A Parallel Algorithm for Bound-Smoothing

Kumar Rajan
Narsingh Deo
Department of Computer Science
University of Central Florida
Orlando, FL 32816-2362
dharmara@longwood.cs.ucf.edu

Abstract

Determining molecular structure from interatomic distances is an important and challenging problem. Given a molecule with n atoms, lower and upper bounds on interatomic distances can usually be obtained only for a small subset of the $\frac{n(n-1)}{2}$ atom pairs, using NMR. Given the bounds so obtained on the distances between some of the atom pairs, it is often useful to compute tight bounds on all the $\frac{n(n-1)}{2}$ pairwise distances. This process is referred to as bound-smoothing. *The initial lower and upper bounds for the pairwise distances not measured are usually assumed to be 0 and ∞.*

One method for bound-smoothing is to use the limits imposed by the triangle inequality. *The distance bounds so obtained can often be tightened further by applying the* tetrangle inequality *— the limits imposed on the six pairwise distances among a set of four atoms (instead of three for the triangle inequalities). The tetrangle inequality is expressed by the Cayley-Menger determinants. For every quadruple of atoms, each pass of the tetrangle-inequality bound-smoothing procedure finds upper and lower limits on each of the six distances in the quadruple. Applying the tetrangle inequalities to each of the $\binom{n}{4}$ quadruples requires $\Theta\left(n^4\right)$ time. Here, we propose a parallel algorithm for bound-smoothing employing the tetrangle inequality. Each pass of our algorithm requires $\Theta\left(n^3 \log n\right)$ time on a CREW PRAM with $\Theta\left(\frac{n}{\log n}\right)$ processors. An implementation of this parallel algorithm on the Intel Paragon XP/S and its performance are also discussed.*

1 Introduction

Given an incomplete, undirected, edge-weighted graph, the *molecular conformation* problem is that of mapping the vertices of the graph to points in the Euclidean k-space so that any two vertices with an edge between them are mapped to points whose Euclidean distance equals the weight of that edge. For all practical purposes, k is either 2 or 3. An important application (for $k = 3$) is in determining three-dimensional molecular structure from a small subset of imprecisely measured interatomic distances. This problem is one of the most important and challenging problems in computational biology [11] today.

The molecular conformation problem is known to be NP-hard in the strong sense [10]. As a result, most of the methods find only approximate solution (in polynomial time). One such method that is particularly useful when determining the molecular structure where the interatomic distances are known only within specified bounds is the *distance geometry* method [4]. The first step of the distance geometry method is *bound-smoothing*. In this step, given the upper and lower bounds on a subset of pairwise distances, triangle and/or tetrangle inequalities are used to obtain the bounds on the remaining distances, as well as to further tighten the existing bounds [4] (pp. 221–285).

In this paper, we are concerned with only the bound-smoothing step of the distance geometry method. Since any three points in the Euclidean space have to satisfy the triangle inequality, bounds could be tightened by applying the triangle inequality to three vertices at a time. Let u_{ij} and l_{ij} represent the upper and lower bounds on the distance d_{ij} between vertices i and j. Given three vertices i, j, and k, the upper bounds on the distances between them must satisfy the triangle inequality $u_{ij} \leq u_{ik} + u_{jk}$. Thus, if $u_{ij} > u_{ik} + u_{jk}$, then u_{ij} is replaced by $u_{ik} + u_{jk}$. Similarly, if the (i, j) lower bound satisfies $l_{ij} < l_{ik} - u_{jk}$, then we can raise it to $l_{ij} := l_{ik} - u_{jk}$. By iterating on this substitution procedure, we can often tighten the measured distance bounds significantly, as well as obtain bounds on the values of those distances for which no bounds were available, obtaining in the end a complete set of bounds that satisfy:

$$u_{ij} \leq u_{ik} + u_{jk} \text{ and } l_{ij} \geq l_{ik} - u_{jk}.$$

It has been shown that the "triangle inequality slack" $u_{ij} - u_{ik} - u_{jk} > 0$ and $l_{ij} - l_{ik} - u_{jk} > 0$ in the distance bounds can be eliminated in polynomial time by means of an all-pairs shortest path algorithm [6].

Often, tighter bounds can be obtained by using the *tetrangle inequality*, which is expressed by means of Cayley-Menger determinants, i.e.

$$0 \leq CM(d_{12}, \ldots, d_{34}) = \begin{vmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & d_{12}^2 & d_{13}^2 & d_{14}^2 \\ 1 & d_{21}^2 & 0 & d_{23}^2 & d_{24}^2 \\ 1 & d_{31}^2 & d_{32}^2 & 0 & d_{34}^2 \\ 1 & d_{41}^2 & d_{42}^2 & d_{43}^2 & 0 \end{vmatrix},$$

where the d_{ij} denote the distances among the four vertices $\{1, 2, 3, 4\}$. It is well-known that $CM(d_{12}, \ldots, d_{34}) \geq 0$, along with the triangle inequality, constitutes a necessary condition for the numbers $d_{12}, \ldots, d_{34}$ to be the distances among a quadruple of points in the 3-D Euclidean space [4]. In order to tighten the bounds on the distance between points 3 and 4, for example, we have 32 inequalities involving u_{34} and 32 involving l_{34}. This means that we need to solve a total of 64 inequalities for tightening the bounds on each of the six distances. Fortunately, providing that the combinations of bounds involved satisfy the triangle inequality, it has been shown that among the 64 possible inequalities that can be used to tighten the bounds on a given distance, only seven are non-redundant [4] (pp. 254–260). These combinations of bounds are shown in Figure 1.

No polynomial time algorithm (similar to the all-pairs shortest path algorithm for the triangle inequality limits) is known for eliminating the tetrangle inequality slack. In fact, Crippen and Havel [4] (page 274) conjecture that this problem is NP-hard. However, since each of the seven expressions $CM(\ldots)$ is quadratic in the square of the $(3, 4)$ bound, these inequalities can be solved analytically in order to eliminate the slack in one of the six distances among each quadruple. For details, the reader is referred to [4] (pp.254–267). We then repeatedly go through all quadruples and eliminate slack in the bounds among each until no further changes occur. This process is shown as procedure *Tetra* in Figure 2. The **for** loop (lines 3–8) considers all the quadruples and tightens the bounds of all the six distances of each. There are $\binom{n}{4}$ such quadruples and each execution of the outer-most **repeat** loop (lines 2–9) takes $\Theta\left(n^4\right)$ time. Each iteration of this **repeat** loop will subsequently be referred to as a "pass".

Sequential implementations of procedure *Tetra* have been studied in the literature [7, 8, 9]. The results of these studies suggest that tetrangle inequalities are far superior to triangle inequalities and usually result in much tighter bounds. When applied to the molecular structure determination problem, the tetrangle inequalities propagate the short-range distance bounds (which are more easily obtained from experimental methods than the long-range distance bounds) to the long-range distances much better than the triangle inequalities [7]. A major drawback of these implementations, as the authors note, is that due to the enormous computational time requirements (it took five passes and about six hours on the Cray X-MP to compute the tetrangle inequality limits for a problem of size 128 [7]), these sequential implementations are limited to problems of small sizes. Although these studies suggest that the iterative limits obtained by procedure *Tetra* are a significant improvement over the triangle inequality limits, questions remain concerning the convergence of the procedure as well as the uniqueness of the limits produced with respect to the order in which the quadruples are considered.

The object of this paper is to present a simple, parallel algorithm for tetrangle-inequality bound-smoothing so that it can be applied to large problem sizes on a parallel machine in a reasonable amount of time. On a CREW PRAM, where the distance matrix can be stored in the shared memory, each pass of our parallel algorithm can be carried out in $\Theta\left(n^3 \log n\right)$ time with $\Theta\left(\frac{n}{\log n}\right)$ processors. We also describe an implementation of this parallel algorithm on the Intel Paragon XP/S, a distributed-memory MIMD machine with a 2D-mesh interconnection network. In the next section, we identify the requirements for parallelizing the sequential algorithm and get a theoretical bound on the maximum speedup that can be obtained. In the remaining sections, we describe our parallel algorithm and discuss the results obtained on the Intel Paragon XP/S.

2 Requirements for Parallelization

One way to parallelize the tetrangle-inequality bound-smoothing process is to compute the quadruples (i.e. tighten the bounds on all the six distances using the seven non-redundant inequalities shown in Figure 1) in parallel. Quadruples that can be computed in parallel are exactly the ones that do not share more than one element. Computing quadruples that share more than one element would result in a concurrent write conflict. In the following definition, we characterize this criterion formally, and then obtain an upper bound on the number of quadruples that can be computed concurrently. Consider the following definition from design theory [13]:

Definition 1 *A t-(v, k, λ) packing consists of a v-set X and a collection of k-subsets of X, called* blocks, *such that every t-subset of X is contained in at most λ of them. The packing number $D_\lambda(v, k, t)$ is the number of blocks in a maximum t-(v, k, λ) packing. A "large set of disjoint t-(v, k, λ) packings" is a partition of the set of all k-subsets of a v-set X*

into mutually disjoint t-(v,k,λ) packings of maximum size (given by $D_\lambda(v,k,t)$).

Given a set of n elements, the problem of generating quadruples that do not share more than one element is the same as the problem of generating 2-$(n,4,1)$ packings. The maximum number of quadruples that can be computed concurrently is given by $D_1(n,4,2)$, whose value was discovered by Brouwer [1]:

Theorem 1 *(Brouwer [1]) The packing number for a 2-$(n,4,1)$ packing is given by:*
$D_1(n,4,2) = \lfloor \frac{n}{4} \lfloor \frac{n-1}{3} \rfloor \rfloor - \epsilon$, *where*

$$\epsilon = \begin{cases} 1 & \text{if } n \equiv 7 \text{ or } 10 (mod 12), n \neq 10, 19 \\ 1 & \text{if } n = 9 \text{ or } 17 \\ 2 & \text{if } n = 8, 10 \text{ or } 11 \\ 3 & \text{if } n = 19 \\ 0 & \text{otherwise} \end{cases}$$

Thus, for n elements, the maximum number of quadruples that can be computed concurrently is $\Theta\left(n^2\right)$. Assuming that we have enough processors to compute all the quadruples in any packing concurrently, the time taken to compute the bounds for all the quadruples is proportional to the number of such packings. For maximum speedup, we require that the number of such packings be minimum. Maximum speedup can be obtained using a "large set of disjoint 2-$(n,4,1)$ packings" if one exists. However, there are no known results concerning the existence of "large sets of disjoint 2-$(n,4,1)$ packings" for arbitrary values of n, except for $n = 13$. For this case, it has been shown by Chouinard [3] that the $\left(\begin{smallmatrix} 13 \\ 4 \end{smallmatrix} \right)$ quadruples can be partitioned into 55 disjoint 2-$(13,4,1)$ packings, each of size $D_1(13,4,2) = 13$. For a discussion on "large sets of disjoint packings", refer to [14] (pp. 586–587).

The problem of partitioning the set of all 4-subsets of an n-set into disjoint 2-$(n,4,1)$ packings can also be formulated as a graph coloring problem as follows. Let $S(n) = \{0,1,\ldots,n-1\}$ be the n-set and let $V(n)$ be the set of all 4-subsets of $S(n)$. Construct a graph $G(n) = (V(n), E(n))$, where $E(n) = \{(u,v) | u,v \in V(n), |u \cap v| = 2 \text{ or } 3\}$. It can be verified that $G(n)$ is a regular graph with $\left(\begin{smallmatrix} n \\ 4 \end{smallmatrix} \right)$ vertices and degree, $3n^2 - 23n + 44$. Any two vertices of $G(n)$ have an edge between them if and only if they have at least two elements in common. Vertices in any independent set of $G(n)$, therefore, have at most one element in common and form a 2-$(n,4,1)$ packing. A proper vertex coloring of $G(n)$ partitions the set $V(n)$ into disjoint 2-$(n,4,1)$ packings, and the number of packings in a "large set of disjoint 2-$(n,4,1)$ packings" of $S(n)$ is the *chromatic number* of $G(n)$. It is known that finding the chromatic number is NP-hard for regular graphs [5]. It is not known whether it remains NP-hard for the graphs $G(n)$.

In the next two sections, we present a simple, parallel algorithm for a proper coloring of the vertices of $G(n)$ (and thereby generating disjoint 2-$(n,4,1)$ packings of the set $S(n)$), by taking advantage of the structure of $G(n)$. Our algorithm does not find the chromatic number of $G(n)$. But for any given $p \leq n$, it colors the vertices of $G(n)$ so that almost all the 2-$(n,4,1)$ packings (independent sets) generated by it are of size p, especially for large values of n.

3 A Parallel Coloring Algorithm for the Graph $G(n)$

Since the vertices of $G(n)$ are the 4-subsets of the set $S(n)$, we denote them by $\{x,y,z,w\}$, where $0 \leq x,y,z,w \leq n-1$. For $0 \leq i \leq n-1$, let π_i denote the permutation on the elements of $V(n)$ defined as $\pi_i(\{x,y,z,w\}) = \{x+i, y+i, z+i, w+i\}$, where all additions are carried out modulo n. The set $\{\pi_0 \ldots \pi_{n-1}\}$ of permutations can be seen to form a group, say $\mathcal{G}(n)$, with the multiplication operation being composition, the identity element being π_0, and the inverse of π_i being π_{n-i}. It is known from the theory of counting (see, for example, Tucker [15]), that the equivalence relation induced by the group $\mathcal{G}(n)$ partitions $V(n)$ into equivalence classes.

The number, $N(n)$, of such equivalence classes can be computed using Burnside's Lemma, stated as the following theorem:

Theorem 2 (Burnside [15], pp. 342–343) *Let A be a set of elements and $\mathcal{H}$, a group of m permutations applied to A. For $a \in A$, let $\phi(a)$ denote the number of permutations in $\mathcal{H}$ that leave a unchanged. Then the number of equivalence classes into which $\mathcal{H}$ partitions A is given by $\frac{1}{n}\sum_{a \in A} \phi(a)$.*

In our case, where the permutations π_i are defined as above, for any $v \in V(n)$, there can be a maximum of four permutations that leave v unchanged. This happens when v is of the form $v = \{x, x+\frac{n}{4}, x+\frac{n}{2}, x+\frac{3n}{4}\}$, in which case the permutations π_0, $\pi_{\frac{n}{4}}$, $\pi_{\frac{n}{2}}$, and $\pi_{\frac{3n}{4}}$ leave v unchanged. Therefore for any $v \in V(n)$, we have $\phi(v) \leq 4$. Since there are $\left(\begin{smallmatrix} n \\ 4 \end{smallmatrix} \right)$ elements in $V(n)$, from Theorem 2 we have, $N(n) \leq \frac{4}{n}\left(\begin{smallmatrix} n \\ 4 \end{smallmatrix} \right)$, or $N(n) \leq \frac{(n-1)(n-2)(n-3)}{6}$. It can also be seen that for any $v \in V(n)$, $\pi_0(v)$ leaves v unchanged. Therefore, $\phi(v) \geq 1, \forall v \in V(n)$, and therefore by Theorem 2, $N(n) \geq \frac{(n-1)(n-2)(n-3)}{24}$. While the number of vertices of $G(n)$ is $\Theta\left(n^4\right)$, the number of equivalence classes is only $\Theta\left(n^3\right)$. We will use this result later to bound the running time of our parallel algorithm. The algorithm we propose, colors the vertices of $G(n)$ in such a way that each vertex in any independent set (i.e., the set of vertices receiving the same color) belongs to a distinct equivalence class. The following result (stated without proof) guarantees that

once we have an independent set with this property, we can generate $\Theta(n)$ independent sets mutually disjoint with each other by just applying the permutations π_i $(0 \leq i \leq n-1)$ to the independent set:

Theorem 3 *Let $I = \{v_1, v_2, \ldots, v_p\}$, $v_i \in V(n)$, be an independent set. Then, any permutation π_i of I, defined as $\pi_i(I) = \{\pi_i(v_1), \pi_i(v_2), \ldots, \pi_i(v_p)\}$, $0 \leq i \leq n-1$, is also an independent set.*

The algorithm for coloring $G(n)$ is is summarized in procedure *SeqColor* (Figure 3), which works as follows. The set *packing* holds, at any time, a set of vertices that receive the same color (i.e., a 2-$(n, 4, 1)$ packing), and the array *color* holds the color of each vertex. The set *packing* is initialized to be empty. The equivalence classes of the group $\mathcal{G}(n)$ are considered in a specific order that is, heretofore, referred to as the *lexicographic order*. Each equivalence class is searched for a vertex that does not have an edge with any of the vertices already present in *packing* (line 7). If such a vertex is found, it is added to *packing* and the next equivalence class is considered. This process is continued until either the maximum size (p) is reached for *packing*, or the next equivalence class does not have any vertex that can be added to *packing*. Once one of these conditions is met, distinct colors are applied to all the n permutations of *packing* (lines 11-14), and the next packing is generated the same way. This process is repeated until all the equivalence classes are considered, with the array *color* containing a proper vertex coloring of $G(n)$ at the end.

Figure 4 plots the percentage of packings of size p for $p = \lceil \frac{n}{2} \rceil$, $\lceil \frac{3n}{4} \rceil$, and n. As n increases, the algorithm is able to partition the quadruples into packings of larger sizes with increasing efficiency. For example, almost 90% of the packings generated by $SeqColor(1000, 1000)$ are of size 1000. This means that if we employ 1000 processors to solve a problem of size 1000, almost 90% of the packings will keep all the processors busy.

We now present a parallel coloring algorithm based on procedure *SeqColor*. In each step of the parallel coloring algorithm, we generate an independent set of vertices (i.e., a 2-$(n, 4, 1)$ packing) in such a way that each processor holds at most one vertex in the set. We assume that there are p processors, $1 \leq p \leq n$. The algorithm for generating one such packing is shown as procedure *ParColor* in Figure 5. It has been designed to be easily called by the parallel tetrangle-inequality bound-smoothing algorithm to be discussed in the next section.

As in *SeqColor*, equivalence classes are considered in the lexicographic order. It is assumed that the information regarding the next p equivalence classes to be considered (lines 3–20) is stored globally. As a result, repeated invocations of the procedure consider successive equivalence classes and generate mutually disjoint packings. The variable $nextProc$ keeps track of the processor to receive the next vertex to be generated, and *packing* contains the packing. Checking to see if there is a vertex in the equivalence class $\mathcal{E}$ that can be added to *packing*, is done in parallel. Since there could be up to n vertices in $\mathcal{E}$, and there are p processors, each processor checks up to $\frac{n}{p}$ vertices (line 6–11).

Checking if a vertex can be added to *packing* can be done in constant time by each processor as follows. The data structure *packing* can be implemented as a two-dimensional $(n \times n)$ boolean array. The ijth entry is 1 if the elements i and j are present in a vertex belonging to *packing*, and 0, otherwise. A vertex can be added to *packing* if and only if none of the six pairs in the vertex has a 1 entry in the array, which can be checked in constant time. Initializing *packing* to "Empty" (as in line 2) corresponds to setting all the entries in this array to 0. For the sake of clarity, the code that deals with this array is not shown in *ParColor*. We just note that it can be implemented so that it does not increase the overall time complexity of the algorithm.

After all p processors have checked their vertices, the vertex belonging to one of them is chosen to be added to *packing* and the procedure continues with the next equivalence class, considering up to p equivalence classes. If no processor can find such a vertex, the procedure terminates and returns *packing*, which contains the packing generated so far. We note that processors might need concurrent read accesses to the entries of the boolean array described above. There is no need for concurrent writes to this array since only one processor (*nextProc*) needs to update this array after it receives its vertex. Thus, our algorithm requires a CREW PRAM. The total time taken by *ParColor* can be verified to be $\Theta(n + p \log p)$. Based on the procedure *ParColor*, we give a parallel algorithm for tetrangle-inequality bound-smoothing in the next section.

4 The Parallel Bound-Smoothing Algorithm

A parallel tetrangle-inequality bound-smoothing algorithm based on procedure *ParColor* is given as procedure *ParTetra* in Figure 6. In line 4, procedure *ParColor* is invoked to get the next 2-$(n, 4, 1)$ packing of size at most p. *ParColor* ensures that the vertices in each packing belong to a distinct equivalence class and that each processor has at most one vertex. Since we know that all the permutations $\pi_i, 0 \leq i \leq n-1$, of these vertices also are 2-$(n, 4, 1)$ packings (Theorem 3), mutually disjoint with each other, the processors can compute all the n permutations of the vertices synchronously and tighten the bounds of each of them in parallel. This is done in the loop in lines 6–8. The **while** loop in lines 3–10 ensures that this process is repeated until all the equivalence classes are considered. This constitutes one pass of the tetrangle-inequality bound-smoothing algo-

rithm. These passes are repeated (**repeat** loop in lines 2–11) until the changes in the distance bounds are within the specified tolerance Tol. Assuming that $p \leq n$, and that the size of each 2-$(n, 4, 1)$ packing generated by procedure *ParColor* is $\Theta(p)$, the total time taken by procedure *ParTetra* can be verified to be $\Theta\left(\frac{n^3}{p}(p \log p + n)\right)$ in contrast to the $\Theta(n^4)$ time taken by each pass of the sequential procedure. The speedup can be derived to be $\Theta\left(\frac{p}{1+\frac{p \log p}{n}}\right)$. On a CREW PRAM with $\Theta\left(\frac{n}{\log n}\right)$ processors, the efficiency can be derived to be $\Theta(1)$.

5 Experimental Results

Procedure *ParTetra* (Figure 6) was implemented on the Intel Paragon XP/S machine with 32 processors. The Paragon XP/S is a distributed-memory, message-passing machine with a 2D-mesh interconnection network. We first compare the parallel time with the sequential time by using randomly generated inputs for a given size n and *sparsity*, where *sparsity* is a measure of the number of measured distances. The n points were first generated randomly inside a three dimensional box of size $2 \times 2 \times 10$, resembling long molecular structures (e.g., proteins). Then for a fixed number of pairs of points (depending on n and *sparsity*), the true distance for each pair was randomly perturbed within a 10% limit to obtain lower and upper bounds, resembling NMR which typically has an experimental error of around $\pm 10\%$ [2]. The unspecified upper bounds were initialized to 99Å, and the unspecified lower bounds to 0Å.

In the following analysis, the speedup for p processors refers to the ratio of the sequential time to the parallel time for p processors. The sequential time is the time taken for one pass of procedure *Tetra* on *Sun Sparc Ultra-Enterprise* sequential machine, and the parallel time for p processors is the time taken for one pass of procedure *ParTetra* on the Intel Paragon X/PS machine with p processors.

To illustrate the algorithm's scalability, Figure 7 compares the sequential time and the parallel time for a various problem sizes (n) on different numbers of processors (p). The values reported are the average of 10 executions for different randomly generated inputs. For a fixed number of processors the performance scales well with problem size as our complexity analysis shows.

We now apply the parallel bound-smoothing algorithm using tetrangle inequality to three real-life molecules for which a subset of the pairwise interatomic distances were obtained through NMR: the murine macrophage inflammatory protein-2 (MIP-2) [12], a 21-residue DNA, and the P5B stem loop RNA from a Group 1 Intron complexed with Cobalt (III) Hexammine. The NMR data for the DNA and the RNA were obtained from the Brookhaven Data Bank, accession numbers 149D and 1AJF, respectively.

There were 201 pairwise distances measured among 128 of the atoms of the RNA, and 467 pairwise distances among 222 of the atoms of the DNA. MIP-2 is a dimer with each monomer having 1137 atoms. For each monomer, a total of 1118 distances were available between 430 of its atoms. We only applied the tetrangle-inequality algorithm to one monomer as the monomers are identical. In Table 1, we summarize the number of atoms that we considered for each of the molecules and the number of pairwise distances that were available for each of them.

For each of the sample problems, the lower and upper bounds of the measured pairwise distances were initialized from the input data and the bounds of unmeasured distances were initialized to 0Å for the lower bounds and 99Å for the upper bounds. In Table 2, we list the number of iterations and the average time taken for each iteration of the tetrangle inequality algorithm. All these problems were run on the *Intel Paragon X/PS* machine configured for 32 processors.

It can be observed from Table 2 that while it took a parallel time of about 11 hours for one iteration of the tetrangle-inequality bound-smoothing algorithm for the MIP-2 molecule, it would take an estimated time of about 10 days for the same on a state-of-the-art sequential machine like the *Sun Sparc Ultra-Enterprise*. Since our parallel algorithm scales well with the number of atoms, with more processors, it should be possible to compute bounds for molecules of fairly large sizes in a reasonable amount of time.

A common measure of the smoothness of a set of pairwise distance bounds is the *root mean square gap* (RMSG) — the root mean square of the differences between the upper and lower bounds of all the distances. Table 3 compares the RMSG obtained after the tetrangle and the triangle inequality algorithms. It was found that even though the RMSG shows only a marginal decrease, the tetrangle inequality is particularly effective in tightening the upper bounds. As Table 3 shows, for the MIP-2, it was found that even though the RMSG improved only to 45.23Å for the tetrangle algorithm from 46.19Å for the triangle algorithm, the tetrangle algorithm tightened as many as 19182 upper bounds, as opposed to just 2347 upper bounds tightened by the triangle inequality.

It is known, especially for large problem instances, that the tetrangle inequality enables one to determine substantially tighter bounds on long-range distances [7]. This property is considered useful in the determination of polypeptide structure from NMR data, since this data is relatively rich in short-range information. In Table 3, we also compare the number of such tight bounds found by triangle and tetrangle algorithms. For the purpose of this calculation, any distance whose lower bound was greater than 6Å was considered to be long-range, and a $(lower, upper)$ bound pair was con-

sidered tight when the bounds differed by less than 3Å. It can be noticed that for the MIP-2 problem, the tetrangle algorithm was able to obtain tight bounds on 124 long-range distances, while the triangle inequality could do so only for 11 of them.

6 Conclusions and Directions for Future Work

We have shown that each pass of the tetrangle-inequality bound-smoothing algorithm can be parallelized. Since our algorithm scales well, for problems of large sizes (for molecules with thousands of atoms), it should now be possible to tighten the bounds using the tetrangle inequality on a parallel machine in a reasonable amount of time. Figure 4 shows that for large problem sizes ($n \geq 1000$), it should be possible to (almost) efficiently use up to n processors.

There are several open questions concerning tetrangle-inequality bound-smoothing and "large sets of disjoint 2-$(n, 4, 1)$ packings". It is not known whether the problem of eliminating the tetrangle inequality slack is NP-hard. There are no known results regarding the existence of "large sets of disjoint 2-$(n, 4, 1)$ packings" for arbitrary values of n. The only known result is the construction of a "large set of disjoint 2-$(13, 4, 1)$ designs" by Chouinard [3]. Even though empirical results suggest that for large values of n, our partitioning program generates packings so that almost all the packings are of size n, the theoretical bounds on the number of packings and the size of the packings generated is still an open problem. We thank Dr.Barry Schweitzer for providing us with the NMR restraints for the MIP-2 molecule.

References

[1] A. E. Brouwer. Optimal packings of k_4's into a k_n. *Journal of Combinatorial Theory*, 26:278–297, 1979.

[2] A. T. Brünger and M. Nilges. Computational challenges for macromolecular structure determination by X-ray crystallography and solution NMR-spectroscopy. *Quarterly Review of Biophysics*, 26:49–125, 1993.

[3] L. G. Chouinard. Partitions of the 4-subsets of a 13-set into disjoint projective planes. *Discrete Math.*, 45:297–300, 1983.

[4] G. M. Crippen and T. F. Havel. *Distance Geometry and Molecular Conformation*. Research Studies Press Ltd., Taunton, Somerset, England, 1988.

[5] D. P. Dailey. Uniqueness of colorability and colorability of planar 4-regular graphs are NP-Complete. *Discrete Math.*, 30:289–293, 1980.

[6] A. W. M. Dress and T. F. Havel. Shortest-path problems and molecular conformation. *Discrete Applied Mathematics*, 19:129–144, 1988.

[7] P. L. Easthope and T. F. Havel. Computational experience with an algorithm for tetrangle-inequality bound-smoothing. *Bulletin of Mathematical Biology*, 51:173–194, 1989.

[8] N. Kumar, N. Deo, and R. Addanki. Empirical study of a tetrangle-inequality bound-smoothing algorithm. *Congressus Numerantium*, 117:15–31, 1996.

[9] K. Rajan, N. Deo, and N. Kumar. Parallel Tetrangle-Inequality Bound-Smoothing on a Cluster of Workstations. *Congressus Numerantium*, 124:211–220, 1997.

[10] J. B. Saxe. Embeddability of weighted graphs in $k - space$ is strongly np-hard. In *Proc. 17th Allerton Conference in Communications, Control and Computing*, pages 480–489, 1979.

[11] D. B. Searls. Grand challenges in computational biology. In S. L. Salzberg, D. B. Searls, and S. Kasif, editors, *Computational Methods in Molecular Biology*, volume 32, pages 3–10. Elsevier, 1998.

[12] W. Shao, L. F. Jerva, J. West, E. Lolis, and B. I. Schweitzer. Solution structure of murine macrophage inflammatory protein-2. *Biochemistry*, 37:8303–8313, 1998.

[13] D. R. Stinson. Packings. In C. J. Colbourn and J.H. Dinitz, editors, *THE CRC HANDBOOK OF COMBINATORIAL DESIGNS*, pages 409–413. CRC Press, 1996.

[14] L. Teirlinck. Large sets of disjoint designs and related structures. In J. H. Dinitz and D. R. Stinson, editors, *Contemporary Design Theory: A Collection of Surveys*, pages 561–592. John Wiley & Sons, Inc., 1992.

[15] A. Tucker. *Applied Combinatorics*. John-Wiley & Sons, Inc., New York, 1984.

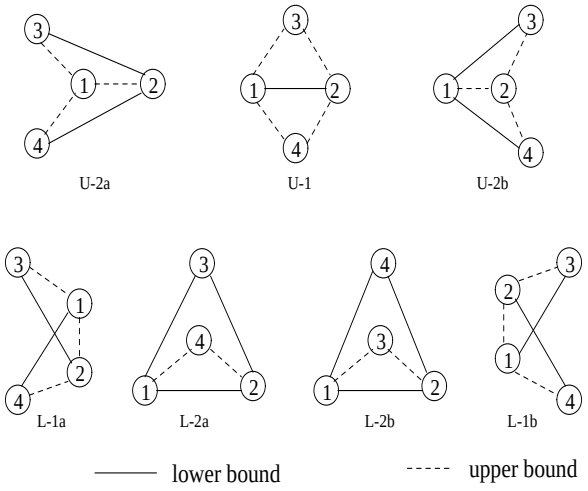

Figure 1. The combination of bounds that must be checked for obtaining upper & lower limits on the $(3,4)$-distance. The solid lines indicate lower bounds and the dashed, upper bounds. For example, in the configuration designated U-2a, a possible limit on the $(3,4)$ distance is calculated by setting the distance $d_{ij} = u_{ij}$ for those pairs (i,j) connected by dashed lines, $d_{ij} = l_{ij}$ for those connected by solid lines, and computing the maximum value d_{34} can assume under these settings.

Procedure *Tetra* (n, Tol)
{ n is the number of elements;
Tol is a user specified tolerance }
1. Initialize upper/lower bounds for pairwise distances
2. **repeat**
3. **for** each of $\binom{n}{4}$ vertices in lex. order **do**
4. **for** each of the six distances of the vertex **do**
5. Apply tetrangle inequalities (Figure 1)
to upper/lower bounds of the distance
7. **end for**
8. **end for**
9. **until** the largest change in any bound is less than Tol

Figure 2. Sequential algorithm for tetrangle-inequality bound-smoothing

Procedure *SeqColor* (n, p)
{ Generate proper coloring of vertices of graph $G(n)$ }
{ with each color applied to at most p vertices }
1. $C := 0$
2. $packing :=$ Empty
3. $k :=$# of equivalence classes induced by group $\mathcal{G}(n)$
4. $i := 0$
5. **for** $j := 1$ to k **do**
6. get the jth equivalence class $\mathcal{E}$, in lexicographic order
7. **if** $(i < p)$ **and** $(\exists v \in \mathcal{E}$ with no edge in $packing)$ **then**
8. $packing := packing \cup \{v\}$
9. $i := i + 1$
10. **else**
11. **for** $r := 0$ to $n - 1$ **do**
12. for each u in $\pi_r(packing)$, set $color[u] = C$
13. $C := C + 1$
14. **end for**
15. $packing := \{v\}$
16. $i := 0$
17. **end if**
18. **end for**
19. return *color*

Figure 3. Algorithm for coloring the vertices of the graph G(n)

Table 1. Molecules Studied

Mol.	No. Atoms	No. Measured Distances
RNA	128	201
DNA	222	467
MIP-2	430	1118

Table 2. Parallel vs Sequential Time for the three molecules

Mol.	No. Iterations	Par. Time†	Seq. Time†
RNA	10	7	110
DNA	8	57	1015
MIP-2	5	632	14483

†Per Iteration (in minutes)

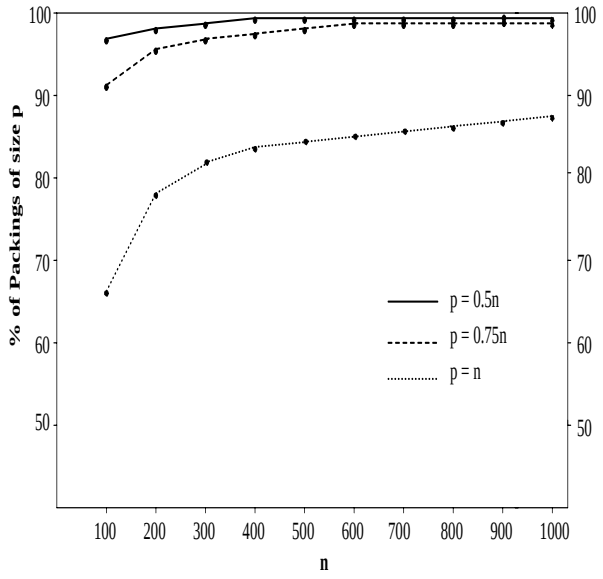

Figure 4. Percentage of 2-$(n, 4, 1)$ packings of size p

Procedure *ParColor* (n, p)
{ $G(n)$ is the graph to be colored }
{ p ithe number of processors ($p \leq n$) }
1. $nextProc := 1$
2. $packing$:=Empty
3. **for** the next p equiv. classes in lex. order **do**
4. get the next class, $\mathcal{E}$
5. **for** each processor i **do in parallel**
6. **for** $k := i, i+p, \ldots, i + \lfloor \frac{n-1}{p} \rfloor p$ **do**
7. $v :=$ kth vertex in $\mathcal{E}$
8. **if** (v has no edge in $packing$) **then**
9. $flag := 1$
10. **end if**
11. **end for**
12. **end for**
13. find processor i with least id that has $flag$ set to 1
14. **if** no proc. has $flag = 1$ **or** $nextProc = p$ **then**
15. return
16. **end if**
17. $packing := packing \cup \{v\}$
18. send vertex v from proc. i to proc. $nextProc$
19. $nextProc := nextProc + 1$
20. **end for**

Figure 5. Parallel algorithm for coloring the graph $G(n)$

Procedure *ParTetra* (n, p, Tol)
{ n is no. of elements; p is no. of processors ($p \leq n$)}
{Tol is a user specified tolerance }
1. Initialize upper/lower bounds for all pairwise distances
2. **repeat**
3. **while** there are more equivalence classes **do**
4. Invoke *ParColor*
5. **for** each processor i with a vertex, q_i **do in parallel**
6. **for** $k := 1$ to n **do**
7. Tighten bounds for all six distances in $\pi_k(q_i)$ using tetrangle inequalities (Figure 1)
8. **end for**
9. **end for**
10. **end while**
11. **until** the largest change in any bound is less than Tol

Figure 6. Parallel algorithm for tetrangle-inequality bound-smoothing

Table 3. Results of Bound-Smoothing algorithms for the three molecules

Mol.	Triangle			Tetrangle		
	N_{ub} †	r ‡	N_{lr} §	N_{ub}	r	N_{lr}
RNA	322	55.09	2	1264	52.95	54
DNA	543	67.80	6	5678	62.34	83
MIP2	2347	46.19	11	19182	45.23	124

† N_{ub} is the no. of upper bounds improved by the algorithm
‡ r is the root mean square gap in Å
§ N_{lr} is the no. long-range distances for which tight bounds were found

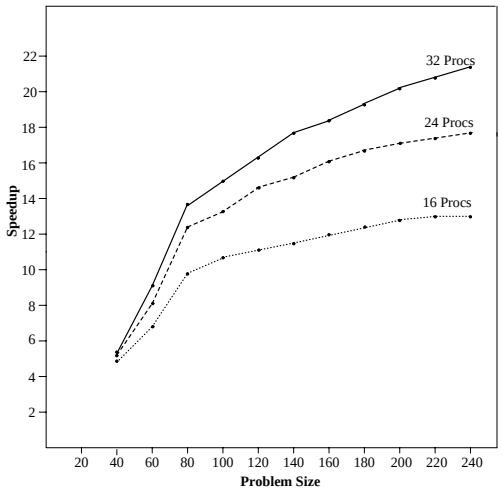

Figure 7. Speedup vs problem size