An efficient clash detection method for molecular structures applications

Miguel M. F. Bugalho * Arlindo L. Oliveira

October 12, 2007

Abstract

Molecular structures applications are usually computationally intensive. Problems like protein docking, ab-initio protein folding and even some visualization software that verify structure correctness, need to constantly determine if two atoms in the structure collide. This problem is usually referred as clash detection. If for the ab-initio and docking problems the cost of these computations determines the number of configurations that can be searched, for the visualization program the cost determines the application response time. Either way, a fast clash detection method can greatly improve the application effectiveness.

This work focus mainly in the ab-initio protein folding problem. Several protein folding methods avoid collisions by using energy functions that consider the Van der Walls forces. For other methods, which apply less expensive scoring functions, a faster clash detection algorithm may greatly increase the overall time efficiency. This will allow for more structures to be considered and thus improve the chances of finding the right structure.

We propose an efficient data structure with which we can achieve constant time clash detection, independently of the size of the protein. We compare the proposed data structure with one of the best known general data structures for this type of problems (range queries) and with the naive approach. The results show that the proposed data structure surpasses the other techniques for any protein size. The proposed data structure takes near half the time of the general data structure and close to a fifth of the time of the naive approach for the larger proteins.

1 Introduction

Clash detection algorithms can be applied to many molecular structure problems, *e.g.* protein docking [26, 17, 21], molecular visualization [25, 10], protein folding[11]. In this work we will focus in the ab-initio protein folding problem. Ab-initio protein folding consists in determining the structure of a protein using only the information of its amino acid sequence. It is a very hard problem that has been proved to be NP-Complete even for extremely simplified versions [16, 9, 3, 18]. For that reason, ab-initio structure prediction of a protein is a time consuming task where improvements on the complexity of the base computations can yield a significative difference in the final results.

The ab-initio algorithms usually generate many conformations and apply an energy or scoring function to determine if the conformation has protein like properties. Some of those

*Supported by the Portuguese Science and Technology Foundation by grant SFRH/BD/13215/2003 and project Biogrid POSI/SRI/47778/2002

properties consist in the propensity for some groups of atoms to be close together or to form sub-structures. Although these properties provide information on the correction of the protein structure, many restrictions exist to the conformations: strict bond and angle limits, less strict dihedral angles limits and the problem of avoiding clashes between atoms. The structural limits have low computational costs since they can be included in the construction of the conformations. That leaves us with the clash detection problem. Since this problem can become the bottleneck for many ab-initio methods, an improvement in its calculation method can have a great impact on the efficiency of those algorithms.

The clash detection problem is a particular case of the well known problem of range query in a metric space. Given a position, a range query consists in determining which points are closer than a predefined distance (range). In the clash detection problem, each point determines the position of an atom and the range is the distance for which such atom is considered to be in collision with other atom. Several algorithms have been proposed for the range query problem (for a comprehensive review see the work of Chavez et al.[6]), especially for cases where the distance calculations are expensive. However, distance calculations in the three dimensional Euclidean space are not particulary expensive, and these algorithms only perform better than the naive approach for very large proteins.

In this work we propose an efficient clash detection data structure that enables the use of more complex and precise models with a lower cost. We will focus our analysis on all heavy atoms models. Nevertheless, this technique may be used with less detailed models. The proposed data structure uses a discrete lattice model of the space where the protein is, to determine the atom relative positions to each other and efficiently detect clashes. This approach has some similarities with the LSR techniques (Locality Sensitive Hashing [1]), although the LSR is a generic range query technique that does not take advantage of the constrains imposed by this problem.

1.1 Paper Organization

In section 2 we describe the clash detection problem in detail and introduce the main concepts used in this paper. In section 3 we review the existing data structures that are applicable to this problem.

Section 4 describes the newly proposed data structure. In section 5 we present and compare the results obtained for the proposed data structure, the naive method and one of the best existing range generic data structures.

Finally, section 6 summarizes the conclusions and discusses repercussions and possible future work.

2 Clash Detection Problem

As mentioned before the clash detection problem is a particular case of the range query problem. There are, however, several characteristics in the clash detection problem that can be explored to obtain better results. First, the clash detection problem has low dimensionality. Many range query data structures focus on problems with greater dimensionality or in database systems. In both cases the distance calculations computational cost is much higher. Although every range query algorithm may be used with lower dimensionality, the efficiency of each data structure will be different for this case. The second characteristic is the dynamic nature of this problem. In ab-inito protein folding several atom positions will be modified

each time the structure is changed to improve an energy function or to avoid a collision. There are also other characteristics, like the fact that we need only to determine whether a collision exists or not (and not every atom that collides), that may or may not have an impact in the data structure efficiency.

To determine if two atoms collide we need first to compute the minimum allowed distance between those two atoms, *i.e.* determine the range of the query. To compute this distance for two atoms we use the Lennard-Jones potential (equation 1). For this purpose the Lennard-Jones potential represents the repulsion of the two atoms. The Lennard-Jones potential is related to the forces of attraction/repulsion caused by the electrons. Nevertheless the attraction forces are small, compared with other forces, while the repulsion forces can be very strong. When two atoms are too close, the repulsion forces are very high and the atoms are considered to be in collision. Thus the clash detection is used to determine if the structure is physically possible.

We used the Lennard-Jones parameters from the amber99 force field [28, 8]. By choosing an energy threshold, we can calculate the distance from the Lennard-Jones potential. Since the energy grows very rapidly with the approximation of the two atoms we can use a large energy value as a threshold. We chose a value of 1000 Kcal/mole for the energy threshold, allowing for some flexibility in the atom positioning that can then be adjusted in the final refinement steps:

$$Evdw = \epsilon * \left[\left(\frac{rad_{ij}}{dist_{ij}} \right)^{12} - 2 * \left(\frac{rad_{ij}}{dist_{ij}} \right)^{12} \right] \quad (1)$$

Where ϵ is a constant, rad_{ij} is an atom radius parameter that depends on the type of atoms i and j, and $dist_{ij}$ is the distance between atoms i and j. As we only consider heavy atoms, we have to increase the normal radius clash size of the carbon atoms to include the hydrogens. We only do this for carbon atoms since, for other atoms with hydrogen bonds, the hydrogen clash sphere is almost completely inside the heavy atoms sphere. Table 1 shows some thresholds values for pairs of atoms.

Atom Pair	Threshold Size
C-C	2,25
C-N	2,02
C-O	1.96
C-S	2,13
N-O	1,7

Table 1: Some examples, in angstrom, of threshold sizes for pair of atoms using the free energy threshold of 1000 in the Lennard Jones potential equation 1.

We used a further simplification to improve efficiency, that consists in using only some well positioned atoms for clash detection. In a protein, each atom has a limited range of positions relatively to the other atoms that is forced by bond lengths, angles and even, in a limited way, dihedral angles. For bond length and angle values the variation is usually so low that we will consider those values to be constant.

The example in figure 1 shows that the distance threshold, presented in some atoms as a circle, is larger than the bonds between the atoms. This would imply that every pair of

bonded atoms would be considered to be in collision. This will not happen because we only consider collisions for atoms that are more than three bonds apart. It is possible to do this because the bonds energy is defined by different potentials. A second observation is related with the fact that, when scanning the area of each atom for collisions some areas will be repeated. When placing this structure, we only need to scan the area that it will occupy, but if we do it atom by atom, we will repeatedly scan some areas. This happens because the scan area of each atom overlaps with some of the other scan areas. To avoid this we can use only some of the atoms of the structure. The dark circles present the scanned area of one of the possible set of atoms. We can see that the covered area of the reduced set is almost equal to the structure area.

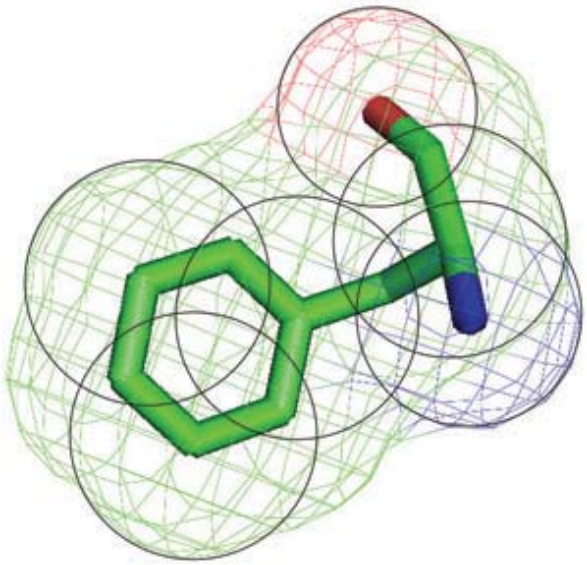


Figure 1: Image of the electronic cloud formed by the eleven atoms of the phenylalanine amino acid (without the hydrogens). The dark circles represent the area covered by six chosen atoms of the amino acid atoms. There is little difference between the two covered areas.

The method for using this reduced set on the clash detection is very simple. First, for each amino-acid, a set of atoms were chosen to represent the structure. Roughly speaking, we choose each atom that is two bonds apart of an already chosen atom, but sometimes we have to correct this set to improve the covered area. Instead of doing a clash detection atom by atom we do an amino-acid clash detection. During clash detection for that amino-acid we use only the reduced set. Each area from the reduced set of atoms is scanned to see if an previous added atom is already at that position. Since the area covered by the represented atoms overlaps almost all the area covered by the non represented atoms, we can be confident that no collision would be found by scanning also the non represented atoms. Any collision in the non represented atoms would also collide with the represented ones, except for some few cases in the outer shell of the structure covered area, where the covered area of the two set of atoms do not overlap. This cases have little impact since they would never imply serious errors, like overlapping entire structures, and also because they can be easily corrected in refinement steps. After the clash detection, if no clash occurs, all the atoms of the amino-acid are added to the structure.

3 Range Query Data Structures

The naive solution to detect an atom clash is to measure the distance of that atom to every other atom and verify if it is below a given threshold. More efficient methods use data structures that efficiently calculate the nearest neighbors. There are various data structures that address the problem of finding a nearest neighbor given a distance measure. Many of these data structures consider that there is little or no variation on the set of points used to calculate the range queries. Without insertions and deletions they focus only on the improvement of the nearest neighbor query. In this problem, however, the set of points varies each time we try a different backbone or side chain conformation.

Table 2 shows some common data structures for the computation of nearest neighbors. For an overview of these data structures see the work of Chavez et al.[6]. From this work we can see that the FQ family, the LAESA/AESA, and the M Tree type data structures support dynamic capabilities.

Another data structure that also supports dynamic capabilities is the Dynamic Spatial Approximation Tree[23] (DSAT or Dynamic SAT), based on the Spatial Approximation Tree[22] (SAT) data structure. The SAT is one of the most efficient Range Query data structures and it was shown by Navarro et al. ([23]) that the Dynamic SAT can incorporate dynamic capabilities with little deterioration of this efficiency.

We have therefore chosen the Dynamic SAT data structure for comparing with the one proposed in this work for the following reasons:

- The FQ family is for discrete distances only. Since there are specific data structures for continuous problems we saw no need to add a discretization step.
- The M Tree type data structures were developed for data base queries and consider that the cost of making one distance calculation is much higher than other computations, which is not the case. These data structures usually need expensive methods for keeping the tree balanced.
- Since in this problem the number of queries is proportional to the number of insertions and deletions (clash detection is only performed when one amino-acid is added or changed in the structure), the AESA data structure has no advantages, because it needs to have all the distances between atoms pre-computed. The LAESA data structure only needs a fixed number of distance calculations. However the problem of choosing the pivots is hard in this application. If we choose the first atoms, they would be clustered in one place. Any other set would need a dynamic way of changing the pivots that would probably degrade the efficiency.
- The Dynamic SAT was initially created to allow an incremental construction which is adequate to this problem. Additionally, the deletion needs for this problem are mainly for atoms that were set last, which reduces the need of re-insertions (see the work of Navarro et al. [23] for a description of the technique used for deleting elements).

3.1 Dynamic Spatial Approximation Trees

The SAT uses a tree representation of the points and spatial approximation to organize the tree structure. Each root of a tree or sub-tree is connected to a set of neighbors, its children,

Name	Description
BKT, FQT, FHQT, FQA	Trees for discrete distance functions
VPT, MVPT, VPF, BST, GHT, GNAT, VT	Continuous distance trees that use a pivoting technique
M Tree [7], Slim Tree [27], R* Tree [2]	Tree structures created for data base queries with dynamic capabilities
SAT[22], Dynamic SAT [23]	Spatial approximation tree, each child in the tree represents a neighbor
AESA, LAESA	Elimination algorithms
LSH [1]	Locally sensitive hashing

Table 2: Range queries and nearest neighbors data structures. For an overview of these data structures see the work of Chavez et al. [6]. For the data structures not present in this work a specific reference is presented in the table.

where each of them is closer to the root than to each other. Also, an element only becomes a child of a node if it is closer to that particular sub-tree root than to any previous element. That means that no limit is imposed to the tree arity. A range query is done by descending this tree and searching for nodes that are closer than the given range. With this tree representation the search can be pruned using the triangle inequality:

$$d(A,C) \leq d(A,B) + d(B,C) \tag{2}$$

One of the pruning conditions for the search uses the maximum distance to the descendent nodes that we will refer as radius. Consider *Radius* as this distance, *Range* as the range of the query and $d(r,q)$ as the distance between the root of the sub-tree to the query. Now consider that we have $d(r,q) > Radius + Range$, this means that for any node a in the sub-tree where r is root we have:

$$\begin{aligned} &d(r,q) > Radius + Range \\ &\text{Using the triangle inequality } 2 \Rightarrow d(a,r) + d(a,q) > Radius + Range \\ &\text{Since } d(a,r) \leq Radius \Rightarrow d(a,q) > Range \end{aligned} \tag{3}$$

In conclusion, using the maximum distance to the descendent nodes *Radius*, if $d(r,q) > Radius + Range$ the sub-tree can pruned since no node will be in the query range.

Another pruning condition is given by the restrictions that an element only becomes a child of a node if it is closer to that particular sub-tree root than to any previous element. This means that for every element x of the sub-tree rooted by a, $d(a,x) < d(b,x)$ for every other node that does not belong to that sub-tree. Consider that during the range query, we have r has a root of the sub-tree and $Child(r)$ as its children. Consider also that c is the closest node to the query of all those nodes. If for some node $a \in Child(r)$ we have that $d(a,q) > d(c,q) + 2 * Range$, then, for every node x in the sub-tree rooted by a, we have:

$$\begin{aligned} &d(a,q) > d(c,q) + 2 \times Range \\ &\text{by equation 2: } d(a,q) \leq d(a,x) + d(x,q) \text{ so} \\ &d(a,x) + d(x,q) > d(c,q) + 2 \times Range \end{aligned} \tag{4}$$

Since c does not belong to the sub-tree rooted by$a$ we have $d(a,x) < d(c,x)$ so
$d(c,x) + d(x,q) > d(c,q) + 2 \times Range \Leftrightarrow d(c,x) - d(c,q) + d(x,q) > 2 \times Range$
by equation 2: $d(c,x) \leq d(c,q) + d(x,q) \Leftrightarrow d(c,x) - d(c,q) \leq d(x,q)$ so
$d(x,q) + d(x,q) > 2 \times Range \Leftrightarrow d(x,q) > Range$

This means that we can prune every sub-tree with root a where $d(a,q) > d(c,q)+2*Range$, since no node in that sub-tree will be in the query range. Figure 2 shows a image from the original paper of Gonzalo et al.[23] that exemplifies a range query.

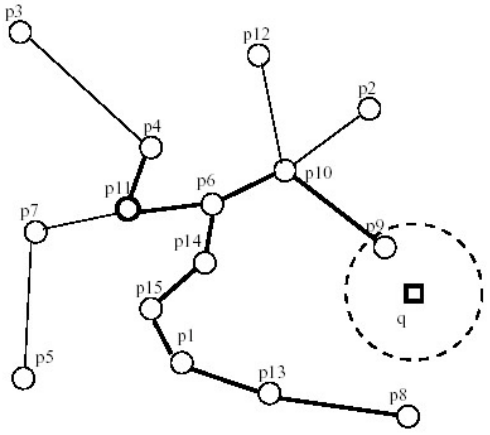


Figure 2: Example of a range query in the *sa-tree* (original figure from the work of Navarro et al. [23]). The search descends trough the tree until it finds element p9, elements p11 and p4 are searched because the inequality 4 holds.

For the Dynamic implementation two new concepts were used. First, a limit on the arity, and second, the usage of a timestamp. Although the limit is not required for the tree to work, for low dimensional spaces, like in this problem, it improves the performance [23].

In the insertion, the same decisions are made with the exception of forcing the element to choose the nearest child to descend if the maximum arity is reached, even if the root is closer. Additionally a timestamp is saved for each node. During the range search the biggest difference between the two data structures is related to the fact that, when a element x was inserted, the elements with higher values of the timestamp did not exist and were not considered.

The Fully Dynamic implementation of the SAT also permits deletion of elements. However this feature requires some elements to be re-inserted in the tree. In some applications we might want to remove an element in the middle of the tree. For instance, we may need to use these deletions if we want to correct a side chain to avoid a clash without changing every atom added afterwards. However, in most cases, the deletions happen with the last elements that were inserted.

For the purpose of comparing with our data structure we only allowed deletions of the last elements. This corresponds to deletions of the leafs of the tree which are easily implemented. However, the radius information cannot be fully corrected.

We tried a simple technique for the radius correction based on the M-trees [7]. We first verify if the element affects the radius by checking if the radius plus the distance to the root

is higher than the father radius. If it is not, then this is not the node that defines the father radius and no change must be made. Otherwise we calculate the maximum of $radius + distance\_to\_root$ and if the value is less than the actual radius, the radius is decremented. Each time we modify a radius we repeat the process on the parent node of the changed node.

We also tried an optimization using a different distance metric. Instead of calculating the distance as $\sqrt{(x0-x1)^2+(y0-y1)^2+(z0-z1)^2}$, we used $max\{|x0-x1|, |y0-y1|, |z0-z1|\}$. This avoids multiplications and the square root. However, using this distance, we make an approximation by excess, since instead of verifying a sphere of radius r we are verifying a cube of side $2 \cdot r$. Nevertheless, an hybrid measure can be created by using the optimized measure as a first limit and a the exact measure only to confirm a clash. This way we can still have some optimization and no error is made.

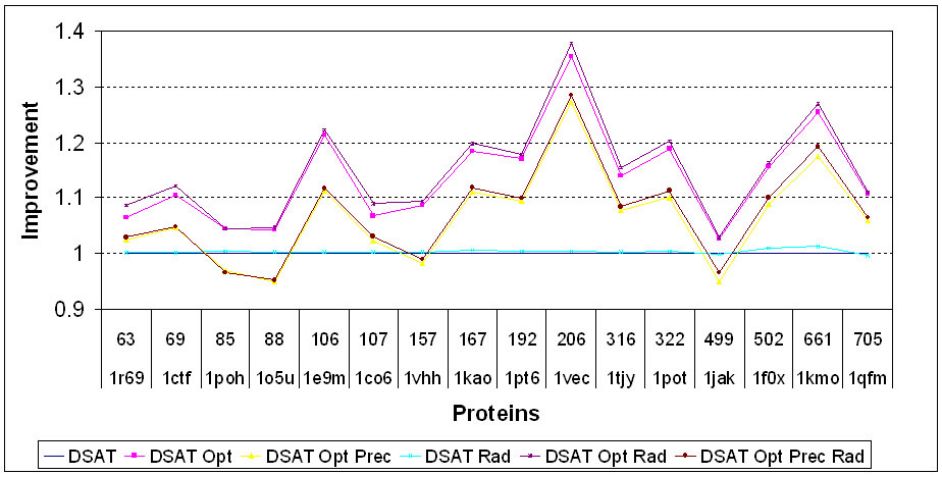


Figure 3: Performance comparison of the Dynamic SAT implementations. The values show the time improvement over the base data structure. The blue line represents the base data structure, therefore has always a value of 1.

Figure 3 shows the performance of Dynamic SAT implementations with proteins of increasing size. We use *Opt* to refer to the implementations that use the optimized distance measure, *Opt Exact* for those that use the hybrid distance measure and *Rad* for those that use the radius correction. For a better description of the data and the testing process see the results section (section 5).

We can see some improvement from the usage of the optimized distance function. In the inexact version a mean improvement of 13,7% is achieved, while in the exact version the improvement decreases to less than half (6.5%). For the radius correction the improvements were negligible (14,9% mean for the inexact version and 7.2% mean for the exact version). Since this is not the subject of this work we chose not to pursue further improvements and analysis of the data structure.

4 Geometric Hashing Method

In this section we propose a data structure that has constant time for setting an element and also for querying for a clash. All placed atoms are saved in a three dimensional array, where each slot corresponds to a three dimensional position (figure 4). Using this structure, placing

an atom corresponds to setting the array slot and determining a clash is made by iterating through the nearest slots.

Each slot of the array corresponds to a small cube in the space. The smaller this cube, the more precise is the distance approximation. We tested cubes with an edge side of 0.7 and 0.5 angstroms. This not only permits a good precision but also ensures the impossibility of two atoms being in the same slot. The maximum cube diagonal will be $0.7^3 = 1.221$ and the minimum bond length between two atoms is 1.229 (between Carbon and Oxygen atoms). Having only one atom in each slot permits some implementation optimizations.

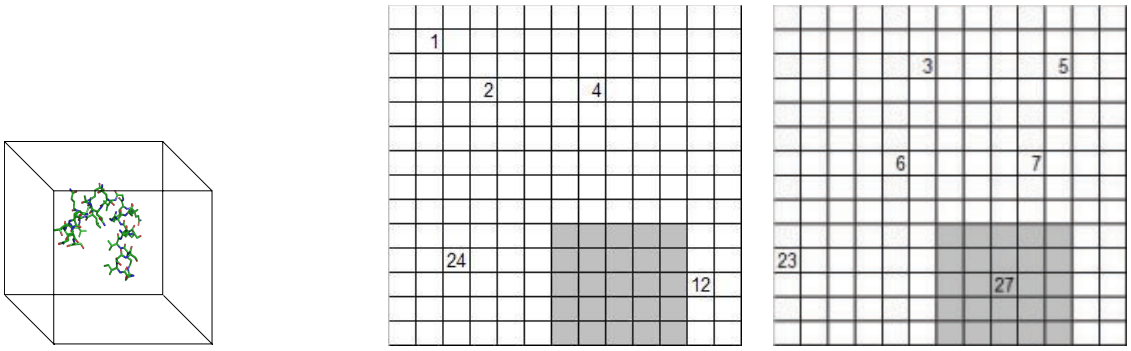


Figure 4: Example of setting an atom in the three dimensional array. The left figure shows the fragment that was already saved. The two other figures represent two planes in the three dimensional array with some of the slots already occupied by atoms of the fragment. The gray area is an example of the area that would be scanned to check if any clash would occur when the new atom (atom 27) is inserted in the array.

The first information required to detect a clash is the maximum distance to scan when checking for clashes on an atom insertion. Using the distance thresholds presented in table 1 we calculate, for each atom, the maximum distance threshold. Searching as far as this distance during an insertion will assure that no clash occurs. However a clash does not occur every time an atom is found in the search. There may be atoms where the pair threshold is smaller than the maximum threshold. When an atom is found in the search, the distance between that atom and the atom being inserted must be checked against the pair threshold of those atoms.

A simple way of detecting the clash in the three dimensional array is to scan a cube centered in the position of the atom. The size of the cube will be two times the maximum threshold. The problem with this simple approach is that the vertices of the cube are much further away than the center of its faces. In this work we used an approximation to a sphere.

To avoid repeating calculations, two data structures are saved:

- The *sphere limits* data structure. A set of three tables with the x, y, and z coordinates limits for the sphere approximation. This is used to determine the limits during the scan.
- The *cell distances* table. A table with the distances of each cell to the center. Each time an atom is found during the scan, its distance to the atom in the center can be

retrieved from this table and checked against the pair thresholds. Instead of calculating the distance between the two atoms a pre-calculated approximation for this distance is used.

The *sphere limits* data structure gives us, for each possible threshold value, the number of the neighboring cells to be scanned in the three dimensional table. The tables save the limit values for all possible values of the distance threshold in increments of 0.1 angstroms. The maximum value in the table corresponds to the maximum threshold value for any pair of atoms.

The x coordinate range is given by the table that converts the threshold values to the 0.5-0.7 angstroms discretization of the three dimensional array. This gives us how many cells must be covered by the scan for that particular threshold. The y and z coordinate range can be calculated using equation 5 for y and equation 6 for z. Notice that r is presented in 0.1 angstroms increments and x and y correspond to the number of cells that have a size of 0.5 or 0.7 angstroms. The values of x,y and r must be converted to the same unit of measurement(*e.g.* angstrom). Figure 5 shows some examples of the saved values.

$$x^2 + y^2 + z^2 = r^2$$

$$y = \sqrt{r^2 - x^2} \qquad \text{When z} = 0 \tag{5}$$

$$z = \sqrt{r^2 - x^2 - y^2} \tag{6}$$

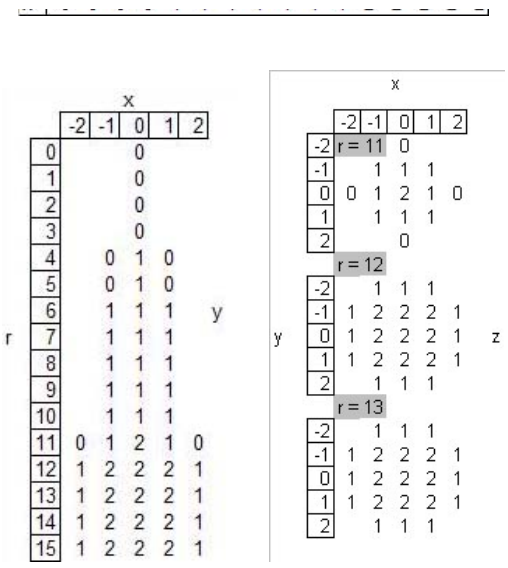
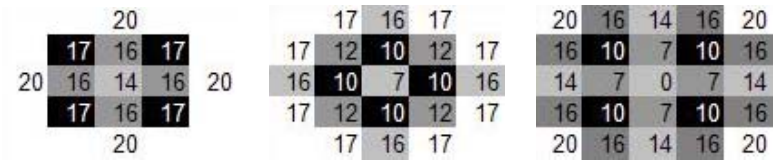


Figure 5: Examples of the tables with limits to the clash scan for 0.7 angstrom cell size. The top table shows the limits for the x coordinate range given the threshold r. The left table shows the y limits given x and r, and the right table the z limits given x, r, and y. While x, y and z correspond to the number of cells to be scanned, r corresponds to the distance in 0.1 angstrom units.

The *cell distances* table is used to determine the distance of two atoms without the need of actually doing the calculations. Of course this value will be an approximation since we

consider that both the atoms will be in the center of their cells. This information is then used to check if an atom is closer than the corresponding pair threshold (since the pair threshold might be different from the maximum threshold used as the scan radius). Figure 6 shows an example of the table with the distances of each cell to the center. The distances are also presented in 0.1 angstrom units.



Outer Shell Center

Figure 6: Example of the three dimensional distance matrix (maximum threshold of 2 angstrom). Considering that the atom to be set is in the middle of the cube, the table gives us a distance approximation of every other atom in the vicinity. The distance is presented in 0.1 angstrom units.

When a non empty slot is found during the scan, the distance in the three dimensional distance matrix (figure 6) is compared with the pair threshold (table 1). If the distance is smaller than the threshold, a clash has been detected.

This method is approximate since we consider that each atom is in the center of the cube. In the worst case, an error equal to the diagonal of the cube may occur. In practice this error does not occur often in decisive areas. Since the atoms are connected in a rigid structure, only some atoms on the outside of that rigid structures may be disregarded or wrongly considered as clashing atoms. Nevertheless, we can always modify the algorithm so that there is no error. First we enlarge the radius of the search to contain every possible clash atom. Since the maximum error is the cell size we need to increase the radius by one cell. We then confirm each clash by calculating the atoms distance and verifying the pair threshold. The first step will assure us completion and the second correction. Algorithm 1 presents the pseudo-code for the presented method. Comments are also presented for the exact version modifications.

Figure 7 compares the performance of exact and approximate implementations for a cell size of 0.5 and 0.7 angstrom. As for the previous data structure the results are presented as an improvement over the base implementation. We considered the inexact implementation with 0.5 cell size as the base algorithm. We use *Exact* to refer to the exact implementations *i.e.* the implementations that use the extended radius and distance calculation.

As we can see in the results, the usage of a larger cell improves the time efficiency (8.2% improvement). However, it also decreases the precision of the method. It is also possible to see that there is no advantage in using an exact solution with smaller cell size, since both methods are exact and the larger cell has best time performance. In fact, the time results of the best precise solution are comparable with the inexact solution with 0.5 angstrom cell size.

We can also observe a decrease on the differences between each implementation, when the protein size increases. The results start from a difference to the base algorithm of more than 10% and end near the 5% value. Although we present no proof, we believe that the occupancy of the hash table might explain this fact. When no clash is found, the time spent in setting the atoms on the hash table is much smaller than the time spent in searching for neighbors. However, if more nearby atoms are set in the table, the neighbors search becomes faster since the algorithm stops as soon as a clash is found. Since the setting time is equal

Algorithm 1 Pseudo code for the proposed clash detection method.

Require: **geom** (hash table), **dist** (distances to the center slot), **sphereX**, **sphereY**, **sphereZ** (scan sphere limits), **maxthresh**, **pairthresh**

```
1: procedure ClashTest(atom)
2:     for x = -sphereX(maxthresh(atom)) to sphereX(maxthresh(atom)) do ▷ Exact
           version → All sphere limits are increased by one
3:         for y = -sphereY(maxthresh(atom),x) to sphereY(maxthresh(atom),x) do
4:             for z = -sphereZ(maxthresh(atom),z) to sphereY(maxthresh(atom),z) do
5:                 atom2 = geom(x + posX,y + posY,z + posZ)        ▷ posX,posY,posZ →
                       coordinates in geom where the atom will be added
6:                 if Exists(atom2) then
7:                     if dist(x,y,z) < pairthresh(atom,atom2) then      ▷ Exact version →
                           Instead of dist we have the distance calculation between atoms
8:                         A clash was detected
1: procedure AddAminoAcid(AminoAcid)
2:     noclash = true
3:     for atom in reducedAtomSet(AminoAcid) do    ▷ Test only the reduced set of atoms
           described in section 2
4:         if ClahsTest(atom) then
5:             noclash = false
6:     if noclash then
7:         Add all AminoAcid atoms to geom
8:     else
9:         Clash found, the structure is not physically possible
```

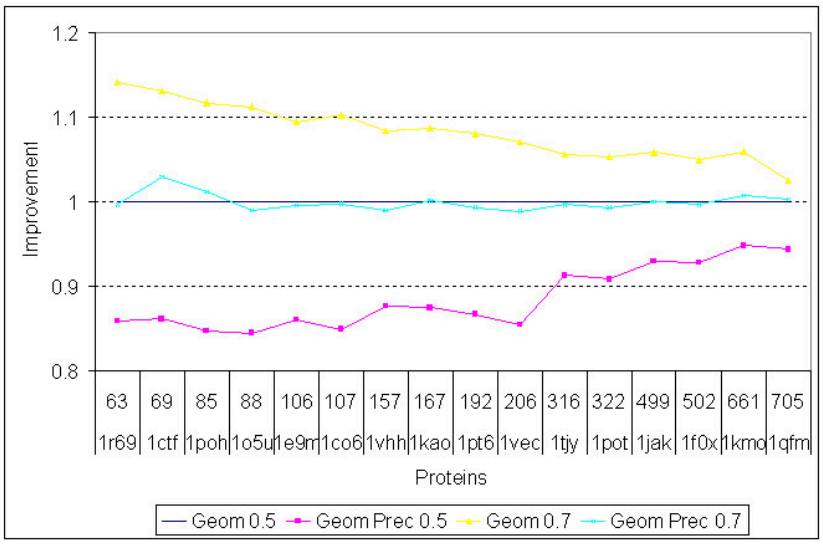


Figure 7: Performance comparison of the Geometric Hashing implementations

for every implementation, this increase in the number of clashes may explain the decrease in the overall time differences.

5 Results

To test the clash detection data structures we compiled a set of protein structures of increasing size (number of amino-acids). The proteins also differ in terms of secondary structure composition (Alpha Beta, Mainly Alpha and Mainly Beta proteins). We have chosen the proteins from a list compiled by the What If database [20](official site [29]). This database uses a filter similar to the PDB_SELECT program [5, 19] to select a set of proteins from the PDB database [4] (official site [30]). The filter uses a similarity threshold and a minimum resolution. Table 3 shows the set of chosen proteins.

Each data structure is tested by executing a search in a discrete state search space. In a discrete state model the atom bonds and angles are fixed, and the phi and psi dihedral angles are chosen from a limited set of values. We use the best discrete state model of four states (phi and psi values of {(-63,-63),(-132,115),(-42,-41),(-44,127)}) from the work of Park et al. [24]. For the side chains we use the Dunbrack Backbone-Dependent Rotamer Library [12, 13, 14, 15].

The search used in this test emulates a normal ab-initio protein folding algorithm. The difference is that instead of using a scoring function we use the actual root mean square distance to the native protein. We use a best first search and backtrack each time the root mean square distance exceeds 5 angstroms. When a structure is found we restart to a previous choice until 100 searches are completed.

In the majority of the ab-initio algorithms several structures are generated until a final structure is reached. These algorithms normally use atomic contact information as the scoring function. Although for this test we do not need the contact information, we will also focus more on the part of the search that has more atoms and therefore more information. Since a search tree structure grows exponentially, this is also the part of the search that has more

Name	Size	Type
1r69	63	Mainly Alpha
1ctf	69	Alpha Beta
1poh	85	Alpha Beta
1o5u	88	All Beta (beta helix)
1e9m	106	Alpha Beta
1co6	107	Mainly Alpha
1vhh	157	Alpha Beta
1kao	167	Alpha Beta
1pt6	192	Alpha Beta
1vec	206	Alpha Beta
1tjy	316	Alpha Beta
1pot	322	Alpha Beta
1jak	499	Alpha Beta
1f0x	502	Alpha Beta
1kmo	661	Mainly Beta
1qfm	705	Alpha Beta

Table 3: Protein data set

possible conformations. Therefore, the backtracks are made in increments of 10%. The first backtrack will be to an amino acid at 90% of the chain, the second to 80%, until 10% and then again to 90%.

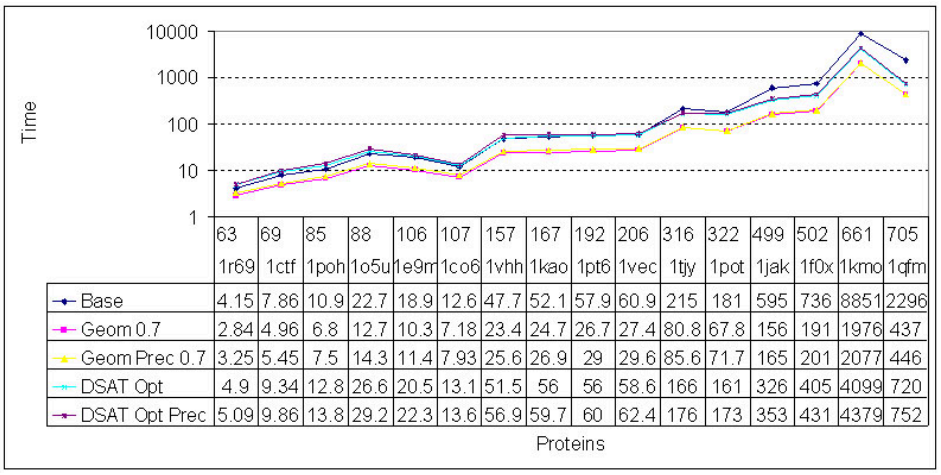
Each time an amino acid is set, the clash detection data structure is used, but no backtrack is made. Since we want for each data structure to have the same number of tests, and not every method used in those data structures is exact or does the same approximation, we cannot use the clash detection to influence the search.

Figures 8 and 9 show the results for the two data structures using the approximate and exact versions.

From the logarithmic scale graphic on figure 8 we can see the execution time differences for different proteins. The results show that the test algorithm running time for each method does not depend only on the size of the protein but also on the protein structures. For instance, for beta type proteins (1o5u and 1kmo) the time increases more than the expected for the size variation, whereas for alpha proteins (1r69 and 1co6) the time decreases slightly. These results are consistent and can be explained from the results obtained by Park et al. [24] for the discrete state model used in this work. In that work the alpha structures where shown to be easier to model than the beta structures. The time values presented on the table show that the test algorithm run time grows rapidly with the size of the protein.

From figure 9 we can see that the geometric hashing data structure outperforms the Dynamic SAT data structure and the naive approach. For the bigger proteins the search performed using geometric hashing is four to five times faster than that of the naive approach and roughly two times faster than the search performed by the Dynamic SAT. Additionally it can be observed that the proposed data structure has improvements for every protein size. The Dynamic SAT only outperforms the naive algorithm for bigger proteins, when the lower number of distance calculations compensates the cost of maintaining the tree structure.

We can also see that there is a small increase in the cost of using an exact method. However this cost does not increase with the size of the protein.



	63 1r69	69 1ctf	85 1poh	88 1o5u	106 1e9m	107 1co6	157 1vhh	167 1kao	192 1pt6	206 1vec	316 1tjy	322 1pot	499 1jak	502 1f0x	661 1kmo	705 1qfm
Base	4.15	7.86	10.9	22.7	18.9	12.6	47.7	52.1	57.9	60.9	215	181	595	736	8851	2296
Geom 0.7	2.84	4.96	6.8	12.7	10.3	7.18	23.4	24.7	26.7	27.4	80.8	67.8	156	191	1976	437
Geom Prec 0.7	3.25	5.45	7.5	14.3	11.4	7.93	25.6	26.9	29	29.6	85.6	71.7	165	201	2077	446
DSAT Opt	4.9	9.34	12.8	26.6	20.5	13.1	51.5	56	56	58.6	166	161	326	405	4099	720
DSAT Opt Prec	5.09	9.86	13.8	29.2	22.3	13.6	56.9	59.7	60	62.4	176	173	353	431	4379	752

Figure 8: Results for the clash detection algorithms in logarithmic scale

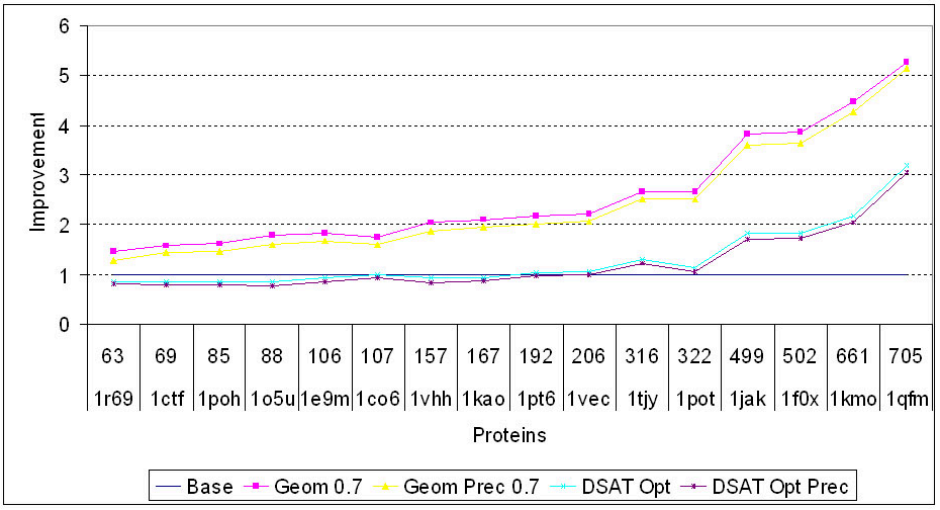


Figure 9: Results for the best clash detection methods for each data structure compared to the naive approach. The values correspond to the improvements over the naive approach.

6 Discussion and future work

The Dynamic SAT results show that the low dimensionality and the heavy dynamic nature of this problem create a difficult task for commonly used range query data structure.

The results also show that the geometric hash data structure takes good advantage of the low dimensionality of the problem. It creates a solution that is not greatly affected by the dynamic nature of the problem. The presented solution not only achieves better results for the biggest proteins, but also shows improvements over the other solutions for the smallest proteins.

We tested this work on the protein folding problem, which is known to be very difficult and with a high computational cost. However, without any constraints, the protein folding algorithms are normally used only for small proteins. Although the improvements described in this work can have a positive impact in this problem, even for small proteins, there are

also other problems where the impact may be even greater. Problems like protein docking or protein visualization can take advantage of the large improvements in bigger proteins. In the protein docking problem more than one protein is considered. Therefore, a large number of atoms must be tested for clashes during the proteins rotations and translations. The visual tools, although usually having a smaller computational cost, require a fast response time to be user friendly. These tools also may have to consider more than one structure of various sizes.

In future work we will analyze the application of this method to other problems. Further time and space improvements may also be found for this type of problems.

We have chosen to test well known range query algorithms. However, these algorithms may be surpassed by others in this low dimensional problem. Although other algorithms were analyzed in this work, we presented only the ones we considered more efficient and best suited for this problem. Future work may include other algorithms that were not considered in this one, specially when applying this method to other problems like protein docking or visualization.

References

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate aearest neighbor in high dimensions. *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 459–468, 2006.

[2] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. *ACM SIGMOD Record*, 19(2):322–331, 1990.

[3] B. Berger and T. Leighton. Protein folding in the hydrophobic-hydrophilic (HP) is NP-complete. *Proceedings of the Second Annual International Conference on Computational Molecular Biology*, pages 30–39, 1998.

[4] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, TN Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28(1):235–242, 2000.

[5] J. Boberg, T. Salakoski, and M. Vihinen. Selection of a representative set of structures from Brookhaven Protein Data Bank. *Proteins*, 14(2):265–76, 1992.

[6] E. Chávez, G. Navarro, R. Baeza-Yates, and J.L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[7] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. *Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 426–435, 1997.

[8] W.D. Cornell, P. Cieplak, C.I. Bayly, I.R. Gould, K.M. Merz, D.M. Ferguson, D.C. Spellmeyer, T. Fox, J.W. Caldwell, and P.A. Kollman. A Second Generation Force Field for the Simulation of Proteins, Nucleic Acids, and Organic Molecules. *Journal of the American Chemical Society*, 117(19):5179–5197, 1995.

[9] P. Crescenzi, D. Goldman, C.H. Papadimitriou, A. Piccolboni, and M. Yannakakis. On the complexity of protein folding. *Journal of Computational Biology*, 5(3):423–466, 1998.

[10] W.L. DeLano. The PyMOL molecular graphics system, 2002.

[11] KA DILL. Principles of protein folding–A perspective from simple exact models. *Protein Science*, 4(4):561–602, 1995.

[12] R.L. Dunbrack and M. Karplus. Conformational analysis of the backbone-dependent rotamer preferences of protein sidechains. *Nature Structural Biology*, 1(5):334–340, 1994.

[13] R.L. Dunbrack Jr. Rotamer libraries in the 21st century. *Current Opinion in Structural Biology*, 12(4):431–40, 2002.

[14] R.L. Dunbrack Jr and F.E. Cohen. Bayesian statistical analysis of protein side-chain rotamer preferences. *Protein Science*, 6(8):1661–1681.

[15] R.L. Dunbrack Jr and M. Karplus. Backbone-dependent rotamer library for proteins. Application to side-chain prediction. *Journal of Molecular Biology*, 230(2):543–74, 1993.

[16] A.S. Fraenkel. Complexity of protein folding. *Bulletin of Mathematical Biology*, 55(6):1199–1210, 1993.

[17] I. Halperin, B. Ma, H. Wolfson, and R. Nussinov. Principles of docking: an overview of search algorithms and a guide to scoring functions. *Proteins*, 47:409–443, 2002.

[18] W.E. Hart and S. Istrail. Robust proofs of NP-hardness for protein folding: general lattices and energy potentials. *Journal of Computational Biology*, 4(1):1–22, 1997.

[19] U. Hobohm and C. Sander. Enlarged representative set of protein structures. *Protein Science*, 3(3):522, 1994.

[20] R.W.W. Hooft, C. Sander, and G. Vriend. Verification of protein structures: side-chain planarity. *Journal of Applied Crystallography*, 29(6):714–716, 1996.

[21] S. Jones and J.M. Thornton. Principles of protein-protein interactions. *Proceedings of the National Academy of Sciences*, 93(1):13–20, 1996.

[22] G. Navarro. Searching in metric spaces by spatial approximation. *The VLDB Journal The International Journal on Very Large Data Bases*, 11(1):28–46, 2002.

[23] G. Navarro and N. Reyes. Fully dynamic spatial approximation trees. *Proceedings of the 9th International Symposium on String Processing and Information Retrieval (SPIRE 2002)*, pages 254–270.

[24] B.H. Park and M. Levitt. The complexity and accuracy of discrete state models of protein structure. *Journal of Molecular Biology*, 249:493–507, 1995.

[25] E.F. Pettersen, T.D. Goddard, C.C. Huang, G.S. Couch, D.M. Greenblatt, E.C. Meng, and T.E. Ferrin. UCSF ChimeraA visualization system for exploratory research and analysis. *Journal of Computational Chemistry*, 25(13):1605–1612, 2004.

[26] G.R. Smith and MJ Sternberg. Prediction of protein-protein interactions by docking methods. *Current Opinion in Structural Biology*, 12(1):28–35, 2002.

[27] C. Traina Jr, A. Traina, C. Faloutsos, and B. Seeger. Fast indexing and visualization of metric data sets using slim-trees. *IEEE Transactions on Knowledge and Data Engineering*, 14(2):244–260, 2002.

[28] J. Wang, P. Cieplak, and P.A. Kollman. How well does a restrained electrostatic potential(RESP) model perform in calculating conformational energies of organic and biological molecules? *Journal of Computational Chemistry*, 21(12):1049–1074, 2000.

[29] http://swift.cmbi.kun.nl/whatif/select/.

[30] http://www.rcsb.org/.