# Algorithm and Data Structures for Efficient Energy Maintenance during Monte Carlo Simulation of Proteins

Itay Lotan[1]  Fabian Schwarzer[1]  Dan halperin[2]  Jean-Claude Latombe[13]

December 4, 2003

[1] Department of Computer Science, Stanford University, Stanford, CA 94305, USA. E-mail: [itayl, schwarzf, latombe]@cs.stanford.edu

[2] School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E-mail: danha@post.tau.ac.il

[3] To whom correspondences should be addressed

## Abstract

Monte Carlo simulation (MCS) is a common methodology to compute pathways and thermodynamic properties of proteins. A simulation run is a series of random steps in conformation space, each perturbing some degrees of freedom of the molecule. A step is accepted with a probability that depends on the change in value of an energy function. Typical energy functions sum many terms. The most costly ones to compute are contributed by atom pairs closer than some cutoff distance. This paper introduces a new method that speeds up MCS by exploiting the facts that proteins are long kinematic chains and that few degrees of freedom are changed at each step. A novel data structure, called the ChainTree, captures both the kinematics and the shape of a protein at successive levels of detail. It is used to efficiently detect self-collision (steric clash between atoms) and/or find all atom pairs contributing to the energy. It also makes it possible to identify partial energy sums left unchanged by a perturbation, thus allowing the energy value to be incrementally updated. Computational tests on four proteins of sizes ranging from 68 to 755 amino acids show that MCS with the ChainTree method is significantly faster (as much as 10 times faster for the largest protein) than with the widely used grid method. They also indicate that speed-up increases with larger proteins.

**Key words:** Monte Carlo simulation, protein folding, energy computation, self-collision detection, deforming chains.

# 1 Introduction

## 1.1 Monte Carlo simulation (MCS)

The study of the conformations adopted by proteins is an important topic in structural biology. MCS [6] is one common methodology for this study. In this context, it has been used for two purposes: (1) estimating thermodynamic quantities over a protein's conformation space [31, 42, 67] and, in some cases, even kinetic properties [54, 55]; and (2) searching for low-energy conformations of a protein, including its native structure [2, 3, 68]. The approach was originally proposed in [47], but many variants and improvements have later been suggested [30].

MCS is a series of randomly generated *trial steps* in the conformation space of the studied molecule. Each such step consists of perturbing some degrees of freedom (DOFs) of the molecule [43, 54, 55, 67, 68], in general torsion (dihedral) angles around bonds (see Section 1.2). Classically, a trial step is *accepted* – i.e., the simulation actually moves to the new conformation – with probability $\min\{1, e^{-\Delta E/k_bT}\}$ (the so-called Metropolis criterion [47]), where $E$ is an energy function defined over the conformation space, $\Delta E$ is the difference in energy between the new and previous conformations, $k_b$ is the Boltzmann constant, and $T$ is the temperature of the system. So, a downhill step to a lower-energy conformation is always accepted, while an uphill step is accepted with a probability that goes to zero as the energy barrier grows large. It has been shown that a long MCS with the Metropolis criterion and an appropriate step generator produces a distribution of accepted conformations that converges to the Boltzmann distribution.

Molecular Dynamics simulation (MDS) is another common approach for studying protein conformations. The forces on the atoms are then computed at each step, and used to calculate the atom positions at the next step. To be physically accurate, MDS proceeds by small time steps (usually on the order of a few femto-seconds), resulting in slow progress through conformation space. Most MDS techniques update the Cartesian coordinates of the atoms at each step, but recently there has been growing interest in directly updating torsion angles [26, 59], as this allows for both a more compact representation of the conformation space and for larger time steps. In general, MCS makes larger conformational changes than MDS and thus tends to explore subsets of the conformation space faster. But unlike MDS, pathways produced by MCS may not be physically valid, hence may not always provide useful kinetic information.

The need for general algorithms to speed-up MCS has often been mentioned in the biology literature, most recently in [67]. In this paper, we propose a new algorithm that achieves this goal, independent of the specific energy function, step generator, and acceptance criterion. More precisely, our algorithm reduces the average time needed to decide whether a trial step is accepted, or not, without affecting which steps are attempted, nor the outcome of the test. It achieves this result by incorporating efficient techniques to incrementally update the value of the energy function during simulation. Although we will describe this algorithm for its application to classic MCS, it could also be used to speed up other kinds of MCS methods, as well as other optimization and sampling techniques. Several such applications will be discussed in Section 10.2.

## 1.2 Kinematic structure of a protein

A protein is the concatenation of small molecules (the amino acids) forming a long backbone chain with small side chains. Since bond lengths and angles between any two successive bonds are almost constant across all conformations at room temperature [24], it is common practice to assume that the only DOFs of a protein are its torsion angles, also called the internal coordinates. Each amino-acid contributes two torsion DOFs to the backbone – the so-called $\phi$ and $\psi$ angles. See Figure 1 for an illustration and Figure 2 for a definition of torsion angle. Thus, the backbone is commonly modelled as a long chain of links separated by torsion joints (the backbone's DOFs). A link, which designates a rigid part of a kinematic chain, is a group of atoms with no DOFs between them. For example, in the model of Figure 1, the $C$ and $O$ atoms of amino-acid $i-1$ together with the $N$ and $H$ atoms of amino-acid $i$ form a link of the protein's backbone, since none of the bonds between them is rotatable. While a backbone may have many DOFs (between 136 and 1510 in the proteins used for the tests reported in this paper), each side-chain has between 0 and 4 torsion DOFs (known as the $\chi$ angles). In Figure 1, these DOFs are hidden inside the ball marked $R$ in each amino-acid.

The model of Figure 1 is the most common torsion-DOF representation used in the literature, and is therefore the one we use in this paper. However, it is also possible to apply our algorithm to models that include additional DOFs, such as: $\omega$ angles (rotations about the peptide bonds C–N between adjacent amino acids), bond lengths, and bond angles. At the limit, one can make each link a single atom and each joint a rigid body-transform. However, while it is theoretically possible to perform MCS in the Cartesian coordinate space, where each atom has 3 DOFs, it is more efficient to run it in the torsion-DOF space [49]. Hence, the vast majority of MCS are run in this space [43, 54, 55, 67, 68].

Due to the chain kinematics of the protein, a small change in one DOF of the backbone may cause large displacements of some atoms. Thus, in an MCS, a high percentage of steps are rejected because they lead to high-energy conformations, in particular conformations with steric clashes (self-collisions). In fact, the rejection rate tends to grow quickly with the number $k$ of DOFs changed in a single step. This is a well-known fact reported in the biology literature [1, 43]. Figure 3 illustrates this point with data gathered during an actual MCS run of 100,000 steps on an unfolded conformation of 1HTB (378 residues). The top plot shows the rejection rate as a function of $k$, while the bottom plot gives the total number of DOF changes in accepted steps during a run of 100,000 trial steps, also as a function of $k$. We note that the largest total number of DOF changes is obtained for $k = 1$. Hence, it is common practice in MCS to change few DOFs (picked at random) at each trial step [35, 37, 43, 54, 55, 67, 68].

### 1.3 Computing the energy

Various energy functions have been proposed for proteins [22, 35, 40, 41, 61]. For all of them, the dominant computation is the evaluation of non-bonded terms, namely energy terms that depend on distances between pairs of non-bonded atoms. These may be physical terms (e.g., van der Waals and electrostatic potentials [41]), heuristic terms (e.g., potentials between atoms that should end up in proximity to each other [22]) and/or statistical potentials derived from a structural database (e.g. [35]).

To avoid the quadratic cost of computing and summing up the contributions from all pairs, cutoff distances are usually introduced, exploiting the fact that physical and heuristic potentials drop off quickly toward 0 as the distance between atoms increases. We refer to the pairs of atoms that are close enough to contribute to the energy function as the *interacting pairs*. Because van der Waals forces prevent atom centers from getting very close, the number of interacting pairs in a protein is often less than quadratic [28].

Hence, one may try to reduce computation by finding the interacting pairs without enumerating all atom pairs. A classical method to do this is the grid algorithm (see Section 2), which indexes the position of each atom in a regular three-dimensional grid. This method takes time linear in the number of atoms, which is asymptotically optimal in the worst case. However, it does not exploit an important property of proteins, namely that they form long kinematic chains. It also does not take advantage of the common practice in MCS to change only a few DOFs at each time-step, which entails that at every step of the simulation large fragments of the protein remain rigid, leaving many partial energy sums are unaffected. The grid method re-computes all interacting pairs at each step and cannot directly identify partial sums that have remained constant. Thus it fails to efficiently sum up the contributions of the interacting pairs.

### 1.4 Contributions

In contrast to the grid method, the method proposed in this paper finds the new interacting pairs and retrieves unaffected partial sums without enumerating all interacting pairs. It uses a novel hierarchical data structure – the *ChainTree* – that captures both the chain kinematics and shape of a protein at successive levels of detail. At each step, the ChainTree can be maintained and queried efficiently to detect self-collision and/or find new interacting pairs. It also enables the identification of unchanged partial energy sums (stored in a companion data structure, called the *EnergyTree*), thus allowing for efficient energy updates throughout the simulation.

We tested both the ChainTree and the grid methods on four proteins (identified by 1CTF, 1LE2, 1HTB, and 1JBO in the Protein Data Bank [5]) of sizes ranging from 68 to 755 amino acids (204 to 2265 backbone atoms). These tests demonstrate that our algorithm is very effective at exploiting the fact that large protein fragments remain rigid at each step. More specifically, our results show that MCS with the ChainTree method is significantly faster than with the grid method when the number $k$ of DOF changes at each step is small. We observed speed-ups by factors up to 12 for the largest of the four proteins. Therefore, not only does a small $k$ sharply increases the step acceptance ratio, it also makes it possible to expedite the evaluation of the acceptance criterion. Simulation methodologies other than classical MCS could also benefit from our algorithm (see Section 10.2).

### 1.5 Outline of paper

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the ChainTree data structure. Section 4 describes our algorithm to detect self-collision (steric clash). The complexity of this algorithm is analyzed in Section 5 and its performance is empirically assessed in Section 6. Section 7 extends the algorithm to find new interacting atom pairs and efficiently update energy at each simulation step. Section 8 gives experimental results comparing our algorithm with the grid algorithm in MCS. In Section 9 we describe a software implementation of the ChainTree algorithm that is available for download. Section 10 discusses applications of our algorithm to more complex MCS methods as well as to other types of molecular simulation methods and points to possible extensions and future directions of research. The application of the ChainTree to test a long kinematic chain for self-collision was previously presented in [45]. A shorter version of this paper appeared in [46].

Throughout this paper, we always use $n$ to denote the number of links of a kinematic chain (e.g., a protein's backbone) and $k$ to denote the number of DOF changes per simulation step. Since the number of atoms in any amino acid is bounded by a constant, the number of atoms in a protein is always $O(n)$. Although $k$ can be as big as $O(n)$, it is much smaller in practice [1, 43, 54, 55, 67, 68].

# 2 Related Work

## 2.1 Collision detection

The problem considered in this paper is closely related to detecting collision among moving objects and checking a deforming object for self-collision. Collision detection has been extensively studied in robotics [10, 18, 19, 20, 44, 52, 53], computer graphics [9, 12, 21, 23, 32, 33, 36, 38, 64] and computational geometry [4, 15, 25, 45], to only cite a few works. Most research, however, has been conducted in environments made of rigid objects, few of them moving.

The collision-detection methods in [4, 44] rely on tracking object features (e.g., closest features) to compute minimal separation distance. They require the identity of the tracked features to change rarely, the so-called *temporal/spatial coherence* assumption. In particular, this assumption implies that during any small time step the placements of the objects undergo small changes. A long kinematic chain does not satisfy this assumption, since a small DOF change may cause relatively large displacements of parts of the chain.

Other collision-detection methods partition the space in which an object moves, for example, into an octree [18, 21], a regular 3-D grid [28], or a set of projections onto subspaces [12]. Usually, these approaches do not lend themselves to incremental updating to handle a deformable object or many objects moving simultaneously. Exceptions include [12, 21], but then the temporal/spatial coherence assumption must be satisfied.

The most popular approach to collision detection pre-computes a bounding-volume hierarchy (BVH) for each object. This hierarchy captures spatial proximity between small components of the object at successive resolutions. The hierarchies are then used to expedite collision tests by quickly discarding pairs of components contained in non-overlapping bounding volumes (BVs) [9, 19, 23, 32, 36, 38, 52, 64]). Various types of BVs have been used, e.g., spheres and bounding boxes. These techniques have been extended in [9, 64] to handle deformable objects by exploiting the facts that neighboring elements in the meshed surface of an object always remain in proximity to each other when the object deforms. For each deformable object, a balanced BV tree is pre-computed by successively grouping topologically close features of the meshed surface and enclosing them in a BV. When the object deforms, the topology of the tree stays unchanged; only the sizes and locations of the BVs are updated in a bottom-up fashion. However, BV techniques lose efficiency when applied to detecting self-collision in a deformable object, because they cannot avoid detecting the trivial self-collision of each object component with itself. They also lose efficiency when many objects (rigid or not) move independently.

The ChainTree borrows from previous work on BVHs. It uses a BVH based on the invariance of the chain topology. But it combines it with a transform hierarchy that makes it possible to efficiently prune the search for a self-collision, when few DOFs change simultaneously.

Only a limited amount of previous research has been dedicated to kinematic chains. A general scheme is proposed in [27] to efficiently update a representation of a kinematic chain designed to efficiently detect collision with fixed obstacles. But this scheme does not support self-collision detection. The work in [25] addresses a problem similar to ours – testing self-collision in a deformable necklace made of spherical beads. Like our method, it builds a BVH based on the chain topology. However, it assumes that all DOFs change simultaneously at each step and does not attempt to take advantage of rigid sub-chains, nor does it consider the problem of maintaining the value of an energy function. When all DOFs change simultaneously our algorithm does about the same amount of work as the algorithm in [25] to detect self-collision.

The problem of deciding whether a torsion angle change causes a self-collision in a 3D polygonal chain has been studied in [57, 58]. It was shown in [58] that determining whether a rotation around a bond causes a self-collision anywhere along its path takes $\Omega(n \log n)$ time, when $n$ is the number of links in a chain. It is further conjectured in [57] that no amount of preprocessing enables performing $n$ such rotations in worst case sub-linear time per rotation. However, in this paper, we only care whether the conformation at the end of the rotation is collision-free and not whether there is a collision during the motion.

## 2.2 Finding interacting pairs

Because biologists are more interested in simulation results than in the computational methods they use to achieve these results, the literature does not extensively describe algorithms for MCS.

A prevailing algorithm – referred to as the *grid algorithm* in this paper – reduces the complexity of finding all interacting pairs in a molecule to asymptotically linear time by indexing the atoms in a regular grid. This approach exploits the fact that van der Waals potentials prevent atom centers from coming very close to one another. In [28] it is formally shown that in a collection $B$ of $n$ possibly overlapping balls of similar radii, such that no two sphere centers are closer than a small fixed distance, the number of balls that intersect any given ball of $B$ is bounded by a constant. This result yields the grid algorithm, which subdivides the 3D space into cubes whose sides are set to the maximum diameter of the balls in $B$, computes the cubes intersected by each ball, and stores the results in a hash-table. This data structure is re-computed after each step in $\Theta(n)$ time. Determining which balls intersect any given ball of $B$ then takes $O(1)$ time. Hence, finding all pairs of intersecting balls takes $\Theta(n)$ time. The grid method can be used to find all pairs of atoms within some cutoff distance, by growing each atom by half this distance. The method is asymptotically optimal in the worst case, but updating the data structure always takes linear time. This is too costly for very large proteins, making it impractical to perform MCS in this case.

A variant of this method mostly used for Molecular Dynamics simulation maintains, for each atom, a list of atoms within a distance $d$ somewhat larger than the cutoff distance $d_c$ by updating it every $s$ steps [62]. The idea is that atoms further apart than $d$ will not come closer than $d_c$ in less than $s$ steps. There is a tradeoff between $s$ and $d - d_c$ since the larger this difference, the larger the value of $s$ that can be used. However, choosing $d$ big causes the neighbor lists to become too large to be efficient. A method for updating neighbor lists based on monitoring the displacement of each atom is described in [48].

Linear complexity seems to be the best one can achieve in practice when all DOFs change at each step, as is the case in Molecular Dynamics simulation. We are not aware of any principled attempt to exploit the chain kinematics of proteins and the small number of changing DOFs at each step to speed up MCS.

Molecule-dependent and simulation-dependent heuristics have also been used to simplify energy computation. However, such heuristics, which often rely on the prior insight of the biologist, require adjustments and tuning to deal with each new molecule or family of molecules, or a different step generator.

# 3 The ChainTree

In this section we describe the ChainTree, the data structure we use to represent a protein. We begin by stating the key properties of proteins and MCS that motivated this data structure (Subsection 3.1). Then follows a description of the two hierarchies that make up the ChainTree. The *transform* hierarchy that approximates the kinematics of the backbone is introduced in Subsections 3.2 and the *bounding-volume* hierarchy that approximates the geometry of the protein is presented in

Subsection 3.3. Next, we discuss the representation of the side-chains (Subsection 3.4). Finally, we describe how the two aforementioned hierarchies are combined to form a single balanced binary tree (Subsection 3.5) and the way it is updated (Subsection 3.6).

In the following we refer to the algorithm that updates the ChainTree as the *updating* algorithm and to the algorithm that tests self-collision or finds interacting pairs as the *testing* algorithm.

## 3.1 Properties of proteins and MCS

A protein backbone is commonly modelled as a kinematic chain made up of a sequence of $n$ links (atoms or rigid groups of atoms) connected by torsional DOFs. The ChainTree is motivated by three key properties deriving from this model:

**Local changes have global effects:** Changing a single DOF causes all links beyond this DOF, all the way to the end of the chain, to move. Any testing algorithm that requires knowing the absolute position of every link at each step must perform $O(n)$ work at each step even when the number $k$ of DOF changes is $O(1)$.

**Small angular changes may cause large motions:** The displacement of a link caused by a DOF change depends not only on the angular variation, but also on the distance between the DOF axis and the link (radius of rotation). So, at each step, any link with a large radius of rotation undergoes a large displacement, implying that the temporal/spatial coherence assumption is not valid.

**Large sub-chains remain rigid at each step:** If we only perturb few DOFs at each step, as is the case during MCS, then large contiguous fragments of the chain remain rigid between steps. So, there cannot be any new self-collisions or new interacting pairs inside each of these fragments.

## 3.2 Transform hierarchy

We attach a reference frame to each link of the protein's backbone and map each DOF to the rigid-body transform between the frames of the two links it connects. The transform hierarchy is a balanced binary tree of transforms. See Figure 4, where ovals and labelled arrows depict links and transforms, respectively.

At the lowest level of the tree, each transform represents a DOF of the chain. Products of pairs of consecutive transforms give the transform at the next level. For instance, in Figure 4, $T_{AC}$ is the product of $T_{AB}$ and $T_{BC}$. Similarly, each transform at every level is the product of two consecutive transforms at the level just below. The root of the tree is the transform between the frames of the first and last links in the chain ($T_{AI}$ in the figure).

Each of the $\log n$ levels of the tree can be seen as a chain that has half the links and DOFs of the chain at the level just below it. In total, $O(n)$ transforms are cached in the hierarchy. We say that each intermediate transform $T_{\alpha\beta}$ *shortcuts* all the transforms that are in the subtree rooted at $T_{\alpha\beta}$.

The transform hierarchy is used from the top down by the testing algorithm to propagate transforms defining the relative positions of bounding boxes (from the other hierarchy) that need to be tested for overlap. It also allows computing the relative position of any two links or boxes in $O(\log n)$ time, but this property is not used by our algorithm.

A structure similar to our transform hierarchy was introduced in [57]. However, it was only used for theoretical analysis and was not implemented.

### 3.3 Bounding-volume hierarchy

The bounding-volume (BV) hierarchy is similar to those used by prior collision checkers (see Section 2.1). As spatial proximity in a deformable chain is not invariant, our BVH is based on the proximity of links along the chain. See Figure 5. Here, for simplicity, we introduce the BVH used for self-collision detection. In Section 7.2 we will describe the extensions allowing us to both detect self-collision and find new interacting pairs using a single hierarchy.

Like the transform hierarchy, the BVH is a balanced binary tree. It is constructed bottom up in a "chain-aligned" fashion. At the lowest level, one BV bounds each link. Then, pairs of neighboring BVs at each level are bounded by new BVs to form the next level. The root BV encloses the entire chain. So, at each level, we have a chain with half the number of BVs in the chain at the level below it. This chain of BVs encloses the geometry of the chains of BVs at all lower levels.

The type of BV we use is the oriented bounding box (OBB) [23], a rectangular bounding box at an arbitrary orientation. We chose OBBs because they bound well both globular objects (single atoms, small groups of atoms) and elongated objects (chain fragments). In addition, unlike simpler axis-aligned bounding boxes [9, 64], but like spheres, OBBs are invariant to a rigid-body transform of the geometry they bound. In the ChainTree, this property allows us not to re-compute the BV of a rigid sub-chain, even when this sub-chain has moved. Finally, OBBs can be efficiently computed and tested for intersection. Spheres are another frequently used BVs [25, 52] that would meet our needs. However, in a chain-aligned hierarchy, we expect them to bound poorly elongated sub-chains, e.g., $\alpha$-helices and $\beta$-strands.

We construct each intermediate OBB to enclose its two children, thus creating what we term a *not-so-tight* hierarchy (in contrast to a *tight* hierarchy where each BV tightly bounds the links of the sub-chain it encloses). In the Appendix (Lemma 2), we show that the size of these OBBs does not deteriorate too much as one climbs up the hierarchy. The major advantage of using this not-so-tight BVH is the efficiency with which each box can be updated. Indeed, the shape of the OBB stored at each intermediate node depends only on the 16 vertices of the two OBBs held by this node's children.

### 3.4 Side-chain representation

Side-chains, one per amino acid, are short chains with up to 20 atoms, that protrude from the backbone [13]. A side-chain may have some internal torsional DOFs (between 0 and 4). The biology literature proposes different ways to model side-chains ranging from a single sphere approximating the entire side-chain, to a full atomistic model [13, 41]. The choice depends on both the physical accuracy one wishes to achieve and the amount of computation one is willing to pay per simulation step. One may choose to make the side-chains completely rigid, or allow their DOFs to change during the simulation. In both cases we expect the overhead of using a sub-hierarchy for the side-chain atoms to exceed any benefit it may provide. Therefore, we allow each link of the protein backbone to be an aggregate of atoms represented in a single coordinate frame and contained in an OBB that is a leaf of the BVH. Each such aggregate includes one or several backbone atoms forming a rigid piece of the backbone and the atoms of the side-chain stemming from it (contained in the circle marked $R$ in Figure 1. When a side-chain DOF is changed, the positions of the affected atoms is recomputed in the coordinate frame of the link they belong to, and the BV of that link is recomputed.

### 3.5 Combined data structure

The ChainTree combines both the transform and the BV hierarchies into a single binary tree as the one depicted in Figure 6. The leaves of the tree (labelled $A$ through $H$ in the figure) correspond to the links of the protein's backbone with their attached side-chains as described in Section 3.4). Each leaf holds both the bounding box of the corresponding link and side-chain and the transform (symbolized by a horizontal arrow in the figure) to the reference frame of the next link.

Each internal node (nodes $J$ through $P$) has the frame of the leftmost link in its sub-tree associated with it. It holds both the bounding box of the boxes of its two children and the transform to the frame of the next node at the same level if any. So, computing the relative position of a box and its left child box requires no coordinate transform, while computing the position of a box relative to its right child box requires one transform, which is stored at the left child node. This remark is used by the testing algorithm to propagate down the relative positions of pairs of boxes that are to be tested for overlap.

The ChainTree contains both pointers from children to parents, which are used by the updating algorithm to propagate updates from the bottom up, as described below, and pointers from parents to children, which are used by the testing algorithm (Section 4).

### 3.6 Updating the ChainTree

When a change is applied to a single arbitrary DOF in the backbone, the updating algorithm recomputes all transforms that shortcut this DOF and all boxes that enclose the two links connected by this DOF. It does this in bottom-up fashion, by tracing the path from the leaf node immediately to the left of the changed DOF up to the root. A single node is affected at each level. If this node holds a transform, this transform is updated. If it holds a box that contains the changed DOF, then the box is re-computed. For example, in Figure 6, if the DOF between the links associated with nodes $F$ and $G$ is changed, then the transforms stored at $F$ and $L$ and the boxes at $O$ and $P$ are re-computed. Since the shape of an OBB is invariant to a rigid-body transform of the objects they bound, all other boxes remain unchanged.

If a DOF is changed in a side-chain, the box stored at the corresponding leaf node of the ChainTree and the boxes of all the ancestors of this node are re-computed, but *all transforms in the hierarchy remain unchanged.*

When multiple DOFs are changed simultaneously (in the backbone and the side-chains), the ChainTree is updated one level at a time, starting with the lowest level. Hence, all affected transforms and boxes at each level are updated at most once before proceeding to the level above it.

The updating algorithm marks every node whose box and/or transform is re-computed. This mark will be used later by the testing algorithm. Figure 7 shows the marked nodes in the ChainTree of Figure 6, after a change in the DOF between the links associated with nodes $F$ and $G$. The nodes with bold contours ($F$ and $L$) are those whose transforms were updated. The nodes in grey ($O$ and $P$) are those whose boxes were re-computed. In general, however, marked nodes may have had both their transforms and boxes updated.

# 4 Self-Collision Detection

## 4.1 Using a BVH

BVHs have been widely used to detect collision between pairs of rigid objects, each described by its own hierarchy [23, 32, 36, 52, 64]. Given the hierarchies of two objects, the algorithm first checks whether the root boxes overlap. If they do not, it can safely return that the two objects do not collide. If they do overlap, then the algorithm descends one level in both hierarchies and tests all four pairs of children. It continues this process iteratively. When the lowest level of one hierarchy is reached, the algorithm continues its descent through the other hierarchy, testing the leaf boxes of one hierarchy against boxes at the newly reached level in the other hierarchy. When it reaches leaves in both hierarchies, it tests the actual components of the objects for overlap and returns a collision when they overlap. The algorithm may stop as soon it has found a collision, or it may run until it has found all collisions. In the former case, it is best for the algorithm to search for a collision in a depth-first manner to reach overlapping leaves as quickly as possible.

The algorithm terminates quickly when the two objects are well separated, because the search then ends near the top of both hierarchies and avoids dealing with the possibly complicated geometry of the actual objects. If one is only interested in detecting whether a collision occurs, the algorithm also terminates quickly when the overlap is large, because the depth-first search is then expeditious in finding a pair of overlapping leaf nodes. The algorithm takes longer when the objects are close but do not collide, or when there are many distinct collisions and one wants to find them all.

A simple variant of this algorithm detects self-collision by testing the BVH of the object against itself. This variant skips the test of a box against itself and proceeds directly to testing the box's children. However, it takes $\Omega(n)$ time, since all leaves will inevitably be visited (each leaf is trivially in collision with itself). The ChainTree allows us to avoid this lower bound by exploiting the third property stated in Section 3.1 — large sub-chains remain rigid between steps.

## 4.2 Using the ChainTree

When only a small number $k$ of DOFs are changed simultaneously, long sub-chains remain rigid at each step. These sub-chains cannot contain new self-collisions. So, when we test the BVH contained in the ChainTree against itself, we prune the branches of the search that would look for self-collision within rigid sub-chains.

There are two distinct situations where pruning occurs:

1. If the algorithm is about to test a box against itself and this box was not updated after the last DOF changes, then the test is pruned.
2. If the algorithm is about to test two different boxes, and neither box was updated after the last DOF changes, and no backbone DOF between those two boxes was changed, then the test is pruned.

The last condition in this second situation – that no backbone DOF between the two boxes was changed – is slightly more delicate to recognize efficiently. We say that two nodes at the same level in the ChainTree are *separated* if there exists another node between them at the same level that holds a transform that was modified after the last DOF changes. This node will be dubbed *separator*. They are also separated if the transform held by the leftmost of the two was modified. Hence, if two nodes are separated, a DOF between them has changed. We remark that:

- If two nodes at any level are separated, then any pair consisting of a child of one and a child of the other is also separated.
- If two nodes at any level are *not* separated, then a child of one and a child of the other are separated if and only if they are separated by another child of either parent.

Hence, by pushing separation information downward, the testing algorithm can know in constant time whether a DOF has changed between any two boxes it is about to test. The algorithm also propagates transforms from the transform tree downward to compute the relative position of any two separated boxes in constant time before performing the overlap test.

To illustrate how the testing algorithm works, consider the ChainTree of Figure 7 obtained after a change of the DOF between $F$ and $G$. $F$ and $L$ are the only separators. The algorithm first tests the box stored in the root $P$ against itself. Since this box has changed, the algorithm examines all pairs of its children, $(N, N)$, $(N, O)$ and $(O, O)$. The box held in $N$ was not changed, so $(N, N)$ is discarded (i.e., the search along this path is pruned). $(N, O)$ is not discarded since the box of $O$ has changed, leading the algorithm to consider the four pairs of children $(J, L)$, $(J, M)$, $(K, L)$, and $(K, M)$. Both $(J, L)$ and $(K, L)$ satisfy the conditions in the second situation described above; thus, they are discarded. $(J, M)$ is not discarded because $J$ and $M$ are separated by $L$. The same is true for $(K, M)$, and so on.

## 5 Complexity Analysis

Two fundamental tradeoffs must be made when using a BVH for self-collision detection:

1. Between the number of box overlap tests needed to detect or rule out self-collision and the cost of one such test: In general, the more complex the BVs, the tighter they bound a given geometry and the fewer tests are required; but the more expensive each test becomes.

2. Between the time needed to update the BVH and the time to detect or rule out self-collision: In general, reducing updating time (e.g., by using not-so-tight boxes or keeping the topology of the hierarchy fixed) impairs the performance of self-collision detection.

The analysis below is aimed at clarifying the choices made in the ChainTree and comparing our updating and testing algorithms to current state-of-the-art algorithms. It is based on the worst-case asymptotic behavior of the algorithms when $n$ grows arbitrarily large. But this behavior is not always meaningful in practice, since for many proteins $n$ is too small and/or the worst case is unlikely. For this reason, Section 6 will complement this analysis with experimental tests.

Throughout this section we ignore the side-chains. Since the size of each one is small and bounded by a constant, the side chains only affect the asymptotic complexity bounds by a constant factor.

### 5.1 Updating the ChainTree

Updating the ChainTree after a 1-DOF change requires re-computing transforms and boxes held in nodes along the path from the leaf node immediately to the left of the changed DOF up to the root of the tree. By doing the updates from the bottom up, each affected transform is re-computed in $O(1)$ time. By using not-so-tight OBBs — thus, trading tightness for update-speed — re-computing each box is also done in $O(1)$ time. Since the ChainTree has $\lceil \log n \rceil$ levels, and at each level at most one transform and one box are updated, the total cost of the update process is $O(\log n)$.

When a $k$-DOF change is made, affected transforms and boxes are updated one level at a time. This ensures that no transform or box is re-computed more than once when the converging update paths merge. The total updating time is then $O(k \log{(n/k)})$. When $k$ grows this bound never exceeds $O(n)$.

The efficient updating time for small values of $k$ derives from the fact that the topology of the ChainTree is never modified. However, this topology is not always optimal to detect self-collision, as it only imperfectly represents spatial proximity among links.

## 5.2 Detecting self-collision

All known algorithms to detect self-collision in an $n$-link chain take $\Theta(n^2)$ time in the worst case, if no further assumption is made about the chain. Hence, they do not behave better than a brute-force algorithm.

In this subsection, we will assume that the protein chains are *well-behaved.* Let us associate with each link of a chain its minimal enclosing sphere. Given two positive constants $\gamma$ and $\delta$, a chain is well-behaved if it verifies the following two properties:

1. The ratio of the radii of the largest and smallest enclosing spheres is smaller than $\gamma$.

2. The distance between the centers of any two enclosing spheres is greater than $\delta$.

It is not hard to convince oneself that proteins form well-behaved chains for some $\gamma$ and $\delta$. The first property follows from the fact that there are only 20 different types of amino acids. The second property is verified if we exclude conformations where atoms overlap almost completely. These conformations are physically impossible because of van der Waals forces . It is shown in [28] that in well-behaved chains of arbitrary length $n$ the number of links that overlap a given link is bounded by a constant. Thus, there are at most $O(n)$ overlaps between the links of a well-behaved chain.

MCS cannot cause a well-behaved chain to degenerate. Each simulation step by itself affects only a small constant $k$ DOFs and thus the number of overlaps per atom can only increase by a constant factor. Moreover, a step that increases the number of overlaps will always be rejected because it causes a steric clash. Thus, no cumulative effect during successive steps will degenerate the chain.

Our algorithm performs two tasks for every pair of nodes it examines. First, it must decide whether to prune this search path. This requires testing if the nodes have been updated after the last DOF change and, if not, whether they are separated. Second, if this search path is not pruned, the two boxes are tested for overlap. Both tasks require constant time. Thus, the complexity of the algorithm is governed by the number of pairs of nodes that are examined, which is proportional to the number of overlapping boxes at all levels of the ChainTree in the worst case.

In the Appendix we prove the following theorem:

**Theorem 1** *The maximum total number of overlapping boxes at all levels of the ChainTree of a well-behaved chain of $n$ links is* $\Theta(n^{\frac{4}{3}})$.

Therefore, the testing algorithm runs in worst-case $\Theta(n^{\frac{4}{3}})$ time. This bound, which is similar to the one established in [25], holds whether the algorithm stops after detecting the first collision or keeps running to detect all self-collisions. Indeed, the fact that two boxes overlap does not imply that they contain colliding links. So, to detect a single self-collision or the absence of it, the algorithm may have to eventually consider all pairs of overlapping boxes.

Note also that the bound is not affected by the pruning of search paths. So, when all DOFs change at each simulation step, and as long as the chain remains well-behaved, self-collision detection still takes $\Theta(n^{\frac{4}{3}})$ time in the worst case, after $O(n)$ updating.

The experimental results of Section 6 show however that when few DOFs change at each step, the algorithm behaves much better in practice than the above bound suggests.

### 5.3 Comparison with other methods

There exist several methods applicable to the problem of detecting self-collision in a chain. The asymptotic worst-case complexity of the most important ones is given in Table 1.

At each step, the brute-force algorithm first re-computes the position of every link that has moved, which is done in worst-case linear time. It then detects self-collision by testing all pairs of links for overlap, resulting in $\Theta(n^2)$ tests in the worst case (e.g., when there are no collisions).

Under the assumption that the chain is well-behaved, the grid algorithm (see Subsection 2.2) reduces the worst-case complexity of both updating and testing to linear time [28].

Although the grid algorithm is optimal in the worst case, BV-hierarchy methods are intended to do better on average. For a chain, two types of BVHs may be used:

**Spatially-adapted hierarchy:** A BVH as described in [23, 36] which is based on a spatial partitioning of each chain conformation to optimally encode spatial proximity between links.

**Chain-Aligned hierarchy:** A BVH as the one in the ChainTree (see [25] for another example) that encodes the *chain-wise* proximity of links along the chain. Links are said to be in chain-wise proximity if they are not separated by more than a few links along the chain.

Ideally, a BVH should be such that the depth of any box in the tree is a good indicator of the spatial proximity of the links bounded by this box. This is precisely what a spatially-adapted hierarchy is intended to achieve. Instead, a chain-aligned hierarchy only encodes chain-wise proximity. If two links are chain-wise close, they are also spatially close. But the reverse is not true: in a given chain conformation, pairs of links that are chain-wise far apart may be spatially very close (e.g., this happens in folded protein conformations). Consequently, testing self-collision with a spatially-adapted hierarchy is more efficient than with a chain-aligned one. But, because spatial proximity varies as the chain deforms, while chain-wise proximity does not, updating a spatially-adapted hierarchy is more expensive.

A spatially-adapted hierarchy of a chain can be tested for self-collision in $\Theta(n)$ time in the worst case. This bound is a special case of a theorem proven in [70]. However, it is expected to do better in practice. Building a new hierarchy at each step takes $O(n \log n)$ time [23]. One could attempt to reduce updating time to $\Theta(n)$ by not changing the topology of the hierarchy and only updating the size and location of the BVs. But a single DOF change may then corrupt the hierarchy, so as to raise the number of BV overlap tests required to detect self-collision to $\Theta(n^2)$. Figure 8 illustrates such a scenario. In 8a the hierarchy is spatially-adapted with no overlap between boxes. A change in the middle DOF creates the conformation in 8b. The boxes were updated to bound the new conformation without changing the topology of the hierarchy. All pairs of boxes at the displayed level now overlap.

In contrast, despite its fixed topology, a chain-aligned BVH guarantees an $O(n^{\frac{4}{3}})$ self-collision test, with an $O(n)$ update when not-so-tight OBBs are used and an $O(n \log n)$ update when tight spheres are used [25]. Thanks to its transform hierarchy, the ChainTree reduces further the updating cost after a $k$-DOF change to $O(k \log (n/k))$.

The worst-case bound on self-collision detection with the ChainTree hides the practical speed-up allowed by search pruning. To evaluate this speed-up, we need to perform empirical tests.

## 6 Test Results for Self-Collision Detection

We conducted various tests with our implementation of our method – called here **ChainTree** – and the following three methods:

**Grid** - This is the grid method presented in Subsection 2.2. The length of a side of each grid cell is the diameter of the largest atom. Both updating and testing take $\Theta(n)$ time.

**1-OBBTree** - Here, a tight spatially-adapted OBB hierarchy is re-computed at each step, and then tested against itself for self-collision. Updating and testing take $O(n \log n)$ and $O(n)$ worst-case time, respectively.

**K-OBBTree** - At each step, this algorithm computes a tight spatially-adapted OBB hierarchy for each rigid sub-chain. It then tests each pair of hierarchies for collision. Updating and testing take $O(n \log n)$ and $O(n)$ worst-case time, respectively.

Both 1-OBBTree and K-OBBTree are based on the PQP library [23, 38] from UNC. The function of ChainTree testing pairs of OBBs for overlap is also from this library. Our experiments were run on a single 400 MHz UltraSPARC-II CPU of a Sun Ultra Enterprise 5500 machine with 4.0 GB of RAM.

In a first series of tests, we created pseudo-protein chains of $n$ spheres of radius 1 unit, spaced 4 units apart by a torsion joint, with $n = 1000$, 2500, 5000 and 10000 spheres in initial compact cube-like conformations. Each simulation consisted of 100,000 steps, each changing a single DOF picked uniformly at random. The magnitude of the change was chosen uniformly at random between $0°$ and $30°$. If a collision was detected, the step was rejected, i.e., the change was undone before proceeding to the next step. This requires keeping two copies of the ChainTree. For each chain, the four runs (one with each method) started at the same initial conformation and used the same seed of the random number generator. Hence, they generated the same sequence of conformations.

The average times per step are shown in Figure 9a, when the algorithms stop after finding the first collision. The plots show both updating times (black) and testing times (grey). Some times are too small to be discernable. ChainTree is 5 times faster than Grid for 1000 spheres and about 60 times faster for 10000 spheres. When finding all self-collisions, ChainTree is 4 times faster than Grid for 1000 spheres and 53 times faster for 10000 spheres. In all cases, 1-OBBTree and

K-OBBTree are the slowest methods because of their high updating time, although K-OBBTree has the fastest testing time of all four methods.

In a second series of tests we used four proteins from the Protein Data Bank[1] (PDB) [5]. We selected these proteins – named 1CTF, 1LE2, 1HTB, and 1JB0 in the PDB – to span different sizes. See Table 2. For each protein, we ran the same simulation as in the first series of tests, starting at the folded (native) state of the protein. However, only the protein's backbone was considered and the side-chains were ignored. At each step the four algorithms stopped after finding the first collision. The results are summarized in Figure 9b. ChainTree is twice as fast as Grid for the smallest protein (1CTF) and 5 times faster for the largest one (1JB0). Note that the chains in this benchmark are significantly shorter than those in the first benchmark.

We also examined the effect of the number $k$ of simultaneous DOF changes at each step on the performance of the algorithms. We let $k$ vary from 1 to 121 and, for each value, measured the average time per step to update and find all collisions. In order to eliminate the effect of compactness on the results, we chose 11 conformations of 1HTB each having a different radius of gyration distributed uniformly between 20Å (the radius of the native conformation) and 85Å. For each such conformation and each value of $k$ we ran a simulation of 10,000 steps starting at that conformation. We constrained each simulation to stay at its starting conformation so its compactness would not change. Since more simultaneous changes cause more collisions, on average it takes less time to find one collision when $k$ is large. Therefore we measured the time to find *all* collisions which is less affected by $k$. Figure 10 shows the results. The time results are averaged over all 11 conformations. In this case, ChainTree is the fastest algorithm until $k = 50$, thanks to both its logarithmic updating time and its fast self-collision detection due to search pruning. When

[1] `http://www.rcsb.org/pdb`

$k$ increases, the updating time of ChainTree deteriorates and search pruning becomes less effective as rigid sub-chains are shorter. For $k > 50$, Grid, whose performance is independent of $k$, becomes faster.

Figure 11 reveals more explicitly the effectiveness of search pruning in ChainTree. It shows the average number of box overlap tests performed by ChainTree with and without search pruning in the previous experiment (averaged over 11 conformations of 1HTB, as explained above), for increasing values of $k$. The numbers are significantly smaller with pruning than without when $k$ is small. As expected, they converge as $k$ increases.

# 7 Incremental Energy Updates in MCS

We now extend our testing algorithm to incrementally update the energy of a protein undergoing MCS. A typical energy function is of the form $E = E_1 + E_2$, where $E_1$ sums terms depending on a single parameter commonly referred to as *bonded* terms (e.g., torsion-angle and bond-stretching potentials) and $E_2$ sums terms commonly known as *non-bonded* terms, which account for interactions between pairs of atoms or atom groups closer than a cutoff distance [35, 37, 40, 43, 54, 55]. Updating $E_1$ after a conformational change is straightforward. This is done by computing the sum of the differences in energy in the terms affected by the change and adding it to the previous value of $E_1$. After a $k$-DOF change, there are only $O(k)$ affected single-parameter terms. So, in what follows we focus on the update of $E_2$.

## 7.1 Overview

At each simulation step we must find the interacting pairs of atoms and change $E_2$ accordingly. Finding these pairs is similar to finding all self-collisions. One may use the same algorithm, after having grown every atom by half the cutoff distance.

However, when $k$ is small, many interacting pairs are unaffected by a $k$-DOF change. The number of affected interacting pairs, though still $O(n)$ in the worst case, is usually much smaller than the total number of interacting pairs at the new conformation. Therefore, an algorithm like the grid algorithm that computes all interacting pairs at each step is not optimal in practice. Moreover, after having computed the new set of interacting pairs, we still have to update $E_2$, either by re-computing all pairwise contributions from scratch, or by scanning the old and new sets of interacting pairs to determine which terms should be subtracted from the old value of $E_2$ and which terms should be added to get the new value. In either case, we perform again a computation at least proportional to the total number of interacting pairs.

Instead, our method directly finds all new interacting pairs, including the previous pairs whose distances have changed. It also detects partial energy sums unaffected by the DOF change (these sums correspond to interacting pairs where both atoms belong to the same rigid sub-chains). The energy terms contributed by the new pairs are then added to the unaffected partial sums to obtain the new value of $E_2$. In practice, the total cost of this computation is roughly proportional to the number of changing interacting pairs. The algorithm makes use of a new data structure, which we call the *EnergyTree*, in which partial sums computed at previous steps have been cached.

## 7.2 Finding the new interacting pairs

We wish to find the new interacting pairs of atoms at each step. These include all interacting pairs that were not interacting at the previous conformation, as well as pairs that were previously interacting but whose distances have changed. They do not include, however, previous interacting pairs that are no longer interacting.

To do this, we can use the ChainTree described in Section 3, after having grown all atom radii by half the cutoff distance. The testing algorithm of Section 4 finds exactly the new interacting pairs. However, changing the cutoff distance for the same molecule requires re-computing all OBBs in the ChainTree. Changing this distance can be useful, e.g., to detect all very close pairs of atoms in a first pass and thus quickly discard conformations that are very unfavorable energetically. See Subsection 8.3.

This drawback led us to replace OBBs by RSSs (for rectangle swept sphere) and overlap tests of BVs by distance computation. An RSS is a type of BV introduced in [38] that is defined as the Minkowski sum of a sphere and a rectangle. The RSS bounding a set of points in 3D is created very much like an OBB. The two principal directions spanned by the points are computed and a rectangle is constructed along these directions to enclose the projection of all points onto the plane defined by these directions. The RSS is the Minkowski sum of this rectangle and the sphere whose radius is half the length of the interval spanned by the point set along the dimension perpendicular to the rectangle. In comparison, the OBB of this set of points is the cross-product of the rectangle by this interval.

We create the RSS hierarchy in the way described in Subsection 3.3. But we slightly modify the algorithm of Section 4 to test whether pairs of RSSs are closer apart than the cutoff distance, instead of testing pairs of OBBs for overlap. The search pruning rules for rigid sub-chains are unchanged, so that we do not re-compute interacting pairs inside rigid sub-chains. These have already been identified at previous steps of the MCS. The partial energy sums corresponding to these pairs are unchanged and can be retrieved from the EnergyTree as described in the next subsection. To compute the distance between two RSSs, one simply computes the distance between the two underlying rectangles minus the radii of the swept spheres. This is faster than computing the distance between two OBBs, and the BVH is independent of the cutoff distance.

The asymptotic bounds for updating and testing stated in Section 5 hold unchanged. For the bound on testing, this derives from the observation that an RSS may be larger than an OBB only by a constant factor. More precisely, the RSS of a set of points is larger than the OBB along its two principal directions by no more than the length of the smallest side of the OBB (the diameter of the swept sphere). Thus, the RSS could easily fit inside an OBB twice as large as the optimal OBB along each dimension. Moreover, computing the distance between two RSSs takes constant time, just like testing two OBBs for overlap.

However, the fact that the asymptotic bound for testing — $\Theta(n^{\frac{4}{3}})$ — is unchanged can be misleading. This bound relies on the fact that protein backbones are well-behaved chains, and as such, the number of atoms that may *interact* with any given atom is bounded by a constant as the protein grows arbitrary large. This constant is likely to be much larger than the constant bounding the number of atoms that may *overlap* any given atom. The situation might even be worse for small proteins, in which almost all pairs of atoms could be interacting. Hence, we may expect the testing algorithm to take significantly more time when it is used to find all new interacting pairs than when it is used to find all self-collisions. We will exploit this remark in Subsection 8.3 to propose a two-pass testing algorithm.

## 7.3 Updating the energy value

Recall that when the testing algorithm examines a pair of sub-chains (including the case of two copies of the same sub-chain), it first tests whether these sub-chains have not been affected by the DOF change and are contained in the same rigid sub-chain. If this is the case, the two sub-chains cannot contribute new interacting pairs, and the algorithm prunes this search path. But, for this same reason, the sum of energy terms contributed by the interacting pairs from these sub-chains is also unchanged. We would like to be able to retrieve it. To this end, we introduce another data structure, the *EnergyTree*, in which we cache the partial sums corresponding to all pairs of sub-chains that the testing algorithm may possibly examine. Figure 12 shows the EnergyTree for the ChainTree of Figure 6.

Let $\alpha$ and $\beta$ be any two nodes (not necessarily distinct) from the same level of the ChainTree. If they are not leaf nodes, let $\alpha_l$ and $\alpha_r$ (resp., $\beta_l$ and $\beta_r$) be the left and right children of $\alpha$ ($\beta$). Let $E(\alpha, \beta)$ denote the total energy contributed by all interacting pairs in which one atom belongs to the sub-chain corresponding to $\alpha$ and the other atom belongs to the sub-chain corresponding to $\beta$. If $\alpha \neq \beta$, we have:

$$E(\alpha, \beta) \quad = \quad E(\alpha_l, \beta_l) + E(\alpha_r, \beta_r) + E(\alpha_l, \beta_r) + E(\alpha_r, \beta_l). \tag{1}$$

Similarly, the partial energy sum $E(\alpha, \alpha)$ contributed by the interacting pairs inside the sub-chain corresponding to $\alpha$ can be decomposed as follows:

$$E(\alpha, \alpha) \quad = \quad E(\alpha_l, \alpha_l) + E(\alpha_r, \alpha_r) + E(\alpha_l, \alpha_r). \tag{2}$$

These two recursive equations yield the EnergyTree.

The EnergyTree has as many levels as the ChainTree. Its nodes at any level are all the pairs $(\alpha, \beta)$, where $\alpha$ and $\beta$ are nodes from the same level of the ChainTree. If $\alpha \neq \beta$ and they are not leaves of the ChainTree, then the node $(\alpha, \beta)$ of the EnergyTree has four children $(\alpha_l, \beta_l), (\alpha_r, \beta_r), (\alpha_l, \beta_r)$, and $(\alpha_r, \beta_l)$. A node $(\alpha, \alpha)$ has three children $(\alpha_l, \alpha_l), (\alpha_r, \alpha_r)$, and $(\alpha_l, \alpha_r)$. The leaves of the EnergyTree are all pairs of leaves of the ChainTree (hence, correspond to pairs of links of the protein chain). For simplification, Figure 12 does not show the leaves of the form $(\alpha, \alpha)$. Each node $(\alpha, \beta)$ of the EnergyTree holds the partial energy sum $E(\alpha, \beta)$ for the current conformation in the simulation. The root holds the total sum. In the EnergyTree of Figure 12, we have $E(P,P) = E(N,N)+E(N,O)+E(O,O)$, $E(N,O) = E(J,L)+E(J,M)+E(K,L)+E(K,M)$, and so on.

At each step, the testing algorithm is called to find new interacting pairs. During this process, whenever the algorithm prunes a search path, it marks the corresponding node of the EnergyTree to indicate that the energy sum stored at this node is unaffected. The energy sums stored in the EnergyTree are updated next. This is done by performing a recursive traversal of the tree. The recursion along each path ends when it reaches a marked node or when it reaches an unmarked leaf. In the second case, the sum held by the leaf is re-computed by adding all the energy terms corresponding to the interacting pairs previously found by the testing algorithm. When the recursion unwinds, the intermediate sums are updated using Equations (1) and (2). In practice, the testing algorithm and the updating of the EnergyTree are run concurrently, rather than sequentially.

To illustrate, assume that a 1-DOF change has been applied to the chain of Figure 6 between $F$ and $G$. In that case, the testing algorithm marks nodes $(N,N)$, $(J,L)$, $(K,L)$, $(L,L)$ and $(M,M)$ in the EnergyTree. The above recursion re-computes from scratch the partial sums at all the unmarked leaves it encounters and updates the partial sums of all other un-marked nodes it visits as the recursion unwinds using Equations (1) and (2).

The size of the EnergyTree grows quadratically with the number $n$ of links. For most proteins this is not a critical issue. For example, in our experiments, the memory used by the EnergyTree ranges from 0.4 MB for 1CTF ($n = 137$) to 50 MB for 1JB0 ($n = 1511$). If needed, however, memory could be saved by representing only those nodes of the EnergyTree which correspond to pairs of RSSs closer than the cutoff distance.

# 8 Experimental Results for MCS

## 8.1 Experimental setup

We extended ChainTree as described in the previous section. Since each step of an MCS may be rejected, we keep two copies of the ChainTree and the EnergyTree. RSS and distance computation routines were borrowed from the PQP library [23, 38].

We similarly extended Grid to find interacting pairs by setting the side length of the grid cubes to the cutoff distance. As we mentioned in Subsection 7.1, Grid finds all interacting pairs at each step, not just the new ones, and does not cache partial energy sums. So, it computes the new energy value by summing the terms contributed by all the interacting pairs.

Tests were run on a 400 MHz UltraSPARC-II CPU of a Sun Ultra Enterprise 5500 machine with 4.0 GB of RAM.

We performed MCS with the new ChainTree and Grid on the four proteins of Table 2. Unlike in the self-collision tests presented earlier, the side-chains were included in the models, as rigid groups of atoms (no internal DOF). So, if two leaf RSSs are within the cutoff distance, ChainTree finds the interacting pairs from the two corresponding links by examining all pairs of atoms. The energy function used we for these tests includes a van der Waals (vdW) potential with a cutoff distance of 6Å, an electrostatic potential with a cutoff of 10Å, and a native-contact attractive quadratic-well potential with a cutoff of 12Å. Hence, the cutoff distance for both ChainTree and Grid was set to 12Å.

Each simulation run consisted of 300,000 trial steps. The number $k$ of DOFs changed at each step was constant throughout a run. We performed runs with $k = 1, 5$ and 10. Each change was generated by picking $k$ backbone DOFs at random and changing each DOF independently with a magnitude picked uniformly at random between 0° and 12°. Each run started with a random, partially extended conformation of the protein. Since the vdW term for a pair of atoms grows as $O(d^{-12})$ where $d$ is the distance between the atom centers, it quickly approaches infinity as $d$ becomes small (steric clash). When a vdW term was detected to cross a very large threshold, the energy computation was halted (in both ChainTree and Grid), and the step was rejected.

We ran the same MCS with both methods on each protein, by starting at the same initial conformation and using the same seed of the random-number generator. Note that for any given protein conformation ChainTree and Grid compute the exact same energy value.

### 8.2 Results

The results for all the experiments are found in Tables 3 and 4. Illustrations of the average time results for $k = 1$ and $k = 5$ are presented in Figures 13a and 13b respectively. As expected, ChainTree gave its best results for $k = 1$, requiring on average one quarter of the time of Grid per step for the smallest protein (1CTF) and one twelfth of the time for the largest protein (1JB0). The average number of interacting pairs for which energy terms were evaluated at each step was almost 5 times smaller with ChainTree than with Grid for 1CTF and 30 times smaller for 1JB0.

We see similar results when $k = 5$. In this case, ChainTree was only twice as fast as Grid for 1CTF and 6 times faster for 1JB0. The average number of interacting pairs for which energy terms were evaluated was twice smaller with ChainTree for 1CTF and 14 times smaller for 1JB0.

When $k = 10$, the relative effectiveness of ChainTree declined further, being only 1.2 times faster than Grid for 1CTF and 4 times faster for 1JB0. The average number of interacting pairs for which energy terms were evaluated using ChainTree was 60% of the number evaluated using Grid for 1CTF and 10 times smaller for 1JB0.

These results are consistent with those obtained for self-collision detection (Section 6). The larger $k$, the less effective our algorithm compared with Grid. When $k$ is small, there are few new interacting pairs at each step, and ChainTree is very effective in exploiting this fact. For both ChainTree and Grid the average time per step decreases when $k$ increases. This stems from the fact that a larger $k$ is more likely to yield over-threshold vdW terms and so to terminate energy computation sooner.

In order to examine the full effect of reusing partial energy sums, we re-ran the simulations for the four proteins without the vdW threshold for $k = 1$ and 5. The results are presented in Tables 5 and 6. Removing the vdW threshold does not significantly alter the behavior of the algorithms. The average time per step is of course larger, since no energy computation is cut short by a threshold crossing. The relative speed-up of ChainTree over Grid is only slightly smaller without the threshold.

### 8.3 Two-pass ChainTree

In the previous MCS the percentage of steps that were rejected before energy computation completed, due to an above-threshold vdW term for 1CTF, for example, rose from 60% when $k = 1$ to 98% when $k = 10$. This observation not only motivates choosing a small $k$. It also suggests the following two-pass approach. In the first pass, ChainTree uses a very small cutoff distance chosen such that atom pairs closer than this cutoff yield above-threshold vdW terms. In this pass, the algorithm stops as soon as it finds an interacting pair, and then the step is rejected. In the second pass the cutoff distance is set to the largest cutoff over all energy terms and ChainTree computes the new energy value. We refer to the implementation of this two-pass approach as ChainTree+. Since ChainTree is much faster with a small cutoff and the first pass will often result in step rejection, we can expect ChainTree+ to be significantly faster than ChainTree. Thanks to the modifications described in Subsection 7.2, both passes use the same ChainTree data structure.

We compared ChainTree and ChainTree+ by running an MCS of 300,000 trial steps with $k = 5$ and measuring the average time per step. The results for the four proteins are given in Table 7. We ran two different simulations for each protein. One that started at a partially extended conformation and another that started at the folded state of the protein. Hence, the conformations reached in the first case were less compact than in the second case. Consequently, the rejection rate due to self-collision was higher in the second case. While ChainTree+ is faster in both cases, speed-up factors are greater (as much as 5) when starting from the folded state.

It is not clear whether a similar improvement could be obtained with Grid. Indeed, the resolution of the indexing grid depends on the cutoff distance. Indexing atoms in two different grids — one with small cells for detecting steric clashes and another with larger cells for computing the energy — may then considerably reduce the advantage of the two-pass approach.

## 9 MCS Software

We have extended our implementation of the ChainTree algorithm to include a physical, full-atomic energy function. We chose to implement the force-field *EEF1* [40]. This force field is based on the CHARMM19 potential energy function [8] with an added implicit solvent term. We chose *EEF1* because it has been shown to discriminate well between folded and misfolded structures [39]. It is well suited for ChainTree because its implicit solvent term is pairwise and thus our algorithm can compute it efficiently. We have packaged our software into a program that runs fast MCS. It can be downloaded from `http://robotics.stanford.edu/~itayl/mcs`.

This software loads an initial structure that is described in terms of its amino acid sequence and the corresponding backbone angles of each residue. It then performs a classical MCS. The user can control some parameters of the simulation (e.g. the number of angles to change, the length of the simulation, the temperature ...) by specifying them on the command line.

Our protein model uses fixed rotamers for each amino acid, and thus a given side-chain is limited to a fixed number of conformations during the simulation. The advantage of using fixed rotamers is that the atom positions and internal energy of each rotamer can be pre-computed. Hence switching between rotamers during the simulation is done extremely fast. We used the *backbone-independent rotamer library* of the Dunbrack lab [16].

# 10 Conclusion

## 10.1 Summary of contribution

This paper presents a novel algorithm based on the ChainTree and EnergyTree data structures to reduce the average step time of MCS of proteins, independent of the energy function, step generator, and acceptance criterion used by the simulator. Tests show that, when the number of simultaneous DOF changes at each step is small (as is usually the case in MCS), the new method is significantly faster than previous general methods — including the worst-case optimal grid method — especially for large proteins. This increased efficiency stems from the treatment of proteins as long kinematic chains and the hierarchical representation of their kinematics and shape. This representation — the ChainTree — allows us to exploit the fact that long sub-chains stay rigid at each step, by systematically re-using unaffected partial energy sums cached in a companion data structure — the EnergyTree. Our tests also demonstrate the advantage of using the ChainTree to detect steric clash before computing the energy function.

### 10.2 Other applications

Although we have presented the application of our algorithm to classical Metropolis MCS, it can also be used to speed up other MCS methods as well as other optimization and simulation methodologies.

For example, MCS methods that use a different acceptance criterion can benefit from the same kind of speed-up as reported in Section 8, since the speed-up only derives from the faster maintenance of the energy function when relatively few DOFs are changed simultaneously, and is independent of the actual acceptance criterion. Such methods include Entropic Sampling MC [42], Parallel Hyperbolic MC [67], and Parallel-hat Tempering MC [68]. MCS methods that use Parallel Tempering [31] (also known as Replica Exchange) such as [67, 68], which require running a number of replicas in parallel, could also benefit by using a separate ChainTree and EnergyTree for each replica.

Some MCS methods use more sophisticated move sets (trial step generators). Again, our algorithm can be applied when the move sets do not change many DOFs simultaneously, which is in particular the case of the moves sets proposed in [2, 3] (biasing the random torsion changes), and in [17] and [56] (moves based on fragment replacement). More computationally intensive step generators use the internal forces (the gradient of the energy function) to bias the choice of the next conformation (e.g., Force-Biased MC [50], Smart MC [34] and MC plus minimization [43]). For such step generators, the advantage of using our algorithm is questionable, since they may change all DOFs at each step. Similarly, the so-called Hybrid MC methods [7, 66] that combine MCS and Molecular Dynamics Simulation would benefit from our algorithm only when relatively few DOFs are changed at each step. The same is true for pure MDS in torsion-angle space [26, 59].

Some optimization approaches could also benefit from our algorithm. For instance, a popular one uses genetic algorithms with crossover and mutation operators [51, 60, 63]. The crossover operator generates a new conformation by combining two halves, each extracted from a previously created conformation. Most mutation operators also reuse long fragments from one or several previous conformations. For both types of operators, our algorithm would allow partial sums of energy terms computed in each fragment to be re-used, hence saving considerable amounts of computation.

## 10.3 Current and future work

We are currently using our MCS software to run MCS of proteins on the order of 100 residues and larger. To the best of our knowledge this has not been attempted so far. We also intend to perform MCS of systems of several small proteins, in order to study protein misfolding, which is known to cause diseases such as Alzheimer. Each protein in the system will have its own ChainTree, which will be used to detect interaction both within each molecule and between molecules.

To be physically realistic a simulation should take into account the solvent in which the protein is immersed. In MDS this is often done by adding solvent molecules into the system. However, such an explicit representation of the solvent would tremendously slow down MCS. Therefore, in MCS the solvent is often accounted for *implicitly* by adding an energy term and changing the parameters of other terms such as the electrostatic potential. A number of such implicit solvent terms have been proposed in the literature. Since our algorithm is most efficient when computing pairwise interactions, a solvent term such as the one incorporated into the EEF1 force field [40], that sums up contributions from pairs of atoms, is most suited for it. Another way of implicitly accounting for the solvent is to use the solvent accessible surface area of a protein to estimate the energetic contribution of the solvent (e.g. [11]). In this case the surface area of the molecule must be known at each simulation step. Two approximate methods for computing the surface area that could integrate well with our algorithm are (1) a method that collects surface area contributions from pairs of atoms and approximates the error introduced by ignoring three-way contributions [14], (2) a method that computes the surface area as a sum over the surface contribution of each atom computed approximately using arrangements of great circles on a sphere [29].

One possible extension of our work would be to use the ChainTree to help select “better” simulation steps. Indeed, the rejection rate in MCS becomes so high for compact conformations that simulation comes to a quasi standstill. This is a known weakness of MCS, which makes it less useful around the native conformation. This happens because almost any DOF change causes a steric clash. To select DOF changes less likely to create such clashes, one could pre-compute the radius of rotation of every link relative to each DOF. These radii and the distances between interacting atom pairs at each conformation would allow computing the range of change for each DOF such that no steric clash will occur.

Another approach to generating moves in a compact conformation, is to attempt local moves, that is, changes that affect only a short sub-chain while keeping its endpoints fixed. Good candidate sub-chains for this kind of change are those that have some free space in their vicinity, into which they can be deformed. The ChainTree could be used to quickly survey the neighborhood of an amino acid to determine whether there is any free space around it. Once a free space is found the sub-chain could be moved into it using inverse kinematics techniques [65, 69].

Another natural extension, which exploits the hierarchical nature of the ChainTree, is to vary the resolution of the molecular representation as MCS progresses. This could be accomplished by changing on the fly the level of the ChainTree that is considered the leaf level (the bottom level). For full atomic resolution, searches in the ChainTree would continue until reaching the absolute bottom level. If a coarser resolution can be tolerated, the search could be stopped at a higher level in the hierarchy, where each node represents one or some amino acids. A different energy function could then be used for each level of resolution. This scheme could entail large savings in CPU time in regions of the conformation space where the protein structure is not very compact, while not compromising precision in other regions when it is needed.

**Acknowledgements:** This work was partially funded by NSF ITR grant CCR-0086013 and a Stanford BioX Initiative grant. Work by Dan Halperin has been supported in part by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2000-26473 (ECG - Effective Computational Geometry for Curves and Surfaces) and contract No IST-2001-39250 (MOVIE - Motion Planning in Virtual Environments), by The Israel Science Foundation founded by the Israel Academy of Sciences and Humanities (Center for Geometric Computing and its Applications), and by the Hermann Minkowski – Minerva Center for Geometry at Tel Aviv University.

# References

[1] R. Abagyan and P. Argos. Optimal protocol and trajectory visualization for conformational searches of peptides and proteins. *J. Mol. Biol.*, 225:519–532, 1992.

[2] R. Abagyan and M. Totrov. Biased probability Monte Carlo conformational seraches and electrostatic calculations for peptides and proteins. *J. Mol. Biol.*, 235:983–1002, 1994.

[3] R. Abagyan and M. Totrov. Ab initio folding of peptides by the optimal-bias Monte Carlo minimization procedure. *J. Comp. Phys.*, 151:402–421, 1999.

[4] P. Agarwal, J. Basch, L. Guibas, J. Hershberger, and L. Zhang. Deformable free space tilings for kinetic collision detection. In B.R. Donald, K.M. Lynch, and D. Rus, editors, *Alg. and*

*Comp. Robotics: New Directions (WAFR '00)*, pages 83–96. A.K. Peters, Wellesley, 2000.

[5] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, and H. Weissig et al. The protein data bank. *Nucl. Acids Res.*, 28:235 – 242, 2000.

[6] K. Binder and D. Heerman. *Monte Carlo Simulation in Statistical Physics.* Springer Verlag, Berlin, 2nd edition, 1992.

[7] A. Brass, B. Pendelton, Y. Chen, and B. Robson. Hybrid Monte Carlo simulation theory and initial comparison with molecular dynamics. *Biopolymers*, 33:1307–1315, 1993.

[8] B. Brooks, R. Bruccoleri, B. Olafson, D. States, S. Swaminathan, and M. Karplus. CHARMM: a program for macromolecular energy minimizationand dynamics calculations. *J. Comput. Chem*, 4:187–217, 1983.

[9] J. Brown, S. Sorkin, J.-C. Latombe, K. Montgomery, and M. Stephanides. Algorithmic tools for real time microsurgery simulation. *Med. Image Anal.*, 6(3):289–300, 2002.

[10] S. Cameron. Collision detection by four-dimensional intersection testing. *IEEE Tr. Robotics Automat.*, 6(3):291–302, June 1990.

[11] W. Clark, A. Tempczyk, R. Hawley, and T. Hendrickson. Semianalytical treatment of solvation for molecular mechanics and dynamics. *J. of the American Chemical Society*, 112(16):6127–6129, 1990.

[12] J. D. Cohen, M. C. Lin, D. Manocha, and M. Ponamgi. I-COLLIDE: An interactive and exact collision detection system for large-scale environments. In *Sym. on Interactive 3D Graphics*, pages 189–196, 218, 1995.

[13] T. E. Creighton. *Proteins : Structures and Molecular Properties.* W. H. Freeman and Company, New York, 2nd edition, 1993.

[14] B. Dahiyat and S. Mayo. Pairwise calculation of protein solvent-accessible surface areas. *Science*, 278:82–87, 1997.

[15] D. Dobkin and D. Kirkpatrick. Determining the separation of preprocessed polyhedra – a unified approach. In *Proc. 17th Int. Colloq. Automata Lang. Program. Lecture Notes Comput. Sci.*, volume 443, pages 400–413, 1990.

[16] R. Dunbrack and F. Cohen. Bayesian statistical analysis of protein sidechain rotamer preferences. *Protein Sci.*, 6:1661–81, 1997.

[17] A. Elofsson, S. LeGrand, and D. Eisenberg. Local moves, an efficient method for protein folding simulations. *Proteins*, 23:73–82, 1995.

[18] B. Faverjon. Obstacle avoidance using an octree in the configuration space of a manipulator. In *IEEE Conf. on Rob. and Auto.(ICRA)*, pages 504–510, 1984.

[19] B. Faverjon. Object level programming of industrial robots. In *IEEE Conf. on Rob. and Auto. (ICRA)*, pages 1406 – 1412, 1986.

[20] A. Foisy and V. Hayward. A safe swept volume method for collision detection. In *The Sixth International Symposium of Robotics Research*, pages 61–68, Pittsburgh (PA), Oct. 1993.

[21] F. Ganovelli, J. Dingliana, and C. O'Sullivan. Buckettree: Improving collision detection between deformable objects. In *Spring Conf. on Comp. Graphics (SCCG2000)*, 2000.

[22] N. Gō and H. Abe. Noninteracting local-structure model of folding and unfloding transition in globular proteins. *Biopolymers*, 20:991–1011, 1981.

[23] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. *Comp. Graphics*, 30(Annual Conf. Series):171–180, 1996.

[24] A. Grosberg and A. Khokhlov. *Statistical physics of macromolecules.* AIP Press, New York, 1994.

[25] L. J. Guibas, A. Nguyen, D. Russel, and L. Zhang. Deforming necklaces. In *Symp. on Comp. Geometry (SoCG)*, pages 33–42, 2002.

[26] P. Güntert, C. Mumenthaler, and K. Wüthrich. Torsion angle dynamics for NMR structure calculation with the new program DYANA. *J. Mol. Biol.*, 273:283–298, 1997.

[27] D. Halperin, J.-C. Latombe, and R. Motwani. Dynamic maintenance of kinematic structures. In J.-P. Laumond and M. Overmars, editors, *Alg. for Rob. Motion and manip. (WAFR '96)*, pages 155–170. A.K. Peters, Wellesley, 1997.

[28] D. Halperin and M. H. Overmars. Spheres, molecules and hidden surface removal. *Comp. Geom.: Theory and App.*, 11(2):83–102, 1998.

[29] D. Halperin and C. Shelton. A perturbation scheme for spherical arrangements with application to molecular modeling. *Comp. Geom.: Theory and App.*, 10(4):273–288, 1998.

[30] H. Hansmann and Y. Okamoto. New Monte Carlo algorithms for protein folding. *Curr. Opin. Struct. Biol.*, 9(2):177–183, 1999.

[31] U. Hansmann. Parallel tempering algorithm for conformational studies of biological molecules. *Chem. Phys. Lett.*, 281:140–150, 1997.

[32] P. M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Tr. on Graphics*, 15(3):179–210, 1996.

[33] P. Jiménez, F. Thomas, and C. Torras. 3D collision detection: A survey. *Computers and Graphics*, 25(2):269–285, 2001.

[34] A. Kidera. Smart Monte Carlo simulation of a globular protein. *Int. J. Quant. Chem.*, 75:207–214, 1999.

[35] T. Kikuchi. Inter-Ca atomic potentials derived from the statistics of average interresidue distances in proteins: Application to bovine pancreatic trypsin inhibitor. *J. Comput. Chem*, 17(2):226–237, 1996.

[36] J. T. Klosowski, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Tr. on Visual. and Comp. Graphics*, 4(1):21–36, 1998.

[37] E. Kussell, J. Shimada, and E. Shakhnovich. A structure-based method for derivation of all-atom potentials for protein folding. *Proc. Natl. Acad. Sci.*, 99(8):5343–8, April 2002.

[38] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha. Fast distance queries with rectangular swept sphere volumes. In *IEEE Conf. on Rob. and Auto. (ICRA)*, 2000.

[39] T. Lazaridis and M. Karplus. Discrimination of the native from misfolded protein models with an energy funbction including implicit solvation. *J. Mol. Bio.*, 288:477–487, 1998.

[40] T. Lazaridis and M. Karplus. Effective energy function for proteins in solution. *Proteins*, 35:133–152, 1999.

[41] A. Leach. *Molecular Modelling: Principles and Applications.* Longman, Essex, England, 1996.

[42] J. Lee. New Monte Carlo algorithm: entropic sampling. *Phys. Rev. Lett.*, 71(2):211–214, 1993.

[43] Z. Li and H. Scheraga. Monte Carlo-minimization approach to the multiple-minima problem in protein folding. *Proc. Natl. Acad. Sci.*, 84(19):6611 – 6615, Oct. 1987.

[44] M. C. Lin and J. F. Canny. A fast algorithm for incremental distance calculation. In *IEEE Int. Conf. on Rob. and Auto. (ICRA)*, pages 1008–1014, 1991.

[45] I. Lotan, F. Schwarzer, D. Halperin, and J.-C. Latombe. Efficient maintenance and self-collision testing for kinematic chains. In *Symp. Comp. Geo.*, pages 43–52, 2002.

[46] I. Lotan, F. Schwarzer, and J.-C Latombe. Efficient energy computation for Monte-Carlo simulation of proteins. In *Workshop on Algorithms for Bioinformatics (WABI)*, Budapest, September 2003.

[47] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21:1087–1092, 1953.

[48] M. Mezei. A near-neighbor algorithm for metropolis Monte Carlo simulation. *Molecular Simulations*, 1:169–171, 1988.

[49] S. Northrup and J. McCammon. Simulation methods for protein-structure fluctuations. *Biopolymers*, 19(5):1001–1016, 1980.

[50] C. Pangali, M. Rao, and B. J. Berne. On a novel Monte Carlo scheme for simulating water and aqueous solutions. *Chem. Phys. Lett.*, 55(3):413–417, 1978.

[51] J. Pedersen and J. Moult. Protein folding simulations with genetic algorithms and a detailed molecular description. *J. Mol. Biol.*, 269(2):240–259, 1997.

[52] S. Quinlan. Efficient distance computation between non-convex objects. In *IEEE Int. Conf. on Rob. and Auto. (ICRA)*, pages 3324–3329, 1994.

[53] F. Schwarzer, M. Saha, and J.-C. Latombe. Exact colision detection of robot paths. In *Proc. 5th Workshop on Alg. Found. Rob.*, 2002.

[54] J. Shimada, E. Kussell, and E. Shakhnovich. The folding thermodynamics and kinetics of crambin using an all-atom Monte Carlo simulation. *J. Mol. Biol.*, 308(1):79–95, April 2001.

[55] J. Shimada and E. Shakhnovich. The ensemble folding kinetics of protein G from an all-atom Monte Carlo simulation. *Proc. Natl. Acad. Sci.*, 99(17):11175–80, August 2002.

[56] K. Simons, C. Kooperberg, E. Huang, and D. Baker. Assembly of protein tertiary structure from fragments with similar local sequences using simulated annealing and bayesian scoring functions. *J. Mol. Biol.*, 268:209–225, 1997.

[57] M. Soss, J. Erickson, and M. Overmars. Preprocessing chains for fast dihedral rotations is hard or even impossible. *Comp. Geom.: Theory and App.*, 26(3):235–246, 2003.

[58] M. Soss and G. Toussaint. Geometric and computational aspects of polymer reconfiguration. *J. Math. Chem.*, 27(4):303 – 318, 2000.

[59] E. Stein, L. Rics, and A. Brünger. Torsion-angle molecular dynamics as a new efficient tool for NMR structure calculation. *J. Magn. Reson.*, 124:154–164, 1997.

[60] S. Sun. Reduced representation model of protein structure prediction: statistical potential and genetic algorithms. *Protein Science*, 2(5):762–785, 1993.

[61] S. Sun, P. Thomas, and K. Dill. A simple protein folding algorithm using a binary code and secondary structure constraints. *Protein Eng.*, 8:769–778, 1995.

[62] S. Thompson. Use of neighbor lists in molecular dynamics. *Information Quaterly, CCP5*, 8:20 – 28, 1983.

[63] R. Unger and J. Moult. Genetic algorithm for protein folding simulations. *J. Mol. Biol.*, 231:75–81, 1993.

[64] G. van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *J. Graphics Tools*, 2(4):1–13, 1997.

[65] W. Wedemeyer and H. Scheraga. Exact analytical loop closure in proteins using polynomial equations. *J. Comput. Chem*, 20(8):819–844, 1999.

[66] H. Zhang. A new hybrid Monte Carlo algorithm for protein potential function test and structure refinement. *Proteins*, 34:464–471, 1999.

[67] Y. Zhang, D. Kihara, and J. Skolnick. Local energy landscape flattening: Parallel hyperbolic Monte Carlo sampling of protein folding. *Proteins*, 48:192–201, 2002.

[68] Y. Zhang and J. Skolnick. Parallel-hat tempering: A Monte Carlo search scheme for the identification of low-energy structures. *J. Chem. Phys.*, 115(11):5027–5032, 2001.

[69] Q. Zheng, R. Rosenfeld, S. Vajda, and C. DeLisi. Loop closure via bond scaling and relaxation. *J. Comput. Chem*, 14:556–565, 1992.

[70] Y. Zhou and S. Suri. Collision detection using bounding boxes: Convexity helps. In *8th Annual European Symposium on Algorithms (ESA 2000)*, pages 437–448, 2000.

## Appendix: Proof of Theorem 1

In this appendix we give a proof of Theorem 1 stated in Section 5.2. Along the way we establish successive useful lemmas.

In order to simplify our proof, we first "regularize" a well-behaved kinematic chain as follows:

1. We replace each link by an enclosing sphere whose radius is equal to that of the largest minimal enclosing sphere of any link in the chain.

2. We grow all these spheres equally until no two consecutive spheres in the chain are disjoint (of course some or many of the spheres may already be intersecting).

So, the links of the new chain are spheres of equal radius $r$. The new chain is also well-behaved since the regularization only change the size of the enclosing spheres by a constant factor, when the distance between the centers of the enclosing spheres of any two successive links of the original chain is lower bounded by a constant (which is the case for proteins). Therefore, the maximal number of links of the new chain that can overlap a single link is still bounded by a constant, though this constant may be greater than the one for the original chain. None of the asymptotic bounds below are affected by this change of constant.

From now on, we only consider the regularized chain. We distinguish between two types of BVH for this chain. In the *tight* hierarchy, each BV tightly bounds the links it encloses. In the *not-so-tight* one, each non-leaf BV bounds tightly the two BVs just below it. In a tight sphere hierarchy, each bounding sphere is the minimum enclosing sphere of the links it encloses.

Proposition 1 and Lemmas 1 through 3 establish the upper bound of Theorem 1. Proposition 2 then presents a chain conformation for which this bound is actually attained, hence showing that the bound is tight.

**Proposition 1** *In the tight, chain-aligned sphere hierarchy of a well-behaved $n$-link chain, the maximum total number of overlapping bounding spheres at all levels is $O(n^{\frac{4}{3}})$.*

**Proof:** At level $i$ of the hierarchy, where $i = 0, 1, \ldots, \log n$, with $i = 0$ being the lowest level, each consecutive sub-chain of $g_i = 2^i$ links is bounded by a single sphere of radius at most $g_i r$. This radius occurs when the links are arranged on a straight line. Take any such sub-chain of $g_i$ links and let $B_i$ denote its bounding sphere. Let $C_i$ be the sphere concentric with $B_i$ and having radius $3g_ir$. Any bounding sphere at level $i$ intersecting $B_i$ is fully contained in $C_i$.

Now let us bound the number of sub-chains of $g_i$ links whose bounding spheres at level $i$ can be contained in $C_i$. Since we aim for an upper bound we assume that each such sub-chain is as tightly packed as possible. Because each link can overlap at most a constant number of other links the volume occupied by a sub-chain of length $g_i$ is bounded from below by $qg_i\frac{4}{3}\pi r^3$, where $0 < q \leq 1$ is a constant. Hence, the number of bounding spheres at level $i$ contained in $C_i$ is at most:

$$M_i \;=\; \frac{\frac{4}{3}\pi 27 g_i^3 r^3}{\frac{4}{3}qg_i\pi r^3} \;=\; \frac{27g_i^2}{q} \tag{3}$$

$M_i$ is therefore an upper bound on the number of bounding spheres at level $i$ that overlap $B_i$. Since there are exactly $n/g_i$ bounding spheres at level $i$, $M_i$ cannot grow larger than $\frac{n}{g_i}$. The level $i_{max}$ where this bound is reached is the smallest $i$ defined by:

$$\frac{n}{g_i} \geq \frac{27g_i^2}{q}.$$

Since $g_i = 2^i$, we get:

$$i_{max} = \lceil \frac{1}{3}\log n + \log \frac{1}{3}q^{\frac{1}{3}} \rceil. \tag{4}$$

For every $i < i_{max}$, $T_i = \frac{n}{g_i}M_i$ is an upper bound on the number of overlapping bounding spheres at level $i$. For every $i \geq i_{max}$, $T_i = (\frac{N}{g_i})^2$ is an upper bound on this number. In what follows, we ignore the constant term on the right-hand side of Equation (4) as it has no effect on the asymptotic bound we prove. The total number of overlapping bounding spheres at all levels is therefore:

$$T \;=\; \sum_{i=0}^{\log n} T_i$$

$$
\begin{aligned}
&= \sum_{i=0}^{\frac{1}{3}\log n} \left(\frac{27g_i^2}{q}\right)\left(\frac{n}{g_i}\right) + \sum_{i=\frac{1}{3}\log n}^{\log n} \left(\frac{n}{g_i}\right)^2 \\
&= \frac{27n}{q} \sum_{i=0}^{\frac{1}{3}\log n} 2^i + n^2 \sum_{\frac{1}{3}\log n}^{\log n} \left(2^{-i}\right)^2 \\
&= \frac{27n}{q}\left(2n^{\frac{1}{3}} - 1\right) + \frac{4}{3}\left(n^{\frac{4}{3}} - 1\right) \\
&= O\left(n^{\frac{4}{3}}\right) \qquad (5)
\end{aligned}
$$

□

As a side note, this proof extends with minor changes to spaces of any dimension $d \geq 3$. We then get an upper bound of $O\left(n^{\frac{2(d-1)}{d}}\right)$.

One should note that the upper bound of Proposition 1 holds for any chain-aligned BVH (tight or not) as long as each BV at level $i$ can be enclosed in a sphere of radius $c\,2^i r$, for an absolute constant $c$. We now use this remark to show that the upper bound holds for the OBB hierarchy of the ChainTree.

**Lemma 1** *Given two OBBs contained in a sphere $D$ of radius $R$, the OBB bounding both of them is contained in a sphere of radius $\sqrt{3}R$ concentric with $D$.*

**Proof:** We let $b_1$ and $b_2$ denote the two OBBs contained in $D$ and $B_{12}$ denote their OBB. Construct the bounding cube $Q$ of $D$ whose faces are parallel to those of $B_{12}$. $Q$ contains $B_{12}$ since along any of the main axes that define $B_{12}$, the faces of $Q$ are farther out (or touching). So, the bounding sphere $E$ of $Q$, which is concentric with $D$, also contains $B_{12}$. Since each side of $Q$ has length $2R$, its diagonal has length $2\sqrt{3}R$ and the radius of $E$ is $\sqrt{3}R$. □

**Lemma 2** *At level $i$ of the not-so-tight, chain-aligned OBB hierarchy of a well-behaved $n$-link chain, each OBB is contained in a sphere of radius $c\,2^i r$, where $c$ is an absolute constant.*

**Proof:** Let us choose $c_1$ such that each OBB at levels $i = 0, 1, \ldots, 4$ of the hierarchy is contained in a sphere of radius $c_1\,2^i r$. We know $c_1$ exists since there are at most 16 spherical links enclosed by each OBB at these levels. We will take the constant $c$ to be at least $c_1$.

We now proceed by induction, by assuming that the lemma is correct up to some level $i-1$ $(i \geq 5)$. Consider 32 consecutive OBBs $b_j$, $j = 0, \ldots, 31$, at level $i-5$ that are bounded together in an OBB of level $i$. The induction hypothesis implies that the bounding sphere of each $b_j$ has radius at most $c\,2^{i-5}r$. We take a sphere $S$ of radius $2^i r$ that contains the sub-chain bounded by the $32 b_j$ boxes. Now each sphere bounding one of the $b_j$ boxes must intersect $S$ since at least one link is contained in both spheres. So, no point of box $b_j$ is further away than $2^i r(1 + \frac{c}{16})$ from the center of $S$.

Let $S_0$ be the sphere of radius $2^i r(1 + \frac{c}{16})$ concentric with $S$. All boxes $b_j$ are contained in $S_0$. Consider pairs which will be bounded together at the next level $(i-4)$. We apply Lemma 1 to those pairs and realize that a sphere $S_1$ of radius $\sqrt{3}$ times the radius of $S_0$ and concentric with $S_0$ bounds all OBBs at level $i-4$. Continuing this line of reasoning up to level $i$ we get that a sphere $S_5$ of radius $\sqrt{3}^5\,2^i r(1 + \frac{c}{16})$ contains the OBB at level $i$ that encloses all of the boxes $b_j$. We must set $c$ such that this radius is smaller than $c\,2^i r$. Thus, $c$ must be such that:

$$\begin{aligned} 2^i r\left(1 + \frac{c}{16}\right)\sqrt{3}^5 &\leq c\,2^i r \\ c\left(\frac{1}{\sqrt{3}^5} - \frac{1}{16}\right) &\geq 1 \end{aligned} \tag{6}$$

So we choose:

$$c = \max\{\left(\frac{1}{\sqrt{3}^5} - \frac{1}{16}\right)^{-1}, c_1\}.$$

□

Lemma 2 assures us that the OBBs in the OBB hierarchy of the ChainTree do not become too big as we climb up the hierarchy, so that the upper bound of Proposition 1 applies to the ChainTree.

**Lemma 3** *In the not-so-tight, chain-aligned OBB hierarchy of a well-behaved $n$-link chain, the maximum number of overlapping bounding boxes at all levels is $O(n^{\frac{4}{3}})$.*

**Proof:** We have noted previously that the proof given for Proposition 1 holds for any chain-aligned BVH (tight or not) of a well-behaved $n$-link chain, as long as each BV at level $i$ can be enclosed in a sphere of radius $c\,2^i r$, for an absolute constant $c$. Lemma 2 establishes that this is verified by the not-so-tight, chain-aligned OBB hierarchy of the chain. □

This completes the proof of Proposition 1, thus establishing the upper bound of Theorem 1. We now show that this bound can be attained.

**Proposition 2** *There exists a well-behaved chain of $n$ links such that the number of BV overlap tests needed to detect self-collision in certain conformations is $\Omega(n^{\frac{4}{3}})$ for any chain-aligned BVH of convex BVs.*

**Proof:** We prove the lemma by presenting a chain conformation that requires this much work no matter what the BVs are as long as they are convex. Given some coordinate frame, we take the first $d$ links of the chain and place them along the $X$ axis starting at the origin and proceeding in the positive direction. The spheres are osculating and the center of the first sphere is at the origin. The center of the $d$th link is therefore at $x_0 = 2(d-1)r$. We now place the next $d$ links of the chain parallel to the $Y$ axis, starting at $(x_0, 2r, 0)$ and going in the positive direction. Next, we place the next $d$ links parallel to the $Z$ axis starting just above where the previous $d$-link sub-chain ended. We call this sub-chain of $3d$ links a *unit.* With the next additional links of the chain we create $\frac{d}{8} - 1$ more units, each in reverse order of the previous one and translated by $(2r, -2r, 0)$ relative to the previous one. We call these $\frac{d}{8}$ units a *layer.* Figure 14 shows one such layer. Finally, with the rest of the chain, we create $\frac{d}{8} - 1$ layers, each in reverse order of the previous one and translated by $(0, -2r, 2r)$ relative to the previous one.

We noticed that the convex hull of each unit contains the point $(2(d-1)r, (d-1)r, \frac{1}{4}(d-1)r)$. So, all these convex hulls are pairwise intersecting and any hierarchy of convex BVs will therefore contain that many intersecting pairs of BVs at the level where all the links of each unit are enclosed together in one BV. Since we created $\frac{d^2}{64}$ units in total, each with $3d$ links, so have $\frac{3d^3}{64} = n$. Hence, $d$ is a small constant times $n^{\frac{1}{3}}$. The $\frac{d^2}{64}$ units yield $\Omega(n^{\frac{2}{3}})$ convex hulls and therefore $\Omega(n^{\frac{4}{3}})$ intersecting pairs of convex hulls. □

Together, Lemma 3 and Proposition 2 prove Theorem 1.

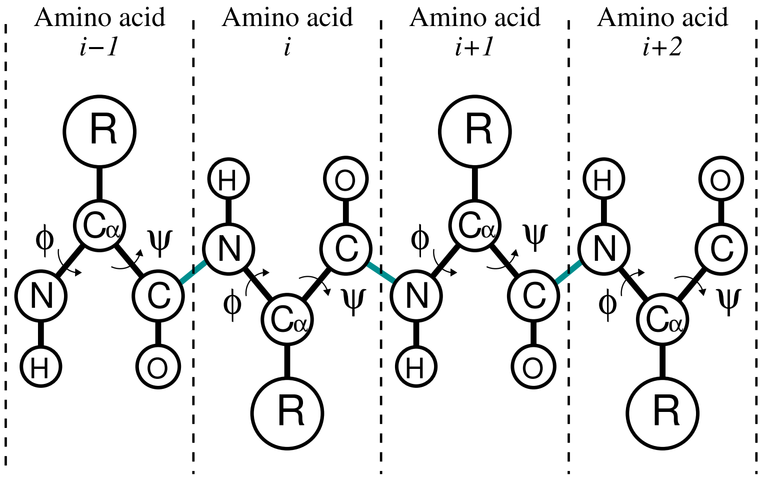


Figure 1: An illustration of a protein fragment with its backbone DOFs. *R* represents any side-chain

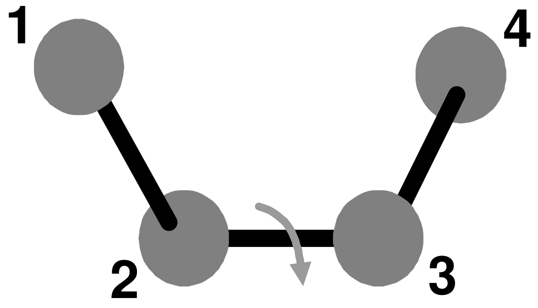


Figure 2: A torsional DOF: the angle between the plane of atoms 1, 2 and 3 and the plane of atoms 2, 3 and 4.

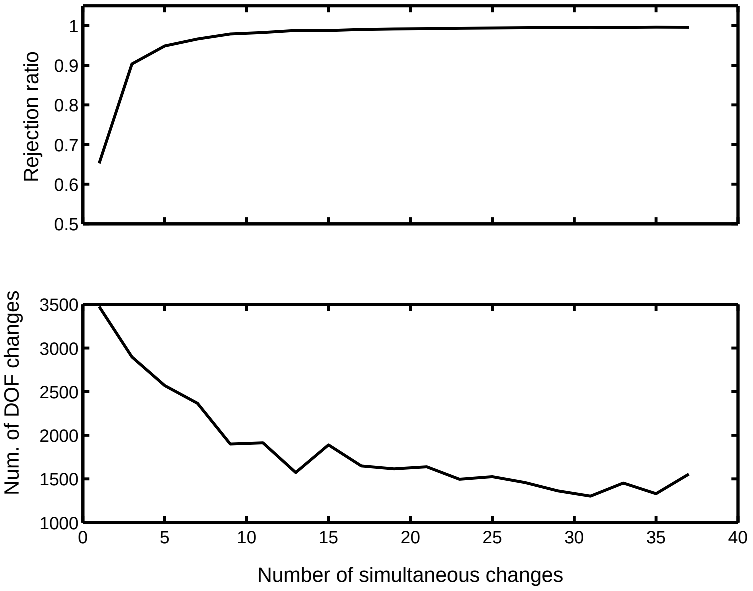


Figure 3: Effect of the number of simultaneous DOF changes. Rejection ration above and total number of changes below.

.

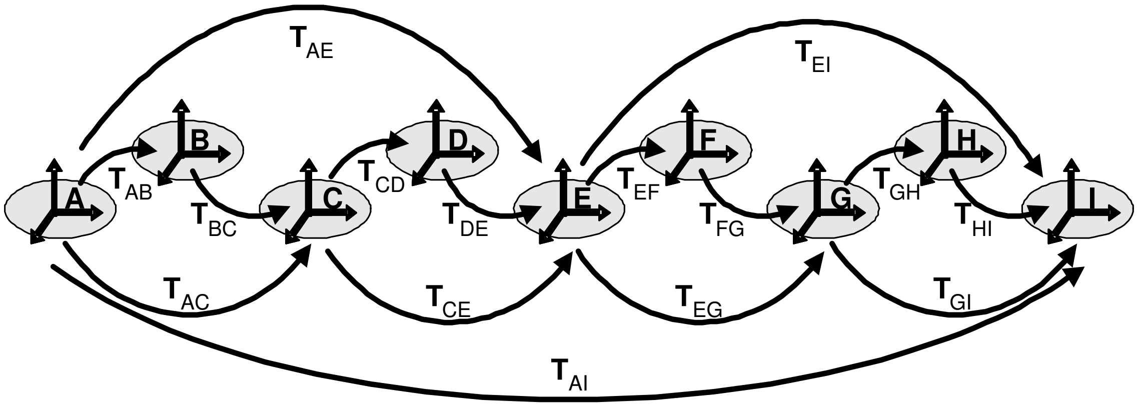


Figure 4: Transform hierarchy: grey ovals depict links; $T_{\alpha\beta}$ denote rigid-body transforms between the reference frames $\alpha$ and $\beta$.

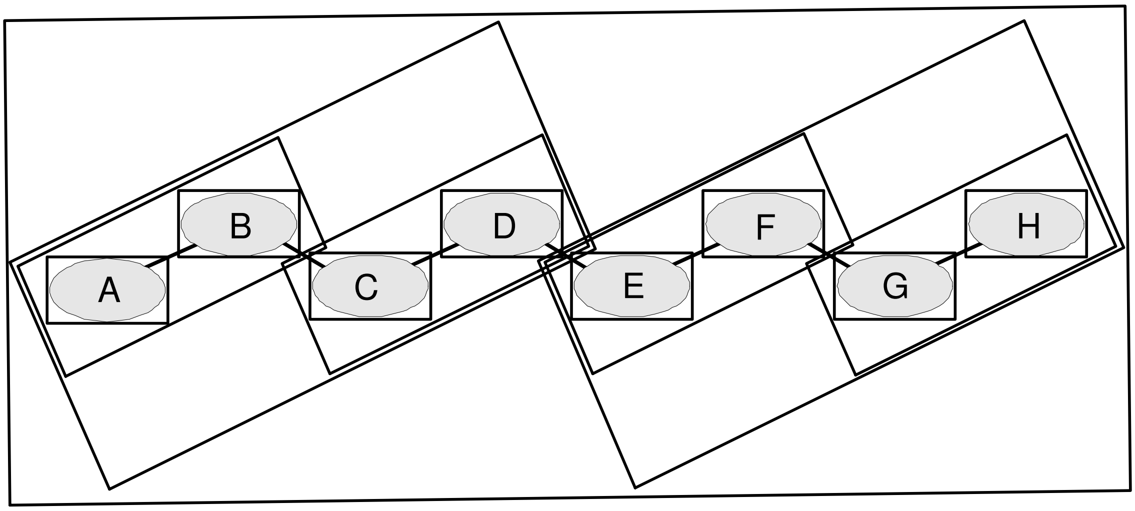


Figure 5: BVH: each box (OBB) approximates the geometry of a chain-contiguous sequence of links.

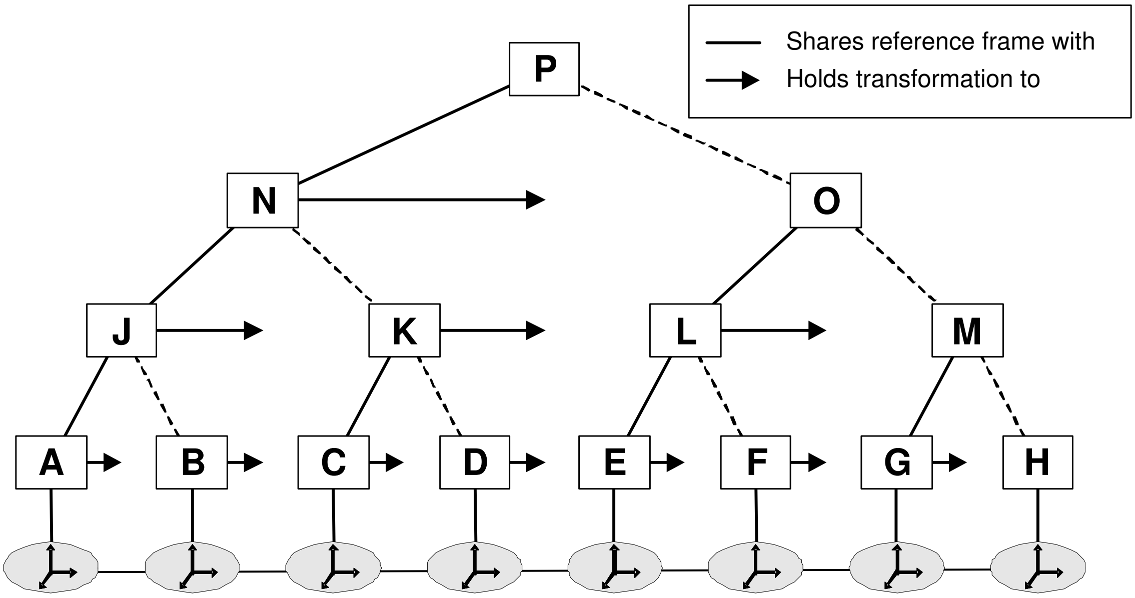


Figure 6: A binary tree that combines the transform and BV hierarchies.

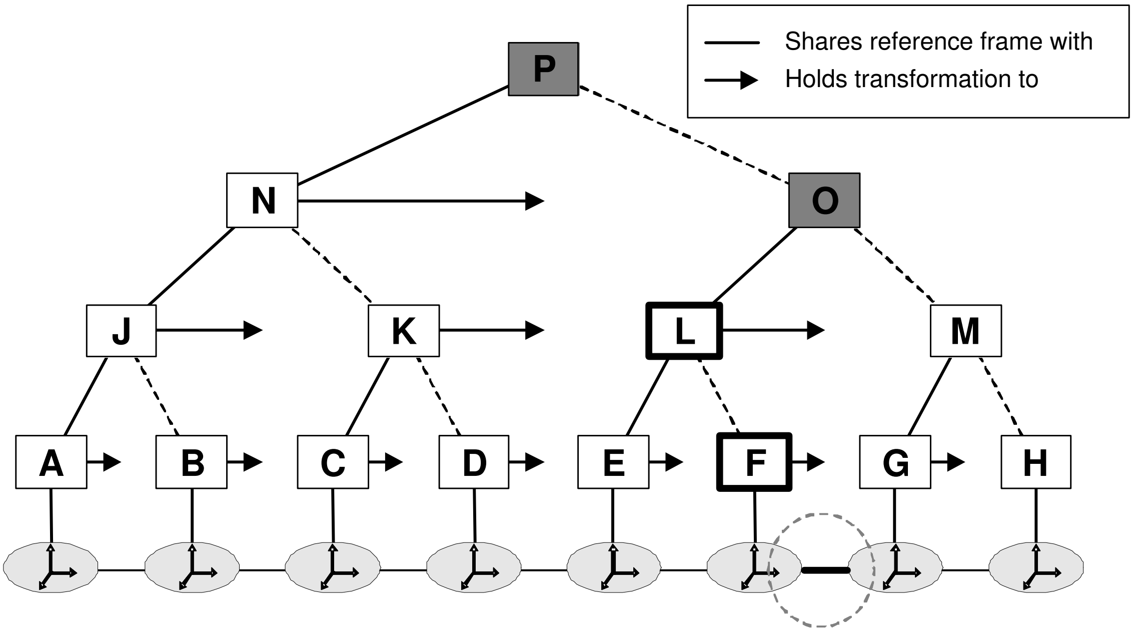
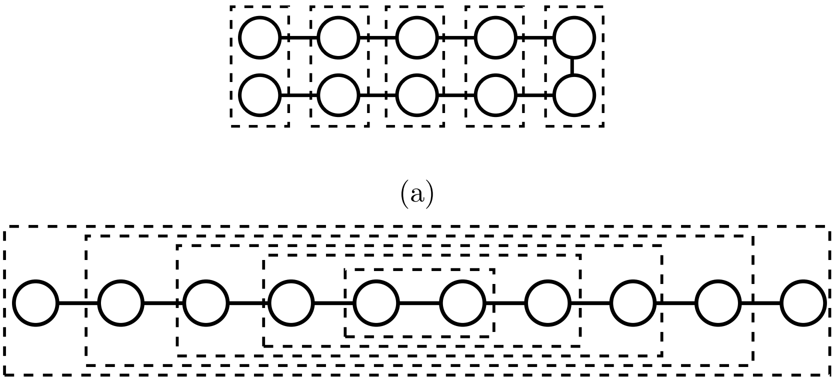


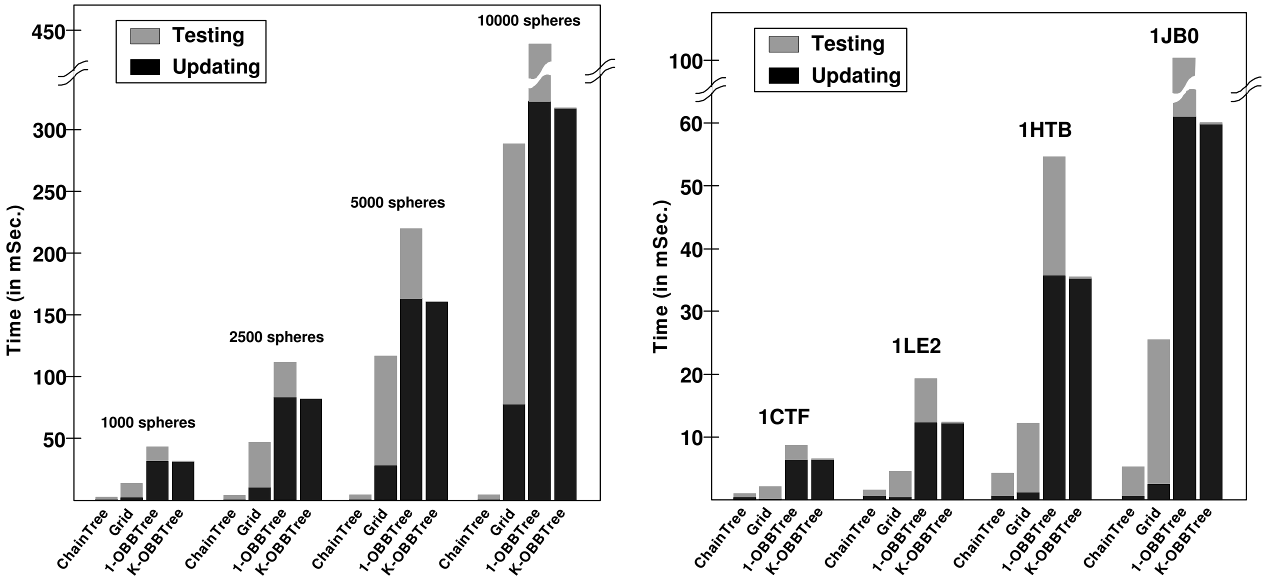
Figure 7: The ChainTree after applying a 1-DOF perturbation. Highlighted nodes were updated



(a)

(b)

Figure 8: A 1-DOF change corrupting a spatially-adapted hierarchy. (a) Before change. (b) After 1-DOF change. Only second level from bottom is shown ($\frac{n}{2}$ boxes).



(a)

(b)

Figure 9: Average time per step for detecting self-collision in (a) a compact pseudo-protein chain and (b) backbones of different proteins.

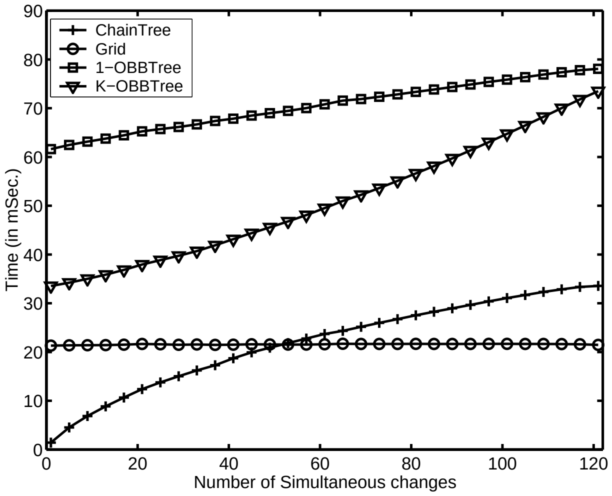


Figure 10: The effect of increasing the number of simultaneous DOF changes on the running time of the four algorithms.

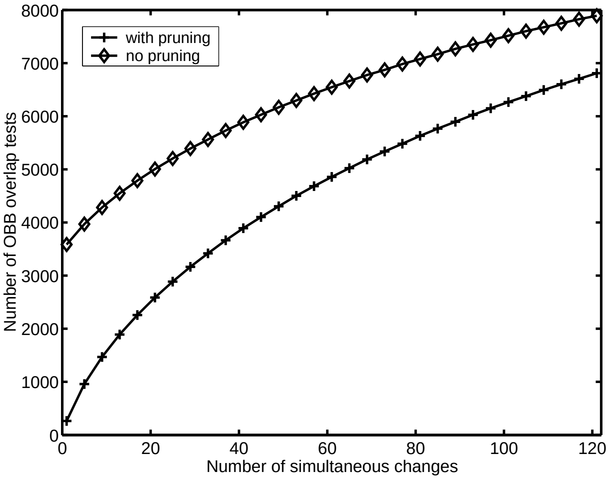


Figure 11: The average numbers of box overlap tests by ChainTree with and without search pruning.

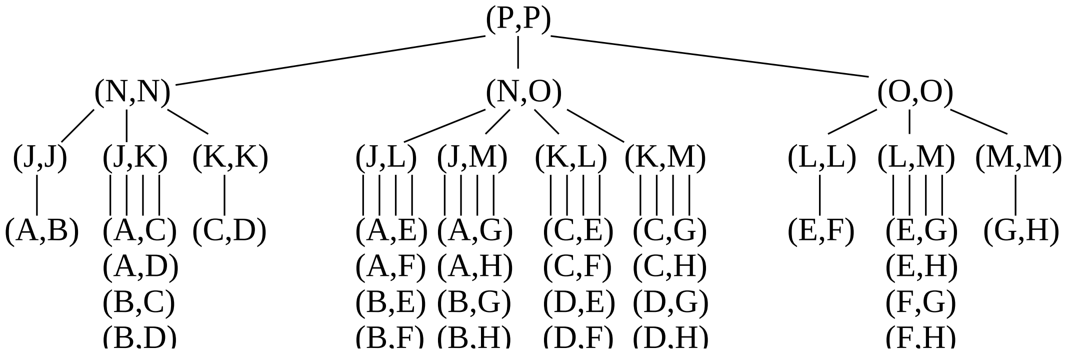
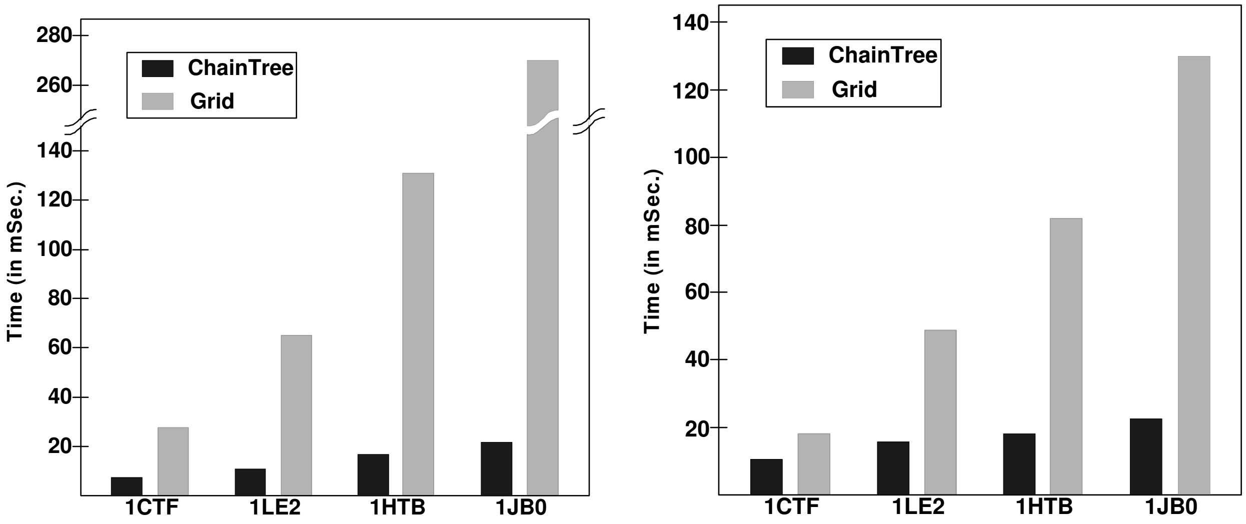


Figure 12: EnergyTree for the ChainTree of Figure 6. Leaves of the form $(\alpha, \alpha)$ are not shown.



(a) (b)

Figure 13: Comparing the average time per MCS step of ChainTree and Grid when (a) $k = 1$ and (b) $k = 5$.

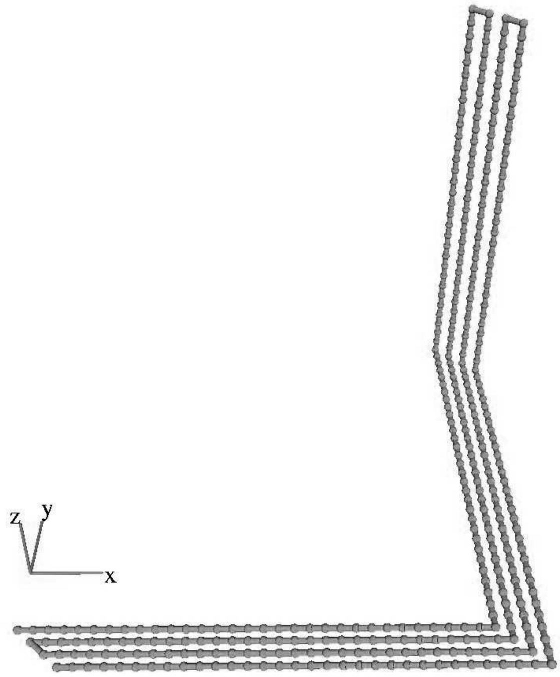


Figure 14: One layer of the chain construction.

|  | Update | Detection |
|---|---|---|
| Brute force | $\Theta(n)$ | $\Theta(n^2)$ |
| Grid | $\Theta(n)$ | $\Theta(n)$ |
| Spatially-adapted hierarchy | $O(n \log n)$ | $\Theta(n)$ |
| Chain-aligned hierarchy | $O(n)$ | $\Theta(n^{\frac{4}{3}})$ |
| ChainTree | $O(k \log \frac{n}{k})$ | $\Theta(n^{\frac{4}{3}})$ |

Table 1: Comparison of complexity measures for updating and detecting self-collision in well-behaved chains.

| Protein | #Amino Acids | #Atoms | # Backbone atoms | #Links |
|---|---|---|---|---|
| **1CTF** | 68 | 487 | 204 | 137 |
| **1LE2** | 144 | 1167 | 432 | 289 |
| **1HTB** | 374 | 2776 | 1112 | 749 |
| **1JB0** | 755 | 5878 | 2265 | 1511 |

Table 2: Proteins used in experiments.

|  | $k = 1$ |  | $k = 5$ |  | $k = 10$ |  |
|---|---|---|---|---|---|---|
|  | ChainTree | Grid | ChainTree | Grid | ChainTree | Grid |
| 1CTF | 7.82 | 27.7 | 8.34 | 18.22 | 12.57 | 15.07 |
| 1LE2 | 11.16 | 65.05 | 14.31 | 48.84 | 14.29 | 27.12 |
| 1HTB | 16.72 | 130.9 | 18.2 | 81.86 | 21.75 | 60.33 |
| 1JB0 | 21.71 | 271.4 | 22.18 | 130.5 | 29.88 | 133.8 |

Table 3: MCS results: average time per simulation step (in milliseconds)

|  | $k = 1$ |  | $k = 5$ |  | $k = 10$ |  |
|---|---|---|---|---|---|---|
|  | ChainTree | Grid | ChainTree | Grid | ChainTree | Grid |
| 1CTF | 5,100 | 25,100 | 7,400 | 16,900 | 8,000 | 13,500 |
| 1LE2 | 5,100 | 48,500 | 6,000 | 36,600 | 7,700 | 23,400 |
| 1HTB | 5,400 | 100,000 | 7,000 | 56,800 | 8,200 | 43,100 |
| 1JB0 | 5,900 | 200,000 | 7,000 | 95,600 | 10,300 | 102,000 |

Table 4: MCS results: number of interacting pairs for which energy terms were evaluated, per step.

|  | $k = 1$ |  | $k = 5$ |  |
|---|---|---|---|---|
|  | ChainTree | Grid | ChainTree | Grid |
| 1CTF | 12.8 | 37.2 | 29.6 | 37.7 |
| 1LE2 | 20.9 | 86.5 | 24.6 | 65.4 |
| 1HTB | 26.6 | 185 | 51.8 | 173 |
| 1JB0 | 40.0 | 401 | 89.1 | 348 |

Table 5: MCS results without threshold on vdW terms: average time (in milliseconds) per step.

|  | $k = 1$ |  | $k = 5$ |  |
|---|---|---|---|---|
|  | ChainTree | Grid | ChainTree | Grid |
| 1CTF | 8,600 | 33,300 | 21,000 | 34,700 |
| 1LE2 | 9,900 | 61,900 | 11,400 | 47,500 |
| 1HTB | 9,900 | 134,000 | 21,500 | 129,000 |
| 1JB0 | 12,000 | 280,000 | 30,300 | 248,000 |

Table 6: MCS results without a threshold on the vdW terms: average number of interacting pairs for which energy terms were evaluated, per step.

|  | unfolded |  | folded |  |
|---|---|---|---|---|
|  | ChainTree | ChainTree+ | ChainTree | ChainTree+ |
| 1CTF | 8.34 | 2.6 | 15.74 | 6.2 |
| 1LE2 | 14.31 | 6.4 | 32.37 | 9.06 |
| 1HTB | 18.2 | 9.23 | 68.92 | 11.35 |
| 1JB0 | 22.18 | 6.33 | 81.15 | 15.51 |

Table 7: Average running times (in mSec.) of ChainTree and ChainTree+ per step.