apply the following procedure to each of the k lowest local minimizers. We again select the atom that contributes the most to the energy of the local minimizer, as was done in the modified sampling phase. Then we apply the adaptive global optimization algorithm described in Section 4 to this minimizer using the energy function (4.1) and *only this atom as a variable.* This is a three variable global optimization problem (assuming the problem is in three dimensional space). It results in the identification of various local minimizers for the three variable problem, i.e. several new positions for this atom within this configuration of the molecule. Next we take the two positions of this atom with the lowest function values (assuming neither is the original position of the atom), and apply local minimizations *in the full parameter space* to these configurations. This generally results in the identification of new local minimizers for the full problem. Finally we repeat this process some number of times, starting from the lowest new minimizer to which this procedure has not already been applied.

We have found this procedure to be very effective in locating improved local minimizers, in cases where simply doing more sampling and searching in the full parameter space was not productive. Again, its cost is very low. The global optimization algorithm applied to the problem where just one atom is a variable is inexpensive because of the small parameter space and the greatly reduced cost of evaluating the energy function. The subsequent local minimizations in the full parameter space generally require very few iterations and so they also are inexpensive. Finally, this step also generalizes to any partially separable problem, although again the determination of which variable(s) contributes most to the objective function may be application dependent.

The structure of the resultant global optimization algorithm that we use for solving molecular configurations problems is given in Algorithm 4.1. For simplicity we describe a sequential version of the algorithm even though in practice we have used a parallel variant of it.

Algorithm 4.1 – Framework of a Stochastic Global Optimization Algorithm for Large Scale Molecular Configuration Problems

Given $f : R^n \rightarrow R$, feasible region D

1. **Sampling in Full Domain :** Generate the coordinates of the sample points in the region D, and evaluate $f(x)$ at each new sample point. Discard all sample points whose function value is below a global "cutoff level".

2. **One-atom Sampling Improvement :** For each remaining sample point : Select the atom that contributes most to the energy function, sample on it, and replace this atom with the sample value that gives the lowest energy. Repeat this process for each sample point until its energy is below some threshhold.

3. **Start Point Selection :** Select a subset of the sample points to be start points for local minimizations.

4. **Full-Dimensional Local Minimizations :** Perform a local minimization from each start point selected in step 2.

5. **Improvement of Local Minimizers :** For each of the lowest k local minimizers : Select the atom that contributes most to the energy function, and apply a global optimization algorithm to this configuration with only this atom as a variable. Then apply a local minimization procedure, using all the atoms as variables, to the lowest configuration(s) that resulted from each the one-atom global optimization. Repeat this one-atom global optimization / full-molecule local minimization process some number of times from new low minimizers.

We have applied this algorithm to the function (4.1), with $\omega = 2$ and d_{ij} the Euclidean distance in three dimensional space, in the cases $m = 5, ..., 27$. (i.e. 15 to 81 parameters). These problems have been studied extensively by chemists and bio-chemists; see e.g. [18], [13], [11], [37], and [32]. Shalloway [32] presents a new "packet annealing" global optimization method that is designed to solve this type of problem, and presents test results for the cases $m = 5, ..., 24$. His algorithm finds the best known minimizer in 14 of the 20 cases (all but $m = 8$, 16, 17, 18, 23, and 24). Our algorithm finds the lowest known minimizer in all cases except $m = 22$, where it finds a *lower minimizer* than the previously known lowest minimizer for this case. This new configuration is a double icosahedron (19 atoms) plus three additional atoms that are asymmetrically placed, two adjoining the middle of the three pentagonal groups of atoms in the double icosahedron and the third adjoining either the top or bottom pentagonal group. Its function value is -86.8097 as compared to -86.148 for the lowest previously known minimizer. In each of the cases $m = 25$, 26, 27 we find the lowest minimizer reported in [18]; to our knowledge these are the lowest known minimizers for these problems. On many of these test problems, both of the one-atom modifications described above, to sampling and local minimization, were crucial in obtaining these results.

We have also applied our algorithm to the version of function (4.1) where $\omega = 1$ and d_{ij} is the Euclidean distance in *two* dimensional space, as has been considered by [9]. (The function (4.1) also is multiplied by 4.) In the case where $m = 50$, i.e, 100 parameters, we obtained a lowest potential energy of -137.3, whereas the lowest potential energy we obtained with the code reported in [9] was -135.4.

This research is still at a very early stage. While the above results indicate that the approach of Algorithm 4.1 is promising, problems that chemists and bio-chemists really want to solve will be far more challenging than the above test problems. They will have far more atoms, the atoms will not be identical, and the energy functions may be more complex. Correspondingly, there are many techniques that could be considered for adapting stochastic global optimization methods to solve these problems. The simplest modification of what we have described is to perturb not one atom, but some small number of atoms, in the low-dimensional sampling and minimization phases. This may be particularly important for molecules that are composed of several different types of atoms. Our future research will consider this possibility, as well as many other new algorithmic features. These may include some techniques that are particular to the molecular configuration problem. In conjunction the research will continue to consider the parallelization of the best algorithms, since it is clear that the solution of real, large problems will require massive amounts of computation and thus will need to utilize the fastest computers, which will almost certainly be massively parallel.

References

[1] Aluffi-Pentini, F., Parisi, V., and Zirilli, F. (1988). A global optimization algorithm using stochastic differential equations. *ACM Transactions on Mathematical Software*, **14**, 345-365.

[2] Anderson, T.E., Lazowska, E.D., and Levy, H.M. (December 1989). The performance implications of thread management alternatives for shared-memory multiprocessors. *IEEE Transactions on Computers*, **38**, (12), 1631-1644.

[3] Booker, L.B., Goldberg, D.E., and Holland, J.H. (1989). Classifier systems and genetic algorithms. *Artificial Intelligence*, **40**, 235-282.

[4] Branin, F.H. (1972). Widely convergent methods for finding multiple solutions of simultaneous nonlinear equations. *IBM Journal of Research Developments*, 504-522.

[5] Brooks, C.L., III, Karplus, M., and Pettitt, B.M. (1988). *Proteins: A Theoretical Perspective of Dynamics, Structure and Thermodynamics*. J. Wiley and Sons, New York.

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

[7] Dixon, L.C.W. and Szego, G.P., eds. (1978). *Towards Global Optimization*, **2**. North-Holland, Amsterdam.

[8] Dongarra, J.J. and Sorenson, D.C. (1987). Schedule: Tools for developing and analyzing parallel fortran programs. In *Characteristics of Parallel Algorithms*, D.B. Gannon, L.H. Jamieson, and R.J. Douglass, eds. MIT Press, 363-394.

[9] Donnelly, R.A. (1990). Discrete generalized descent for global optimization. Preprint, Department of Chemistry, Auburn University.

[10] Eager, D.L., Lazowska, E.D., and Zahorjan, J. (1986). A comparison of receiver-initiated and sender-initiated adaptive load sharing. *Performance Evaluation*, **6**, 53-68.

[11] Farges, J., DeFeraudy, M.F., Raoult, B., and Torchet, G. (1985). Cluster models made of double icosahedron units. *Surface Sci.*, **156**, 370-378.

[12] Finkel, R. and Manber, U. (1987). Dib - a distributed implementation of backtracking. *ACM Transactions on Programming Languages and Systems*, **9**, 235-256.

[13] Freeman, D.L. and Doll, J.D. (1985). Quantum Monte Carlo study of the thermodynamic properties of argon clusters: the homogeneous nucleation of argon in argon vapor and "magic number" distributions in argon vapor. *J. Chem. Phys.*, **82**, 462-471.

[14] Goldstein, A.A. and Price, J.F. (1971). On descent from local minima. *Mathematics of Computation*, **25**, 569-574.

[15] Griewank, A.O. and Toint, Ph.L. (1982). Partitioned variable metric updates for large sparse optimization problems. *Numerische Mathematik*, **39**, 119-137.

[16] Hansen, E.R. (1980). Global optimization using interval analysis - the multidimensional case. *Numerical Math*, **34**, 247-270.

[17] Hoare, M.R. (1979). Structure and dynamics of simple microclusters. *Adv. Chem. Phys.*, **40**, 49-135.

[18] Hoare, M.R. and Pal, P. (1979). Physical cluster mechanics: statics and energy surfaces for monatomic systems. *Adv. Phys.*, **20**, 161-196.

[19] Holland, J.H. (1975). *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor.

[20] Hsu, C.H. and Liu, J. (August 1986). Dynamic load balancing algorithms in homogeneous distributed systems. In *Proceedings of the 6th International Conference on Distributed Computing Systems.*

[21] Kale, L.V. and Shu, W. (August 1988). The chare-kernel language for parallel programming: a perspective. Technical Report UIUCDCS-R-88-1451, Department of Computer Science, University of Illinois at Urbana-Champaign.

[22] Kapenga, J.A. and DeDonker, E. (1988). A parallelization of adaptive task partitioning algorithms. *Parallel Computing*, **7**, North-Holland, 211-225.

[23] Kindervater, G. (1989). Exercises in parallel combinatorial computing. Ph.D. Thesis, Erasmus University, Rotterdam.

[24] Kirkpatrick, S., Gelatt, C.D. Jr., Vecchi, M.P. (1983). Optimization by simulated annealing. *Science*, **220**, 671-680.

[25] Levy, A.V. and Montalvo, A. (1985). The tunneling algorithms for the global minimizer of functions. *SIAM Journal on Scientific and Statistical Computing*, **6**, 15-29.

[26] Lin, F.C.H. and Keller, R.M. (August 1986). Gradient model: a demand-driven load balancing scheme. In *Proceedings of the 6th International Conference on Distributed Computing Systems.*

[27] McCammon, J.A. and Harvey, S.A. (1987). *Dynamics of Proteins and Nucleic Acids.* Cambridge University Press, Cambridge, England.

[28] Rinnooy Kan, A.H.G. and Timmer, G.T. (1984). A stochastic approach to global optimization. In *Numerical Optimization*, P. Boggs, R. Byrd, and R.B. Schnabel, eds., SIAM, Philadelphia, 245-262.

[29] Rinnooy Kan, A.H.G. and Timmer, G.T. (1989). Global optimization. In *Handbooks in operations research and management science, volume I : optimization*, G.L. Nemhauser, A.H.J. Rinnooy Kan, and M.J. Todd, eds., North-Holland, 631-662.

[30] Ross, A. and McMillin, B. (April 1991). Experimental comparison of bidding and drafting load sharing protocols. *The Fifth Distributed Memory Computing Conference*, 968-974

[31] Schnabel, R.B., Koontz, J.E., and Weiss, B.E. (1985). A modular system of algorithms of unconstrained minimization. *ACM Transactions on Mathematical Software*, **11**, 419-440.

[32] Shalloway, D. (1991). Packet annealing : a deterministic method for global minimization, with application to molecular conformation. Preprint, Section of Biochemistry, Molecular and Cell Biology, Cornell University. To appear in *Global Optimization*, C. Floudas and P. Pardalos, eds., Princeton University Press.

[33] Shubert, B.O. (1972). A sequential method seeking the global maximum of function. *SIAM Journal on Numerical Analysis*, **9**, 379-388.

[34] Smith, Sharon L. (1991). Adaptive asynchronous parallel algorithms in distributed computation. Ph.D. Thesis, Department of Computer Science, University of Colorado, Boulder, Colorado.

[35] Smith, S.L., Eskow, E., and Schnabel, R.B. (1989). Adaptive, asynchronous stochastic global optimization algorithms for sequential and parallel computation. In *Proceedings of the Workshop on Large-Scale Numerical Optimization*, T.F. Coleman and Y. Li, eds. SIAM, Philadelphia, 207-227.

[36] Smith, Sharon L. and Schnabel, Robert B. (1991). Centralized and distributed dynamic scheduling for adaptive, parallel algorithms. Technical Report CU-CS-516-91, Department of Computer Science, University of Colorado, Boulder, Colorado.

[37] Wille, L.T. (1975). Minimum-energy configurations of atomic clusters: new results obtained by simulated annealing. *Chem. Phys. Lett.*, **133**, 405-410.