Motion Simulation of Geometric Constraint Structures

Felicia Cordeiro

This thesis was prepared under the guidance of Professor Audrey St. John.

Presented to the faculty of Mount Holyoke College in partial fulfillment of the requirements for the degree of Bachelor of Arts with Honors

Department of Computer Science

South Hadley, Massachusetts
May 2012

I give permission for public access to my thesis and for any copying to be done at discretion of the archives librarian and/or the College librarian.

May 9, 2012 .....

Felicia Cordeiro

Acknowledgements

I would like to thank my thesis advisor, Audrey St.John, for her hard work, guidance, and for her high expectations and advice about how to get there.

I would also like to thank all my Computer Science and Math Professors for their support.

I would like to thank my good friend Phoebe for introducing me to rock climbing and for knowing just when I needed a friend to pull me from my studies to the gym, and my pals Lucia and Liz Mac for all the great laughs! And I can't forget Rittika and Ilene for their continuous support, confidence and friendship; you are like sisters to me.

Finally, I want to thank my student colleagues, the track and field team, my coaches, and all my other good friends, for all their help and confidence in me.

Abstract

Motion simulation is a classical problem in areas of research such as CAD (Computer Aided Design), robotics, and protein folding and flexibility. Efficient motion simulation techniques could facilitate advancements in many such domains. Improvements in CAD motion simulation would offer mechanical engineers more quantitative feedback about the components of their mechanisms resulting in faster development. A model of protein motion and flexibility could help predict how molecules will interact, which may be able to considerably reduce trial and error for researchers attempting to design drugs that could ameliorate or cure various diseases.

While sophisticated motion simulation techniques exist, most have computational limitations. FEA (Finite Element analysis) allows engineers to study structural performance of their mechanisms. However, the growing complexity in models requires newer techniques for motion simulation. Molecular dynamics is widely recognized as the most accurate method for simulating the motion of proteins; however, similarly, it is computationally very expensive and, in some cases, requires prohibitive computing power and resources.

Rather than relying on current, computationally expensive techniques, we worked to help distill the intricacies of structural motion by using geometric constraints to model structural interactions. This level of abstraction has facilitated development of more efficient motion simulation techniques, through incorporating methodologies from other domains, such as the video game industry. In particular, we focused on generalizing the approach of Ragdoll physics.

In this thesis, we will describe the software tool we developed to simulate the motion of geometric constraint structures and new strategies for doing so. We will also explain and analyze the experimental results from applying our different techniques and comparing them to some current methods, specifically those of Ragdoll physics. For certain strategies and classes of structures, we found that our techniques outperform existing ones. These results indicate that our work may lead to algorithms that improve simulation for molecular motion.

Contents

List of Figures

Chapter 1

Introduction

Motion simulation entails computationally generating parameters encoding feasible motions within the domain and range of a given application. The goal of motion simulation is to provide a highly reality-related portrayal of actual movement events, but on a computer, at, or faster than, real-time. Simulation allows observation of a model, instead of testing an actual system, which would be too time consuming, too expensive, nearly impossible or dangerous. Simulation using an accurate model can project future effects or alternative results, routes, or strategies, or allow designs of systems to be tested and evaluated before building them.

Advancements in technology are continual, and the role of technology in research continues to be central and critical. The growing integration of technology with most fields of study all but demands the incorporation of simulation into every field of research. Models are used in the armed forces, to simulate battles and train soldiers on land, sea, and air, in flight simulation, to train pilots, in video games, for more realistic interactions

among characters, in the automotive industry, to train novice truck drivers, in medicine, to simulate surgical procedures, in biology, to simulate the movement and interaction of proteins and other molecules, and in many more venues, with many more actions. Simulation in general, and motion simulation in particular, is often considered a bridge or link between theory and experiment since it allows hypotheses and theories about motion to be tested [11].

1.1 Motivation

Motion simulation is a classical problem in areas of research such as CAD (Computer Aided Design) software design, robotics, and protein folding (by predicting their flexibilities). Efficient motion simulation techniques, specifically for geometric constraint structures, could facilitate advancements in many such domains.

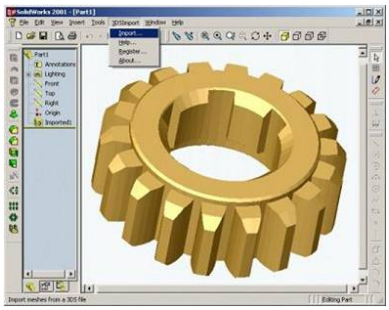


Figure 1.1: An example of a CAD part being created in SolidWorks. Figure reproduced from http://www.prlog.org/10341476-solidworks-3d-modeling-services.jpga.

CAD software is used by mechanical engineers to create sophisticated designs. CAD can be used to make designs or objects in either 2-dimensional or 3-dimensional space; see Figure 1.1. CAD is commonly used to create floor plans for houses, to create models of cars or motorcycles, and to design machinery. When these designs are finalized, they can later be used to build the actual physical parts, machines, and assemblies. Advancements in CAD motion simulation would offer mechanical engineers more intuitive, qualitative and quantitative feedback about components, enabling faster development of mechanisms.

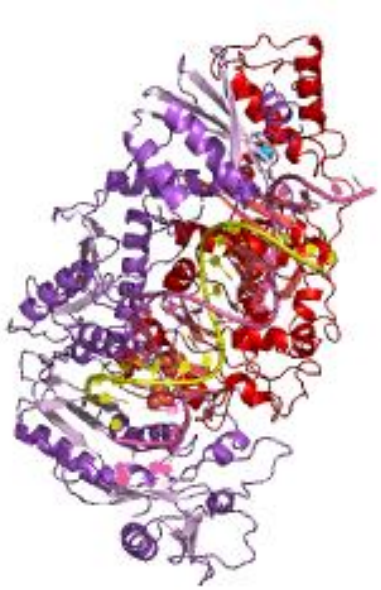


Figure 1.2: HIV-1 protein structure.
Figure reproduced from http://polbase.neb.com/structures/402.

The flexibility of proteins affects how they work as catalysts, muscle building-blocks, and chemical messengers. The three-dimensional shape of

a protein, known as its stereochemistry, has a major influence on its biological activity; see Figure 1.2. A model of protein motion and flexibility would help predict how proteins fold to become active, and how other molecules interact with given proteins, thereby possibly considerably reducing the trial and error researchers must encounter, attempting to design drugs that could ameliorate or cure various diseases.

1.2 Problems

Although sophisticated motion simulation techniques exist, most have computational limitations. For example, FEA (Finite Element Analysis), one of the first simulation tools utilized in CAD software, which reduces a complex problem into finite simpler ones, allows engineers to study the structural performance of their mechanisms. However, the growing complexity of real (and virtual) assemblies necessitates newer techniques for motion simulation. Even the approach of *molecular dynamics*, which is widely recognized as the most accurate method for simulating the motion of proteins, is computationally very expensive and, in some cases, it requires prohibitive computing power and resources.

Real time motion simulation is currently hampered by the computational limitations of the customary techniques. The key step of many

motion simulation techniques is an integration process, used to calculate the new position of the structures or entities, after they have been subjected to a force. Fortuitously, this step can be approximated, substantially reducing its computational expense. However, a new problem arises: how do you resolve the constraints of a structure after a force breaks them? This problem will be referred throughout this thesis as the *constraint satisfaction problem.*

A current research question that naturally arises, which is treated by this thesis, is how to evaluate techniques designed to attempt to solve the constraint resolution problem. Questions that need to be answered include: how does one method compare to another, in terms of speed, efficiency, and perceived quality? Is one method more effective than all others, and if so, to what extent? The problem of determining a quantitative and qualitative method for comparison and evaluate of different motions simulation techniques will be referred to throughout the rest of this thesis as the *performance evaluation problem.*

In order for researchers to incorporate motion simulation tools and related software into their studies, not only does the simulation need to be done in real time or faster, but the tool needs to be easy to use and needs to supply informative feedback. There should be an interactive, intuitive interface that allows users to gain insights about the structures they are

studying.

Our primary research question is to determine if the goal can be realized: can motion simulation be made more computationally tractable by using a modified form of Ragdoll physics, the current method, which incurs low computational cost. The questions remains as to what relative motion simulation qualities and benchmark timings are produced by using Ragdoll physics versus our enhanced techniques.

1.3 Related Work

Motion simulation is used within many different domains including, but not limited to, molecules such as proteins, computer aided design, robotics, and video games. There is a rich history of research throughout each domain, and many techniques have been developed. Ragdoll physics animates characters in video games, molecular dynamics is most commonly used for molecular motion simulation, finite element analysis was the first simulation tool used by CAD software, and kinematics is prevalent in robotics.

1.3.1 Video Games

Visually-realistic gameplay, encompassing the characters' appearances, motions, and interactions, has had a significant influence on the popularity

of video games. A common goal for video game designers is to create an accurate, immersive, and fun virtual world that characters can interact with, through the user. In this section we will describe some of the current research involved with realistic simulation in video games.

Ragdoll Physics

A current internal methodology in video games is known as *Ragdoll physics*, which is used to animate a player's death or fall and to produce realistic interactions within the video game's environment. Ragdoll physics enables video game developers to simulate the laws of physics by programming so-called physics engines using mathematical formulas. Ragdoll physics employs a common motion simulation approach, *break-resolve-iterate*, as shown in Figure 1.3. We use Ragdoll physics as our benchmark because, as demonstrated in sophisticated video-gaming displays, it is a clearly effective and computationally "cheap" methodology, and may well be a reasonable candidate for optimizations needed for more demanding applications and structures, such as in biology. Details will follow in the next chapter.

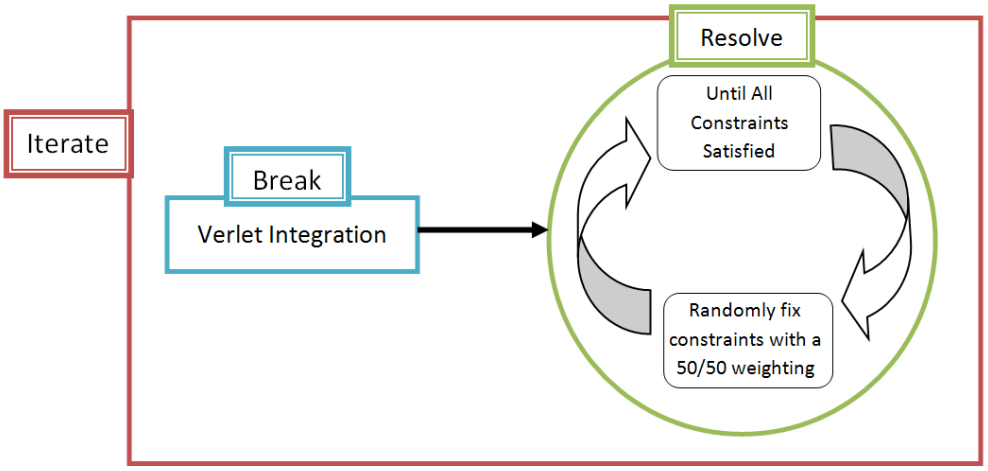


Figure 1.3: Ragdoll Physics Motion Simulation Approach.

Fluid Simulation in Video Games

Although physical simulations in video games have become more realistic, "pervasive simulations of continuous media" are not very common. For example, a cloth, or thread, or soft bodies are not usually simulated. (See figure 1.4.)

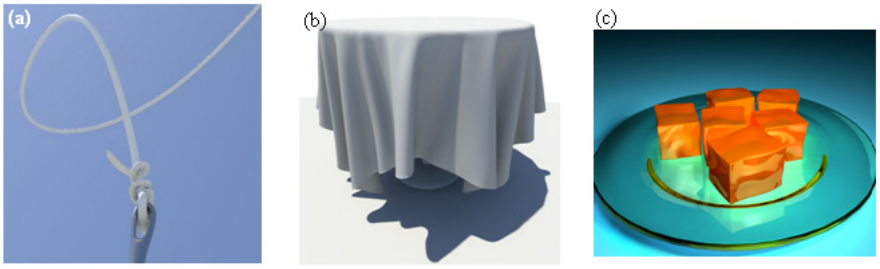


Figure 1.4: Deformable bodies: (a) thread, (b) cloth, and (c) soft bodies. Figure reproduced from [10].

According to [10] a *fluid* is any substance that flows (or takes the shape of its container) and does not resist deformation (it can slide when

dragged). Simulation of fluids involves fluid dynamics which entails large, complex computations. Since fluids have more degrees of freedom and non-linear motion, their equations involve non-linear partial differential equations along with initial and boundary constraints.

According to [10], the key differences between reality and simulation are *approximation and discretization*, and thus they are approximating the fluid dynamics equations to the extent that the simulations look acceptably real. Since the simulations are for video games, their accuracy is not as demanding as those of molecular simulations.

1.3.2 Molecular Simulations

Molecular simulation involves producing bio-chemically feasible motions and is used to understand protein folding and flexibility, enzyme catalysis, DNA, conformation changes. Improvements to molecular simulations could help predict how molecules will interact, which would aid researchers when designing new drugs to cure various diseases. In this section we will discuss current molecular simulation research.

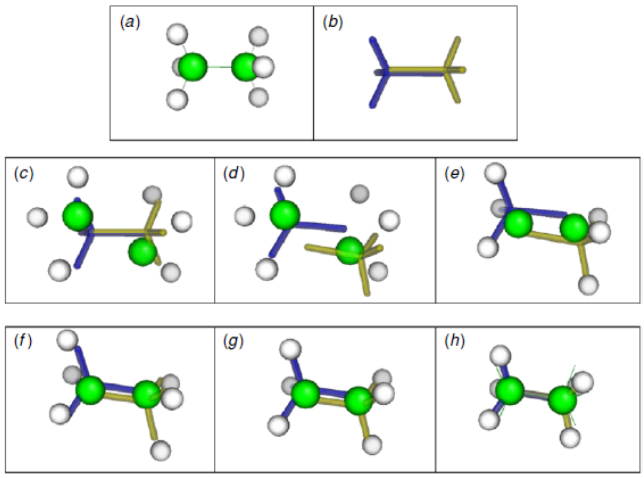


Figure 1.5: The motion of an ethane molecule determined by FRODA. (b) is the ghost template. Figure reproduced from [24].

FRODA

FRODA (framework rigidity optimized dynamic algorithm) was developed at Arizona State University and is a computational method to explore the flexibility of proteins. The algorithm first determines the rigid regions in the protein, which are replaced by so-called *ghost templates*, then random perturbations are applied so the available conformation phase space of a protein can be explored. Ghost templates are virtual rigid bodies and are used by FRODA to represent the potential energies of the atoms. FRODA can find the conformational space of a 100 residue protein in about 10-100 minutes of computing time, depending on how complexly the 100 amino acids are stereochemically conformed, using a computer containing only a single processor. Of course, the elapsed determination time depends sig-

nificantly on the throughput of the computer being used. They state that their algorithm is O(N), linear [24].

FRODA uses a break-resolve-iterate approach, similar to that of Ragdoll physics.(See Figure1.6.) Instead of Verlet integration, FRODA applies random displacements to the atoms, then resolves the structure by iteratively fitting the templates to atomic positions to a certain error tolerance [24].

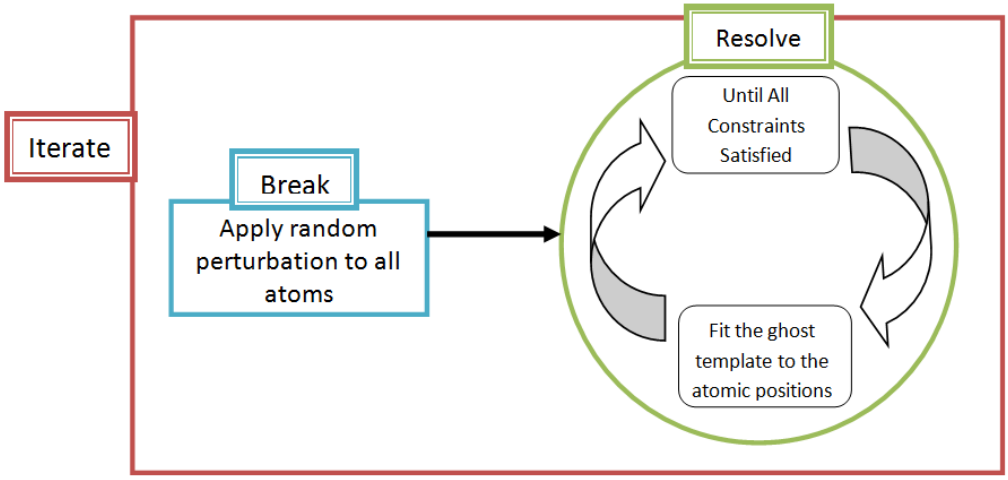


Figure 1.6: The break-resolve-iterate approach of FRODA.

IMD:Interactive Molecular Dynamics

It is hoped that our optimized motion simulation methods may be extended to model the motions and interactions of macromolecules. Stone et al. of the University of Illinois Urbana designed and built a system called Interactive Molecular Dynamics (IMD) which combines a software

tool called Visual Molecular Dynamics (VMD) with a molecular dynamics program (NAMD) and a haptic device to show molecular dynamics simulations [20].

IMD is an improvement of a previous system called Steered Molecular Dynamics (SMD) which used springs. A limitation of SMD that IMD has alleviated is that SMD had to set certain constants before the simulation could begin, and also it ran in batch mode at large supercomputer facilities. IMD takes advantage of the newer, faster PCs and personal supercomputer workstations and allows the tool to run as an interactive system, making it more effective; users of IMD can more easily and quickly try a myriad of experimental cases [20].

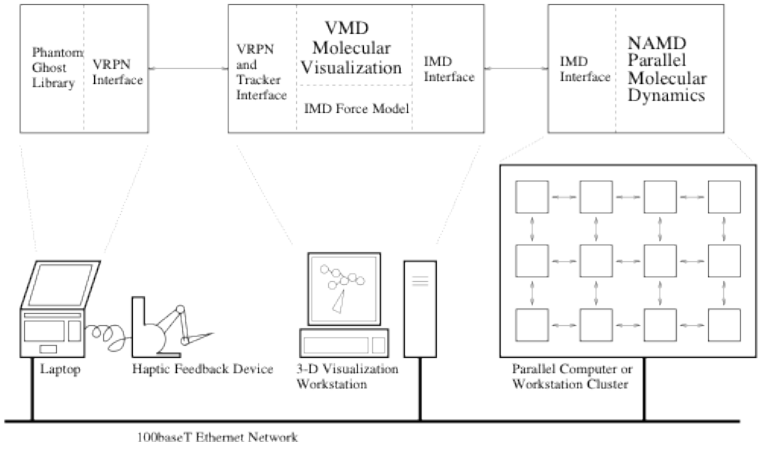


Figure 1.7: Decomposition of IMD components into asynchronous communicating processes. Figure reproduced from [20].

The IMD system was built such that each component runs on a separate machine and therefore must communicate, requiring an efficient

network setup. Although the components need not reside on separate machines, IMD results are based on such a setup. While IMD extends molecular dynamics, it also inherits its computational expensiveness. Even when force feedback communication through a haptic device (an advanced force-feedback joystick) permits intuitive 3D interaction with the tool, IMD still is limited by its computational bottleneck.

1.3.3 Simulation in CAD Software

Mechanical engineers using CAD software need a way to determine the kinematics, the dynamics, and the structural performance of the mechanisms they are designing. Accurate simluations can replace expensive and time-consuming prototype creation. In this section, we will discuss current simulations used in CAD software.

Finite Element Analysis (FEA)

Finite Element Analysis (FEA) was one of the first simulation tools adopted by computer aided design (CAD) software, dating back to the 1980s when CAD became more widely used by engineers, which simulates structural performance of mechanisms. In SolidWorks, a CAD program, FEA is combined with another simulation tool to offer mechanical engineers more quantitative feedback about the components of their mecha-

nisms which extends to faster development [4].

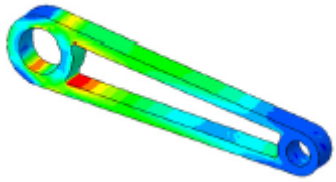


Figure 1.8: Finite Element Analysis done on a connecting rod. Figure reproduced from [2].

FEA uses a numerical analysis method called finite element method (FEM) which basically breaks a complex problems into smaller, simpler problems, called the finite elements. FEM turns the model into a mesh which has certain material and structural properties defining how it will react to certain conditions. It uses mathematical techniques to find solutions to the partial differential equations governing the model by approximating them with ordinary differential equations and numerically integrating. This method still cannot avoid computational expense. [4].

1.4 SolidWorks Simulation

SolidWorks, a common CAD software tool, has several methods of simulation throughout. One mode is the assembly animation that shows the relative motion of assembly components, however, this mode considers speed and timing irrelevant. For a designer to find velocities, accelerations, power

requirements and other quantitative feedback he must use a motion simulation tool. See 1.9 for an illustration of motion in an elliptical trammel [2].

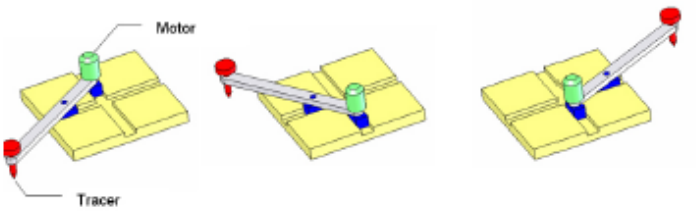


Figure 1.9: Motions of an elliptical trammel simulated with a CAD simulator. Figure reproduced from [2].

SolidWorks states that they have a method of providing complete information about the kinematics and the dynamics of all components of a moving mechanism. Their simulation tool uses material properties from the CAD parts to define the inertial properties and translates the mating conditions from the CAD assembly to kinematic joints. The problem of recognizing kinematic joints in geometric constraint systems introduces another research problem described in [16]. SolidWorks models their mechanisms as assemblies of rigid components with few degrees of freedom, and states that quantitative information can be realized almost instantly. However, SolidWorks is commercial software, and thus their techniques are proprietary [2].

1.4.1 Robotics

Humans and other animals utilize proprioception and other senses to know where and how their bodies are oriented, however, robotic movement in the 3D world depends on spatial perception and knowledge of the degrees of freedom for various robotic parts. (See Figure 1.10.) When a person grabs a fork and uses it to pick up a piece of food on a plate to bring it to their mouth to eat it, the person has no thought about the position of their arm, its degrees of freedom, and the angles through which they must move their joints in order to grab the fork. However, those are necessary questions which need to be answered when dealing with a robotic arm. In this section, we will describe current technique for robotic simulation.

Kinematics

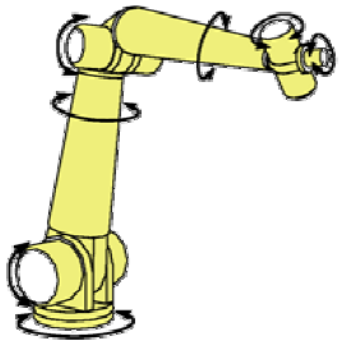


Figure 1.10: A 7R robot with arrows showing the degrees of freedom. Figure reproduced from [17].

One of the ways robotics deals with motion is via kinematics (the study of motion, independent of causative forces). Two types of kinematics that are involved are forward kinematics and inverse kinematics. Forward kinematics finds the position and orientation of any point from the angle of the joints and the length of the links. Inverse kinematics finds the required angle of each joint which produces the desired end position with the desired length of the links [17].

Basically, forward kinematics solves the problem of: "Given all of the joint angles, what is the position of the robot's arm?" and inverse kinematics solves the problem of: "Given the desired position and orientation of the robot arm, what must the joint angles be?" [17].

Kinematics involves many systems of equations and is a computationally expensive method particulary for solving large structures.

Geometric Methods In Robotics

Currently, a research group at the University of Catalonia is working to develop efficient algorithms for solving kinematic constraints. They are using methods from distance geometry to translate the number of kinematic constraints to the number of distance constraints among a set of points. The problem is, given a set of n points in 3-space, and a set of specified distances between them, how to compute all spacial realizations

of the point set that will satisfy all set distances [17].

The group is expressing polynomials as Cayley-Menger determinants which only depend on unknown distances. Their algorithm takes advantage of the fact that the Cayley-Menger equations are multilinear, and has been able to solve problems in Robotics (see figure 1.11), CAD, and structural biology [17].

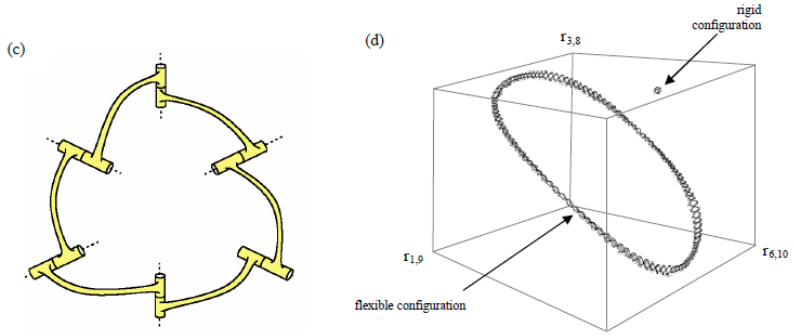


Figure 1.11: Inverse Kinematics of an overconstrained 6R robot. Figure reproduced from [17].

1.5 Thesis Structure

The subsequent chapters explain the theory behind the motion simulation and the techniques that have been developed, tested, and analyzed. Chapter 2 introduces graph theory, geometric constraints structures, Ragdoll physics, and our motion simulation approach. Chapter 3 builds on the

basics from chapter 2, giving both an overview of our technique in general and going into detail on the specific classes of structures we study. and the techniques developed specifically for each class. Chapter 4 will describe the software toolkit for evaluation of motion simulation techniques and chapter 5 will describe in detail, the techniques that we developed, and their performance as analyzed from the results. Finally, the thesis will end with chapter 6 where we state our conclusions along with future work and directions for the research.

Chapter 2

Preliminaries

Proteins (or other molecules), CAD designs, and robotics arms can all be modeled by geometric constraint structures. By representing them abstractly, we are making their complexities transparent, allowing techniques from other domains to be utilized. We begin by defining one of the most basic combinatorial objects, the graph, and relating it to geometric constraint structures and Ragdoll physics. We then give the details of the Verlet integration step used in Ragdoll physics and an overview of the general motion simulation process we have developed.

2.1 Definitions

In this section we will introduce standard definitions relating to graph theory, stylistically following the presentation in [9].

2.1.1 Graph Theory

A *graph* G = (V, E) is a combinatorial object consisting of a *vertex set* V with |V| = n, and an *edge set* E with |E| = m, where E is a collection of unordered pairs of vertices. A *directed graph* is a graph, G = (V, E) where the edges have a direction associated with them, that is E is a set of ordered pairs of vertices from G.

A vertex v is *incident* with an edge e if and only if e={v,w} for some vertex w. A *walk* is an alternating sequence of vertices and edges that begins and ends with the same vertex, in which each vertex, with the exception of the last, is incident with the edge that follows and the last vertex is incident with the preceding edge. A walk is *closed* if the first vertex is the same as the last, otherwise the walk is said to be *open*.

A *path* is an (open) walk (all the vertices are *distinct* [none in common]), whereas a *trail* is a walk with all distinct edges. A circuit is simply a closed trail.

A graph, G, is *connected* if there is a walk between any two vertices in G. A *cycle* in a graph is a circuit where the first vertex appears exactly twice, and there is no other vertex that appears more once. [9] This is not to be confused with a *cycle graph*, C_n which is a graph that entails a single cycle of n vertices. See Figure 2.1.

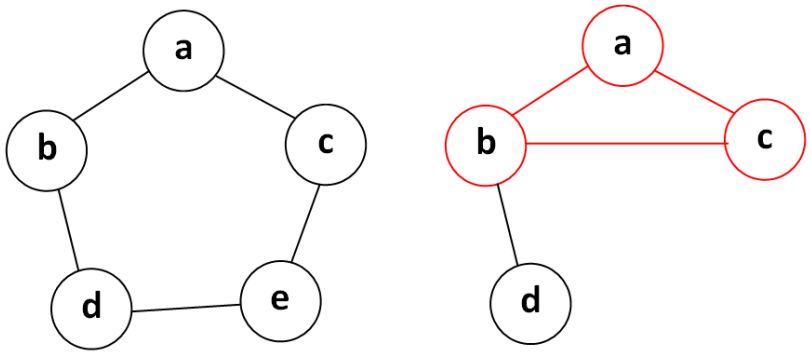


Figure 2.1: A cycle graph C_5, left vs. a cycle in a graph (red) right.

A *subgraph* of a graph, G is a graph whose vertices and edges are subsets of those of G. A *tree* is a connected graph that does not contain any circuits, where as a *spanning tree* of a graph, G, is a subgraph which is a tree and is comprised of all the vertices of G.

2.1.2 Geometric Constraint Structure

A *geometric constraint structure* is a structure that contains geometric entities upon which *constraints* are placed. Constraints restrict entities to satisfy a geometric relationship, such as two lines remaining perpendicular, or two points a fixed distance apart.

A fundamental type of geometric constraint structure is the *bar-and-joint*. Bar-and-joint structures have joints serving as points, and bars representing constraints between them as fixed-length distances. The bars

restrict the joints, defining the structure's degrees of freedom and motions. A graph G = (V, E) along with a distance function on the edges L : E $\rightarrow$ $\mathbb{R}$, defines the framework (G, L) which describes a bar-and-joint structure.

2.2 Search Algorithms

Two of the most basic and commonly used graph search algorithms are Breadth-First Search (BFS) and Depth-First Search (DFS). Both are linear-time algorithms that traverse a graph, G, from a root node, s, and output a spanning tree of G of reachable nodes from s. BFS explores nodes layer by layer, while DFS explores nodes going as deeply as possible, then retreats, or backtracks. See Figure 2.2 for a visual interpretation and Appendix B for pseudo-code [8].

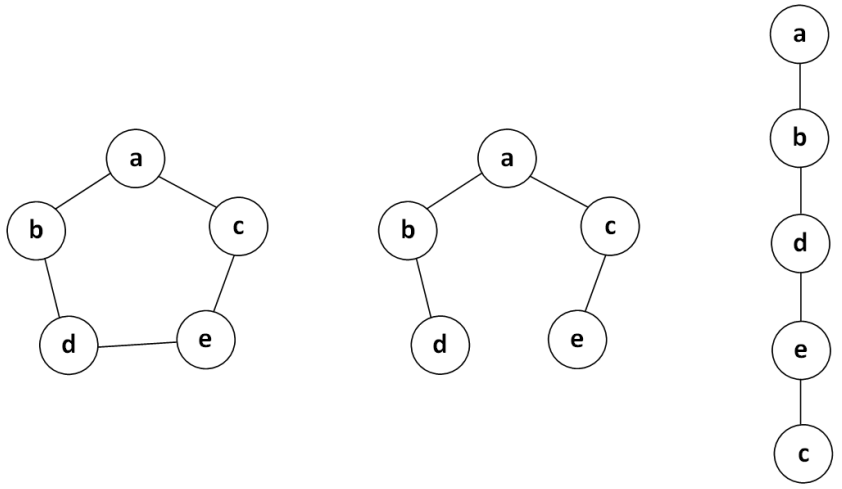


Figure 2.2: A cycle graph C_5 (left), BFS spanning tree from root a, DFS spanning tree from a (right). BFS order produced:{(a,b),(a,c),(b,d),(c,e)}. DFS order produced: {(a,b),(b,d),(d,e),(e,c)}.

2.3 Ragdoll Physics

As mentioned earlier, Ragdoll physics, used to animate a player's death or fall and to produce realistic interactions within a video game's environment, employs the break-resolve-iterate approach of motion simulation. Verlet integration determines the effect of the forces on the structure and calculates new positions of the affected joints, and then a constraint satisfaction technique involving a random ordering and a .5/.5 weighting *resolves* the structure. *Iteration* occurs as forces are repeatedly applied.

2.3.1 Verlet Integration

Verlet integration stems from basic physics formulas, but throttles them by holding the acceleration and the timestep constant. The velocity is inferred by calculating the difference between the current and previous position of the particle or body. This approximation simplifies the computational approach to motion, while remaining fairly accurate and realistic. Basically, Verlet integration allows greater stability and fewer computations compared to other approaches [12].

Another important step in Ragdoll physics is collision handling. Since we are not handling collisions, we refer the reader to a seminal article on character physics [12], which elucidates how to handle collisions.

2.3.2 Constraint Satisfaction

After Verlet integration moves affected joints, it is likely that constraints have been violated. At this stage, Ragdoll physics examines the structure's constraints in a random *order*, resolving them with a .5/.5 *weighting*. See Figure 2.3.

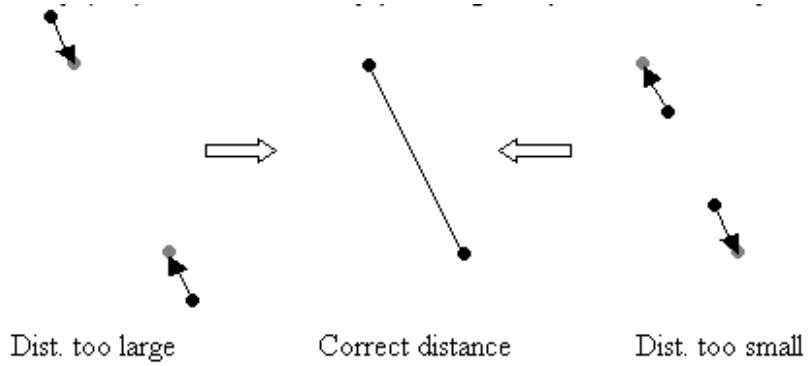


Figure 2.3: Fixing a broken constraint with .5/.5 weighting. Figure reproduced from [12].

During the examination of a constraint, the error, if any, is calculated as the difference between the length of the constraint and the distance between the joints in the constraint. A negative error implies the joints need to move closer together, inversely, a positive error indicates the joints need to move further apart. See Figure 2.3.

The weighting ratio refers to the percent of the constraint distance error that each joint must resolve. A .5/.5 weighting implies each joint

involved in the constraint moves 50% of the error. See Figure 2.4 for pseudo-code.

```
delta = x2-x1;
deltalength = sqrt(delta*delta);
diff = (deltalength-restlength)/deltalength;
x1 += delta*0.5*diff;
x2 -= delta*0.5*diff;
```

Figure 2.4: 1D pseudo-code for satisfying the constraint between x1 and x2. Figure reproduced from [12].

2.3.3 Implementation Considerations

Ragdoll physics is based on numerical calculations, particularly during the constraint satisfaction stage, thus requiring certain parameters to be set before execution, such as error tolerance and maximum constraint iterations. An error tolerance states an acceptable error for a constraint to be proclaimed satisfied, since basing it on perfection, error of 0, is unrealistic and possibly impossible. An upper bound must also be set to stop an unsolvable break in a constraint to cause infinite looping. These parameters have a large impact on the efficiency and accuracy of Ragdoll physics.

Chapter 3

Methodologies

Graphs can be of many different types, sizes, and shapes, but to simplify the issue of real time motion simulation we will be extending the break, resolve, iterate (BRI) approach of Ragdoll physics, and focusing in on smaller specific classifications of graphs. We are working to find the best techniques for each specific structure however, formal mathematical analysis is a research problem of its own. Thus we have developed a software tool for evaluation and comparison of our techniques to the current methods.

3.1 Our Approach

Rather than relying on current, computationally expensive techniques, such as molecular dynamics, finite element analysis, and kinematics, we are elucidating the intricacies of structural motion by using geometric constraints to model the structural interactions. The geometric constraint

structure that we focus on is the 3D bar-and-joint structure which reduces the complexity of the structures while also supplying an expandable, universal format. A 3D bar-and-joint complex can model many different CAD constraints, molecular structures, and robotic body parts; see Figures 3.1 and 3.2. This level of abstraction facilitates the development of more efficient motion simulation techniques, by incorporating methodologies from other domains, such as the video game industry.

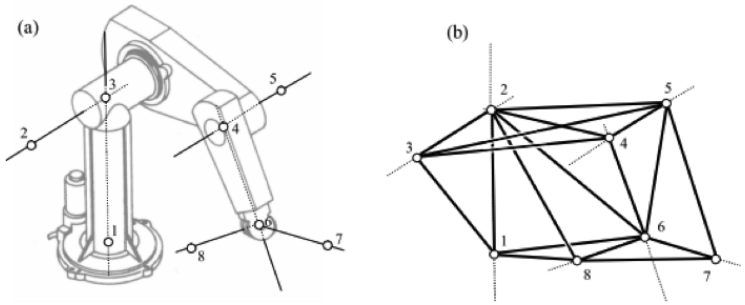


Figure 3.1: (a) A PUMA robotic arm (b) it's associated bar-and-joint framework. Figure reproduced from [17].

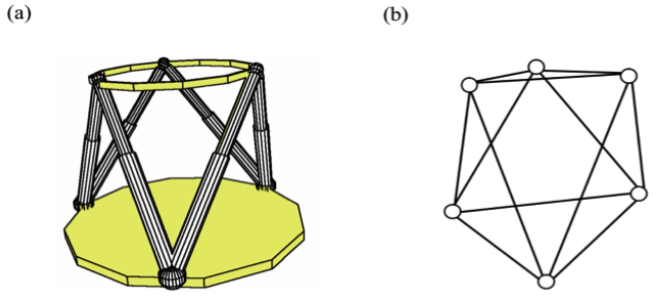


Figure 3.2: (a) An octahedral Stewart-Gough platform (b) it's associated bar-and-joint framework. Figure reproduced from [17].

Our approach for improving motion simulation is to enhance Ragdoll physics by adjusting the *ordering* and the *weighting* of the constraints. (See Figure 3.3.) Where Ragdoll physics strictly uses a .5/.5 weighting, we

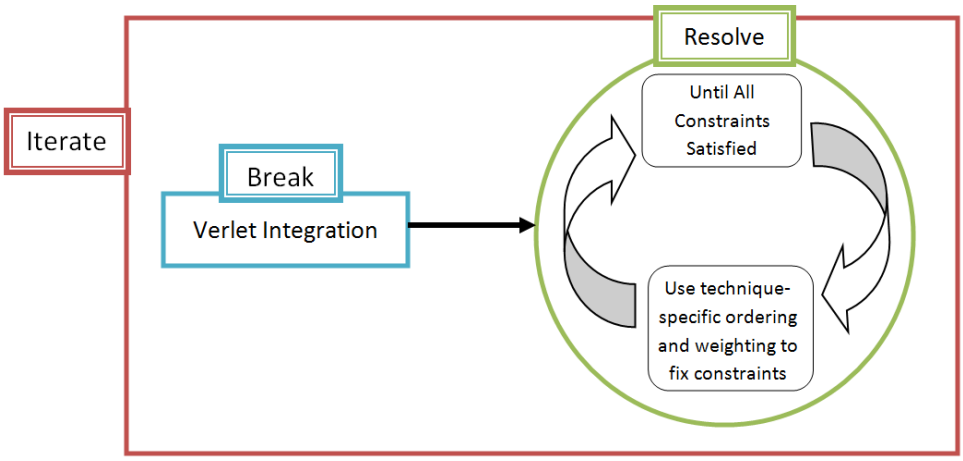


Figure 3.3: An overview of the generalized Ragdoll physics motion simulation approach.

are exploring weighting techniques including 1/0, .75/.25, and also .5/.5 and where Ragdoll physics strictly uses a random ordering, we are exploiting BFS and DFS orderings.

To begin we first classify a specific set of structures for evaluation. Then we enumerate a large random selection of structures, develop constraint satisfaction algorithms, and evaluate each technique on each class of structures. The yardstick by which the techniques are measured is the number of constraint iterations required to solve the each structure. An assumption is that individual iterations take the same amount of time, since each iteration processes every constraint in the structure. See Figure 3.4 for a diagram illustrating our approach.

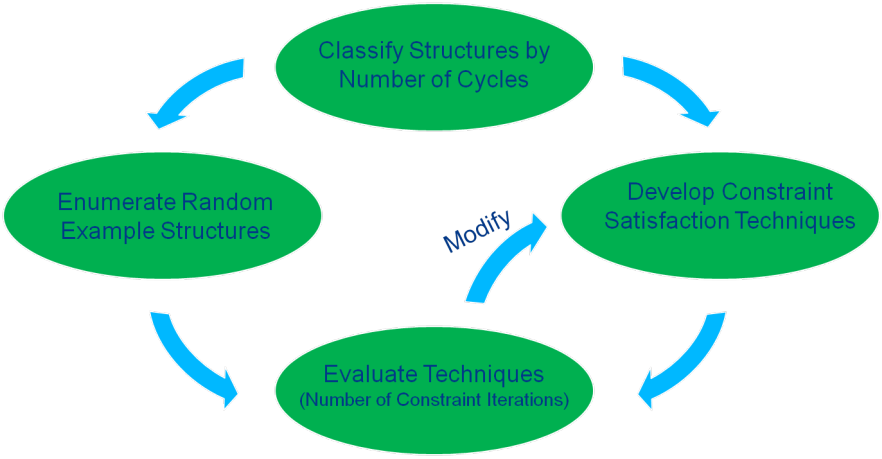


Figure 3.4: An overview of our approach.

3.2 Structure Classification

The first step in our approach is to categorize different classes of structures. In this section we will explain the different classes of graphs that we will be focusing on, and give an example of each.

For the purposes of this thesis, we redefine a graph to have an enhancement: that is, a graph G=(V,E,C) is a combinatorial object consisting of a vertex set V, with |V| = n, an edge set, E, a collection of unordered pairs of vertices, with |E| = m, and an additional parameter, a cycle set, C, a collection of sets of ordered edges that create a cycle, with |C|=k, the number of cycles in the graph.

3.2.1 Tree

The first type of graph that we have chosen to focus on is the tree. A tree, T, is any undirected, connected graph such that n=m+1, and contains no cycles, k=0. There is a path between every pair of vertices in a tree. It is important to note that if one edge is added to T, connecting two existing vertices, then T will contain a cycle and no longer be a tree. See Figure 3.5 for an example.

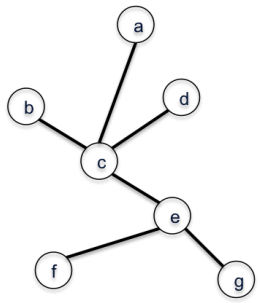


Figure 3.5: An example of a tree graph, T.

The graph T=(V,E,C) in Figure 3.5 contains a vertex set V={a,b,c,d,e,f,g}, an edge set E={(a,c),(b,c),(c,d),(c,e),(e,f),(e,g)}, and a cycle set C=$\emptyset$, where |V|=7, |E|=6, and |C|=0.

3.2.2 1-Cycle

The next type of graph that we consider is the 1-cycle graph. We define a 1-cycle graph to be any undirected, connected graph that contains exactly one cycle. That is, if S is a 1-cycle graph, then |C|=1. Every cycle graph, C_n is also a 1-cycle graph. A 1-cycle graph is a tree with one edge added to it, connecting two vertices. Throughout the rest of this thesis, we will call this type of edge a *problem constraint*.

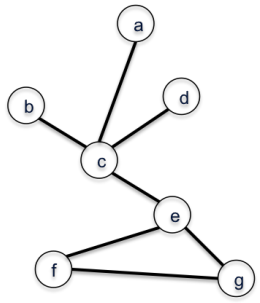


Figure 3.6: An example of a 1-cycle graph, S, created from the tree graph, T.

Starting with T in Figure 3.5, then adding an edge between two vertices, f and g, creates a 1-cycle graph, S. See Figure 3.6 for an example. The graph S=(V,E,C) contains vertex set V={a,b,c,d,e,f,g}, edge set E={(a,c),(b,c),(c,d),(c,e),(e,f),(e,g),(f,g)}, and cycle set C={(e,g,f)}, where |V|=7, |E|=6, and |C|=1.

3.2.3 2-Cycle

The next type of graph that we have chosen to focus on is the 2-cycle graph. We define a 2-cycle to be any undirected, connected graph that contains exactly two cycles. That is, if D is a 2-cycle graph, then |C|=2. A 2-cycle graph is a tree with two edges added to it between two separate pairs of vertices, that is, a 2-cycle has two problem constraints.

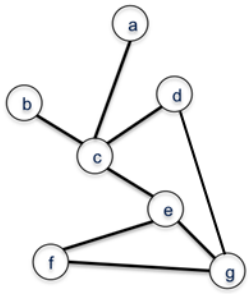


Figure 3.7: An example of a 2-cycle graph, D, created from the 1-cycle graph, S.

Starting with the graph T shown in Figure 3.5, then adding one edge between two vertices, f and g, will give us a 1-cycle, S, shown in Figure 3.6. If we then add an edge between vertices d and g, we will get a 2-cycle, D. This is represented by Figure 3.7. There are three different *types* of 2-cycles. Either two 1-cycles joined by a tree, two 1-cycles joined at one vertex, or a 1-cycle with a path between a pair of its vertices.

3.2.4 n-Cycle

The complexity of cyclic graphs rapidly increases with increasing numbers of cycles, but the concepts and algorithms used in this thesis are general, and can be applied to directly evaluate greater complexities. That is, an n-cycle would be any connected, undirected graph with exactly n cycles. An n-cycle graph is a tree graph with n edges added to it between n pairs of vertices, that is, an n-cycle graph has n problem constraints. However, for this thesis, we will be focusing on trees, 1-cycles, and 2-cycles.

3.3 Structure Enumeration

In order to fully test our motion simulation approach, we randomly generate structures of various shapes and sizes. How to generate completely random structures is another research question in itself. In this section we will describe the direction we have taken to enumerate our test structures.

3.3.1 Tree Graph Generation

To randomly generate tree graphs:

1. Create a root joint, r, at a randomly generated position,(x,y,z)

2. Create a new joint, j, again, at a randomly generated position, and then create a bar connecting r and j.

3. Set the root joint r equal to the new joint j; r=j.

   Steps 2 and 3 together are called createBar-joint-pair

4. While we have not reached the randomly preset numberOfChildren, call createBar-joint-pair. Then if we have not reached the preset treeDepth, make a recursive call.

3.3.2 1-Cycle Graph Generation

To randomly generate 1-cycle graphs we either:

1. Create a random tree and add a bar between a random pair of joints.

2. Create a random number of bar-joint-pairs and then connect the root to the last joint from the bar-joint pairs. Then add random trees to random joints in the cycle.

3.3.3 2-Cycle Graph Generation

To randomly generate 2-cycle graphs, we either:

1. Create a random tree, and add two bars between two separate, random, pairs of joints.

2. Create a 1-cycle graph and then create another 1-cycle graph starting from a random joint in the first 1-cycle graph. Both 1-cycle graphs share a joint.

3. Create two 1-cycle graphs and connect them by a random tree. A tree separates both 1-cycles.

4. Create a 1-cycle graph and create a path between two random vertices in the 1-cycle.

Chapter 4

Software Toolkit

As structures get larger and more complicated, obtaining mathematical proofs that a technique will be optimal becomes a challenging research question of its own. Thus in order to evaluate the proficiency of our constraint satisfaction techniques, we have developed a software tool employing the motion simulation approach shown in Figure 4.1, that allows manipulation of structures and resolution of their constraints through a user selection of ordering and weighting. In this chapter we will describe this software, MotionSim, its design and infrastructure, as well as the output and evaluation methods used.

4.1 MotionSim

MotionSim is a complex software tool, comprised of many separate programs, whose aim is to provide real-time motion simulation of geometric constraint structures through utilization of the constraint satisfaction tech-

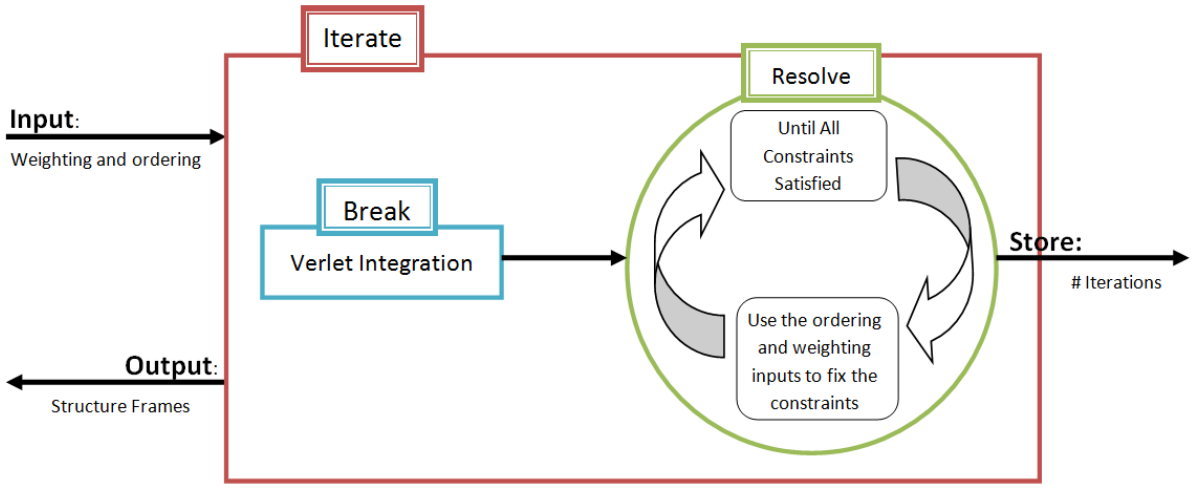


Figure 4.1: The basic architecture of the software tool.

niques we developed, and to allow for quick and easy development of new techniques. Detailed information regarding our techniques will follow in chapter 5, also to see an example of the motionSim process; see Figures 4.2 - 4.5.

In the rest of this section, we will describe the individual components of the software tool, and later in this chapter its features, and implementation details.

4.1.1 Software Components

Structure Generator

One of the main components of MotionSim is the structure generator which outputs random XML representations of trees, 1-cycles, and 2-cycles,

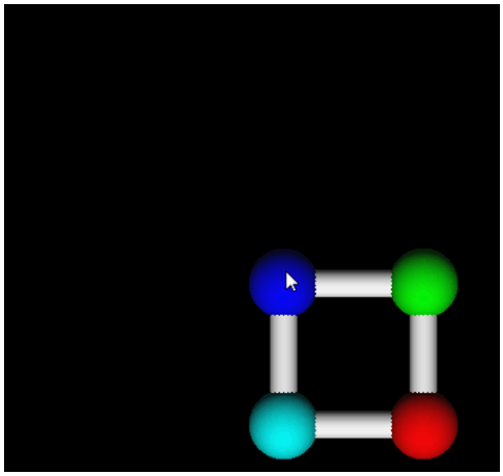


Figure 4.2: Read in the XML graph file and display it on the screen.

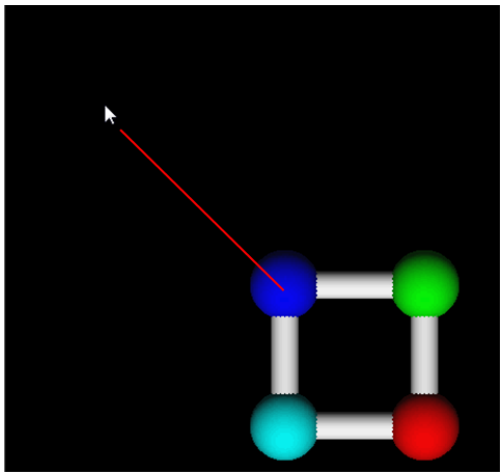


Figure 4.3: Apply a force to a joint in the structure.

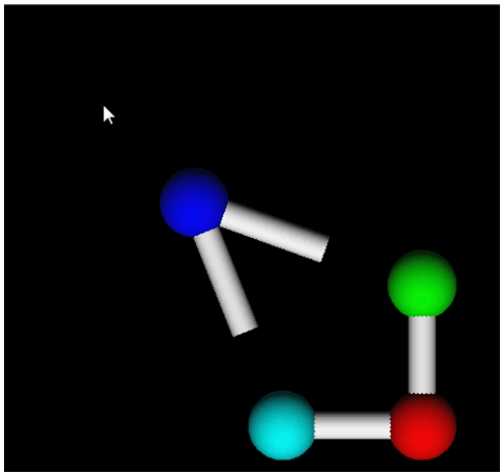


Figure 4.4: Apply Verlet Integration to the affected joint.

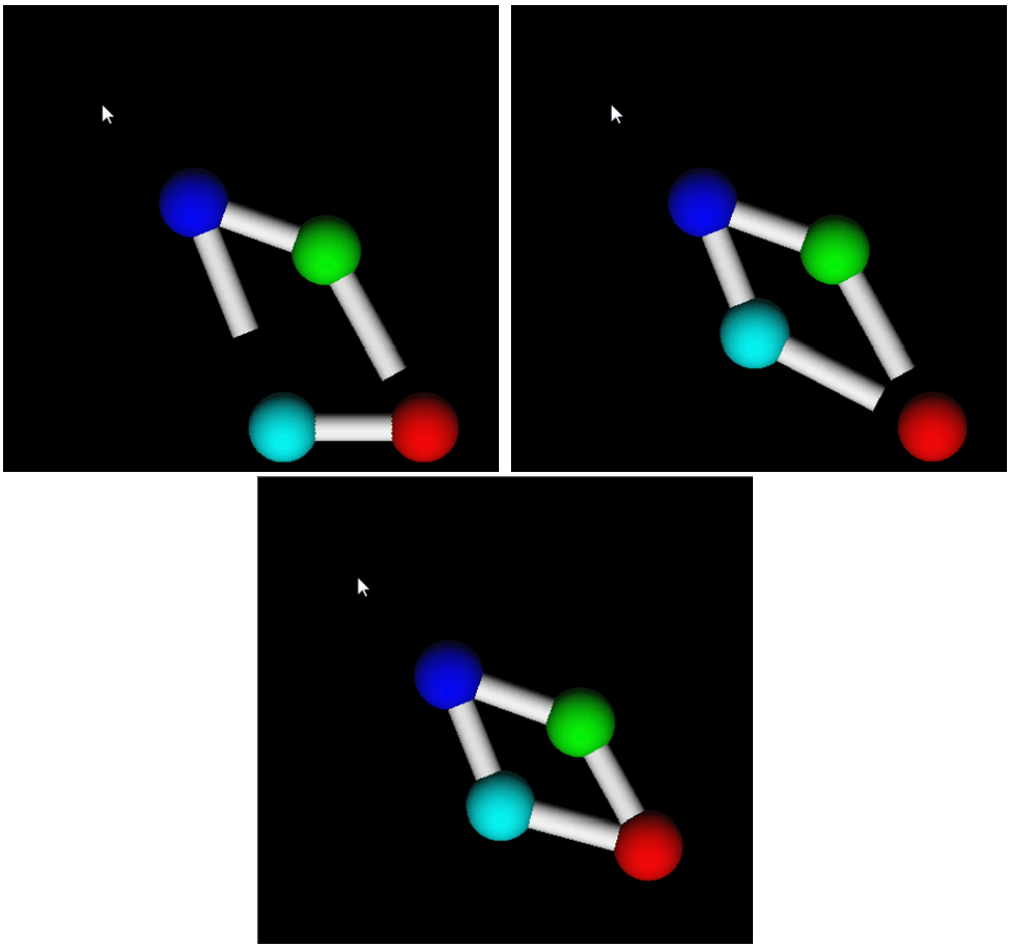


Figure 4.5: Repeatedly apply a constraint resolution technique, counting the number of iterations required before the structure is resolved.

refer to Chapter 3 for the methodology behind the generation. structGen can be used to output a single XML graph file of a specified type, or for testing, it will output a complete set of trees, 1-cycles, and 2-cycles.

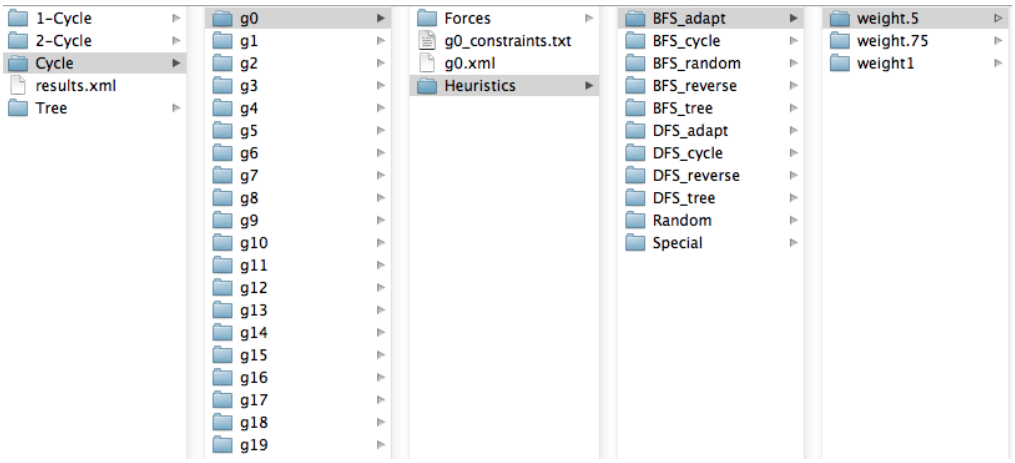


Figure 4.6: The file structure of StructGen's output when in testing mode.

In testing mode, the generation of unique data files, by category, is a main feature of this component. (See Figure 4.6.) Note that folders are created for each type of graph (trees, 1-cycles, and 2-cycles), and within those folders there is a sub-folder created for each graph that is put out. Continuing to drill down the folder hierarchy, each graph's folder contains an XML file that represents the graph, a constraint file declaring the constraint ordering, and a heuristic folder. The heuristic folder has a sub-folder for each heuristic being tested, and a sub-folder of files with random forces to be applied to the graph. Back at the top of the folder hierarchy, there is an XML file, results.xml, which stores a representation of the entire set of generated files. Results.xml is used when running the

test of all the structures so that the output ends up in the correct path.

Force Generator

ForceGen is used to create an input file for simulating random forces. The file contains the (x,y,z) of the forces, the affected joint, as well as the maximum constraint iterations, error tolerance, and timestep. This tool can be used independently to create force files for a single graph, or when in testing mode, structGen will automatically call forceGen for every graph it creates and will output the force files as well.

Constraint Solver

ConstraintSolver contains implementations of all our constraint satisfaction techniques and is used by motionSim to order the constraints of the structure after every force application, during the *resolve* phase.

4.2 Input, Output and Features

In this section we will describe the different ways that the user can interact with the tool through input and output, along with the many features that the tool supports.

4.2.1 Input

When running motionSim, there are many ways to input data each depending on what the user wishes to utilize the tool for. To apply forces by hand, the user must specify a specific XML graph file, generally one generated by StructGen, the constraint file path, the timestep, maximum constraint iterations, error tolerance, and especially the ordering and weighting technique to be used for constraint satisfaction. If instead of applying forces by hand (mouse drag), the user wishes to read in a file of random forces created by forceGen, then it is not necessary to specify a timestep, maximum constraint iterations, or error tolerance since it will be preset in the force file.

Force Files

As briefly mentioned, motionSim can input and output force files. When applying forces to a structure, the forces can be recorded and then could be reused to reapply the same forces. This is useful when comparing the same forces on the same structure, with different constraint satisfaction techniques. The feature enables us to keep our experiments accurate and consistent.

4.2.2 Output

By default, motionSim will output the frames of the structural motion to a file, as well as record the number of constraint iterations required for each force applied. The constraint iteration files are used when calculating averages and for comparing the techniques while the structure files enable the user to plot the motion of the structure that was simulated.

4.2.3 Features

MotionSim has many features such as:

- reading in force files generated by forceGen
- recording user applied forces to a readable force file
- accepting 2D or 3D forces. A typical mouse uses 2D, however, if the tool were connected to a haptic device, as done in other motion simulation systems, then 3D forces (as true vectors) could be applied with minor software changes.
- either turning on or off seeing the breaks and constraint resolution

process as forces are applied to a structure

- zoom in and out of the structure
- change the background color

4.3 Implementation

The Model-View-Controller (MVC) architecture is a popular framework used by software engineers to separate a program into logical sections. The controller is responsible for handling input from the user interface, and converting it to an understandable action for the model. The model section contains information specific to the appearance and behaviors of the model and reacts to input received from the controller. The view is constantly querying the model and updating the graphics accordingly. In short, the controller handles events and sends messages to the model which changes accordingly and is visually updated by the view section.

MotionSim is utilizing a Model-View-Controller architecture, an overview can be seen in Figure 4.7. Since the tasks MotionSim manages are complex, it is required to have several controls and views. The view is co-managed by viewGL and viewFormGL software components; viewGL handles displaying the bar-and-joint structure, while viewFormGL handles displaying any changes the user might make via the GUI; see Figure

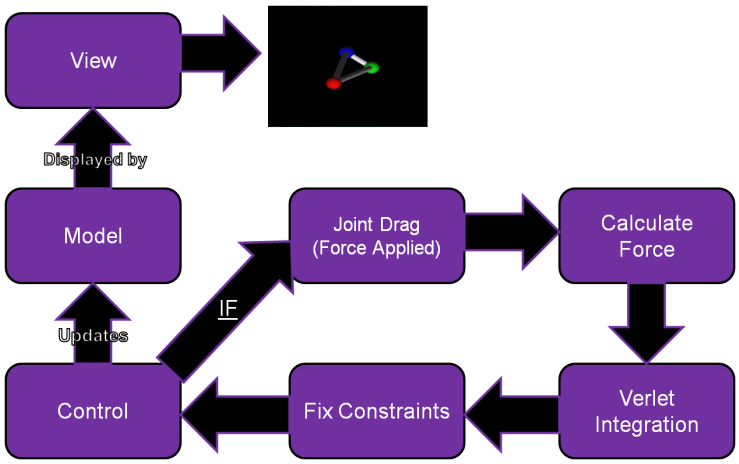


Figure 4.7: The MVC architecture of the software tool.

4.8. There are also several controller classes, specifically, controllerGL, and controllerFormGL. The controllerGL listens for events on the form seen in Figure 4.8 left, where as controllerFormGL listens for events that happen in the openGL window, specifically manipulation of the model; see Figure 4.8 right. With all components working in concert, we are able to present an interactive tool to visualize bar-and-joint structures, and how user-applied forces affect them, including how they distort, and possibly break and repair; see Figure 4.9.

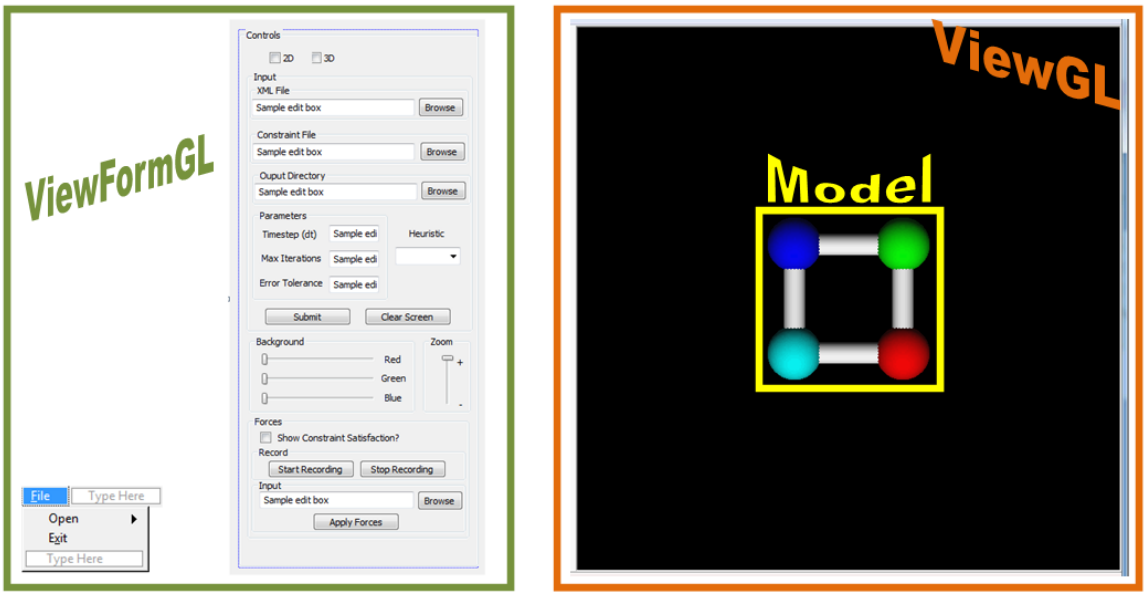


Figure 4.8: The Model-View-Controller separation of MotionSim.

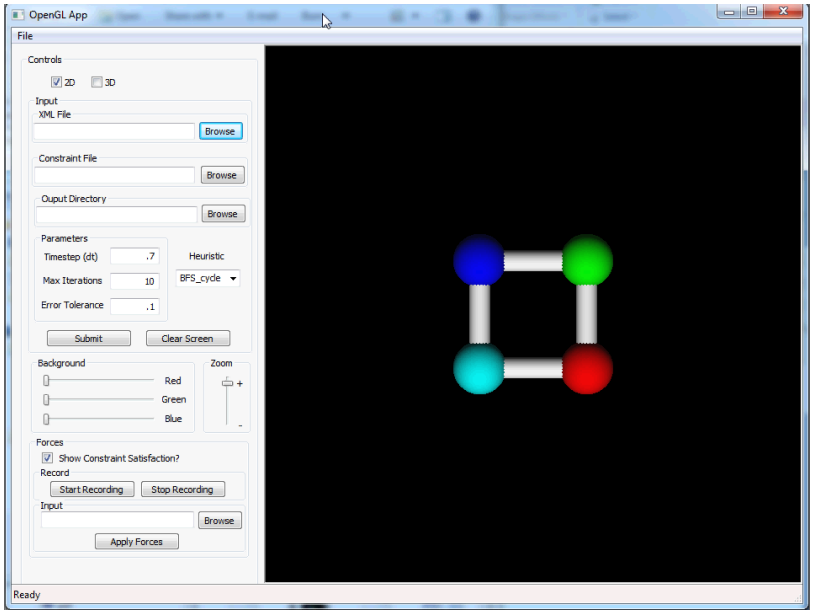


Figure 4.9: The graphical user interface that the user interacts with.

Chapter 5

Techniques, Results and Analysis

In this chapter, we present various techniques developed for constraint satisfaction. All the techniques have been generalized from the Ragdoll physics method, and are implemented with each of the three different weighting schemes, .5/.5, .75/.25, and 1/0. Throughout the rest of this chapter the *root joint* will refer to the force-affected joint, which is the joint where all our orderings begin.

Note that evaluation of our techniques is based on the number of iterations required to resolve the structure, and that all comparisons will be made to the results of Ragdoll physics, which employs random ordering and .5/.5 weighting.

Our data is based on 20 graph files of each structure type, generated randomly by structGen. Each structure has had 50 random forces applied to it and the number of constraint iterations required to resolve the struc-

ture has been recorded and plotted for each technique. To ensure reliable comparisons we evaluate each technique on the same structure with the same random forces.

We will begin each section with a brief overview of the technique's approach, followed by a figure illustrating the ordering, and ending with a discussion and analysis of the results. The reader should note that the techniques we developed are a product of intuition we gained from initial testing of base techniques on simple structures.

5.1 Techniques for Tree Structures

In this section we address the constraint satisfaction problem of tree structures. We first introduce the techniques, we then evaluate the results and, lastly, we compare and contrast the results to those of our own Ragdoll implementation. Our statistical analysis is based on the calculated mean, mode, and standard deviation of the data and histograms for visual representation. Our results discussion will based on the bar-charts in Figure 5.1 for which the raw data can be found in Appendix C.

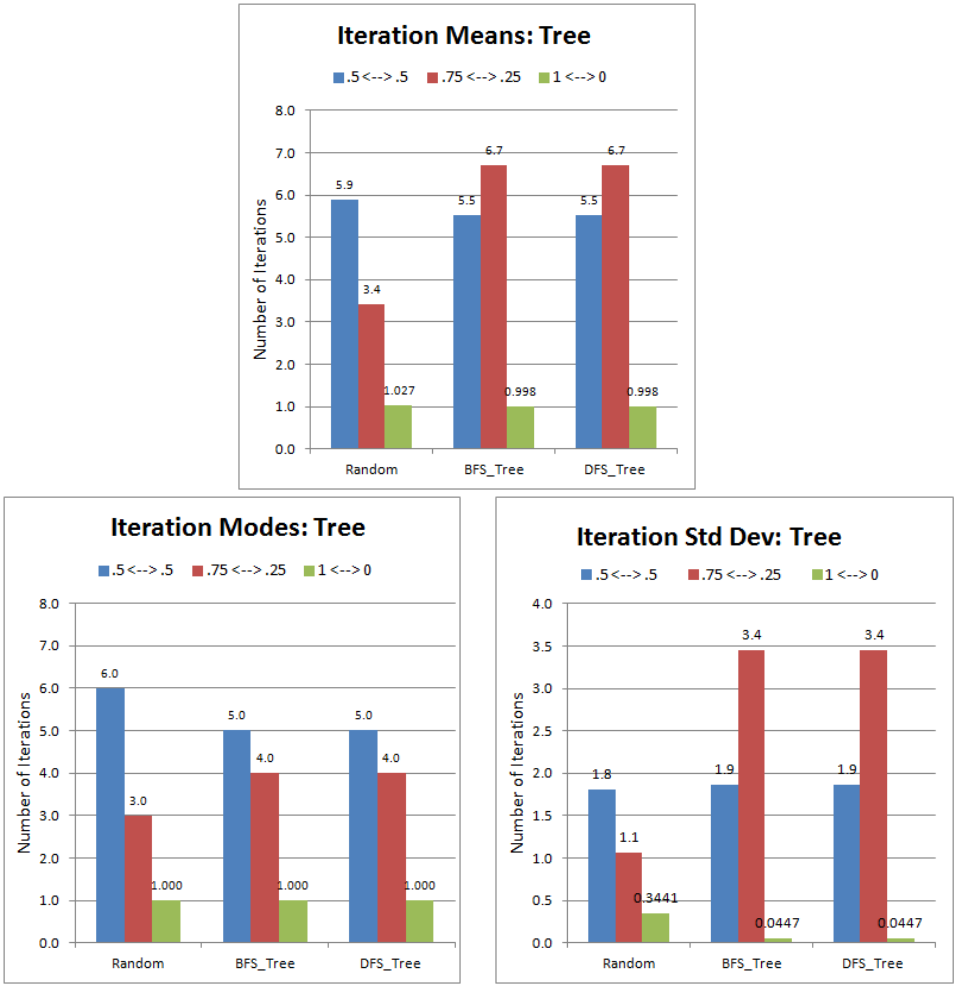


Figure 5.1: BFS_tree and DFS_tree tree graph results.

5.1.1 Tree

Two of the most common ways to search a tree are by using Breadth-First Search and Depth-First Search. BFS searches *layer by layer*, while DFS searches *depth first*. Since BFS and DFS are effective search algorithms, we expect that BFS and DFS will provide an efficient and effective approach for traversing trees, and solving their (broken) constraints. We

developed a 1/0 weighting scheme specifically for tree structures and with BFS or DFS ordering, a tree can always be resolved in 1 iteration. It is straightforward to prove.

Approach. Perform a Breadth-First Search or Depth-First Search from the root joint. Refer back to Figure 2.2 for an example of BFS and DFS.

Results and Discussion. Our results after performing BFS_tree and DFS_tree on tree graphs can be seen in the charts in Figures 5.1 which will be compared to the results of a random ordering.

Looking at the means one can see that BFS_tree and DFS_tree with a 1/0 weighting consistently solve the constraints in one iteration. Ragdoll ordering with a 1/0 weighting is also very close to a mean of 1, however, if we look at the raw data in Appendix C C.3, we can see that it does not always solve the constraints in 1 iteration. Also, looking at the modes and the standard deviations we can see the BFS_tree, and DFS_tree have a standard deviation of 0 and a mode of 1, meaning they can be considered nearly completely consistent techniques, solving the constraints in (essentially) 1 iteration. Random ordering with a 1/0 weighting also has a very low standard deviation and mode.

Random with a .5/.5 weighting is the equivalent of Ragdoll physics. So far we have found 3 techniques that outperform Ragdoll physics for tree graphs. Our results so far show that weighting has a large impact

on the number of constraint iterations. The graphs show that weighting gates ordering; an enhanced traversal-search is dependant on (can only excel in) a facilitative environment. Understanding why this is so may not be entirely intuitive, since it is difficult to determine how deeply into a bar-and-joint structure externally-applied forces may propagate, and therefore, how extensively the structure's constraints are damaged.

5.2 Techniques for 1-cycles and 2-cycles

This section will introduce the more complicated techniques we developed for 1-cycle and 2-cycle graphs. We will explain the ordering approach, followed by an example, and end with a discussion and analysis of the results. As before, we will be comparing our techniques to our Ragdoll implementation.

5.2.1 Cycle

BFS and DFS were very effective for constraint satisfaction of trees, which influenced our choice to try a similar algorithm for graphs containing cycles. However, BFS and DFS produce a spanning tree, which will not contain all the edges of the original graph. We must look at every constraint in a structure, to ensure the entire structure can be satisfied.

Therefore we extended BFS_tree and DFS_tree to add additional edges to the ordering.

Approach. Perform a Breadth-First Search or a Depth-First Search from the root joint, however this time marking both the vertices and the edges. By additionally marking the edges, we can guarantee that all constraints are added to the ordering. See Figure 5.2 for an example. If a vertex is reached that was previously marked, but its edge is not marked, we add the edge to the ordering but assign it a .5/.5 weighting. These edges are defined as *problem constraints*.

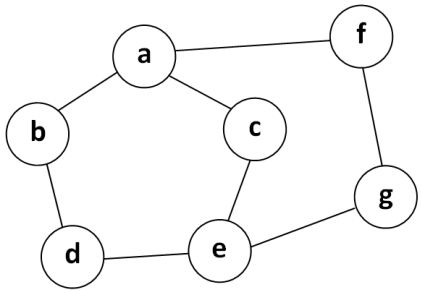


Figure 5.2: An example of a 2-cycle graph. BFS_cycle order produced from root a: {(a,b),(a,c),(a,f),(b,d),(c,e),(f,g),(d,e),(e,g)}. Problem constraints:{(d,e),(e,g)}. DFS_cycle order produced from root a: {(a,b),(b,d),(d,e),(e,c),(c,a),(e,g),(g,f),(f,a)}. Problem constraints:{(c,a),(f,a)}.

Results and Discussion. Our results after performing BFS_cycle and DFS_cycle on 1-cycle and 2-cycle graphs can be seen in the charts in Figures 5.4 and 5.3 which will be compared to the results of a random ordering.

First, looking at the means, observe that (1) BFS_Cycle with a .5/.5 weighting outperforms all other technique and weighting combinations for

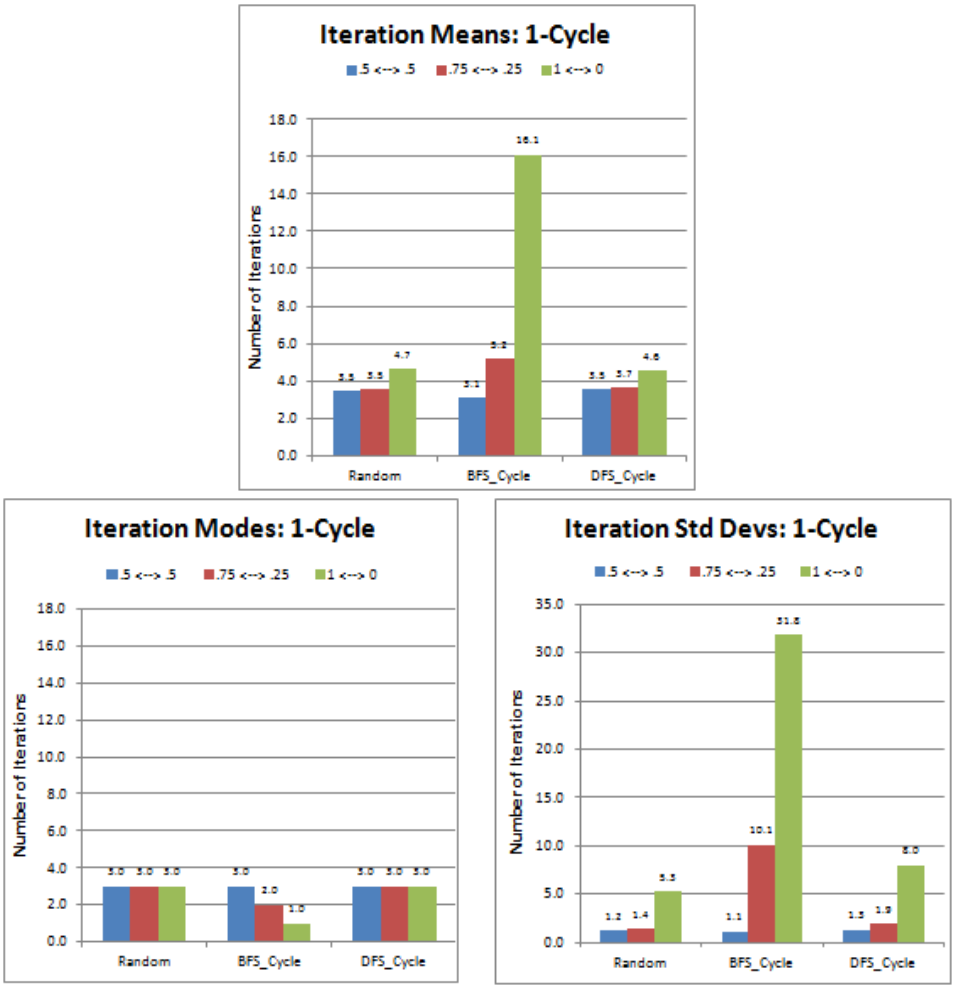


Figure 5.3: BFS_cycle and DFS_cycle 1-cycle graph results.

both 1-cycle graphs and 2-cycle graphs, (2) DFS_Cycle performs best with a .5/.5 weighting for both 1-cycles and 2-cycles, (3) Ragdoll also performs best with a .5/.5 weighting for both 1-cycles and 2-cycles, which is the original Ragdoll physics, (4) BFS_cycle with a 1/0 weighting performs significantly worse than any other weighting, and (5) both BFS_cycle and DFS_cycle with a .5/.5 weighting perform at or better than Ragdoll for 1-cycles and 2-cycles. This chart seems to point out that .5/.5 weighting

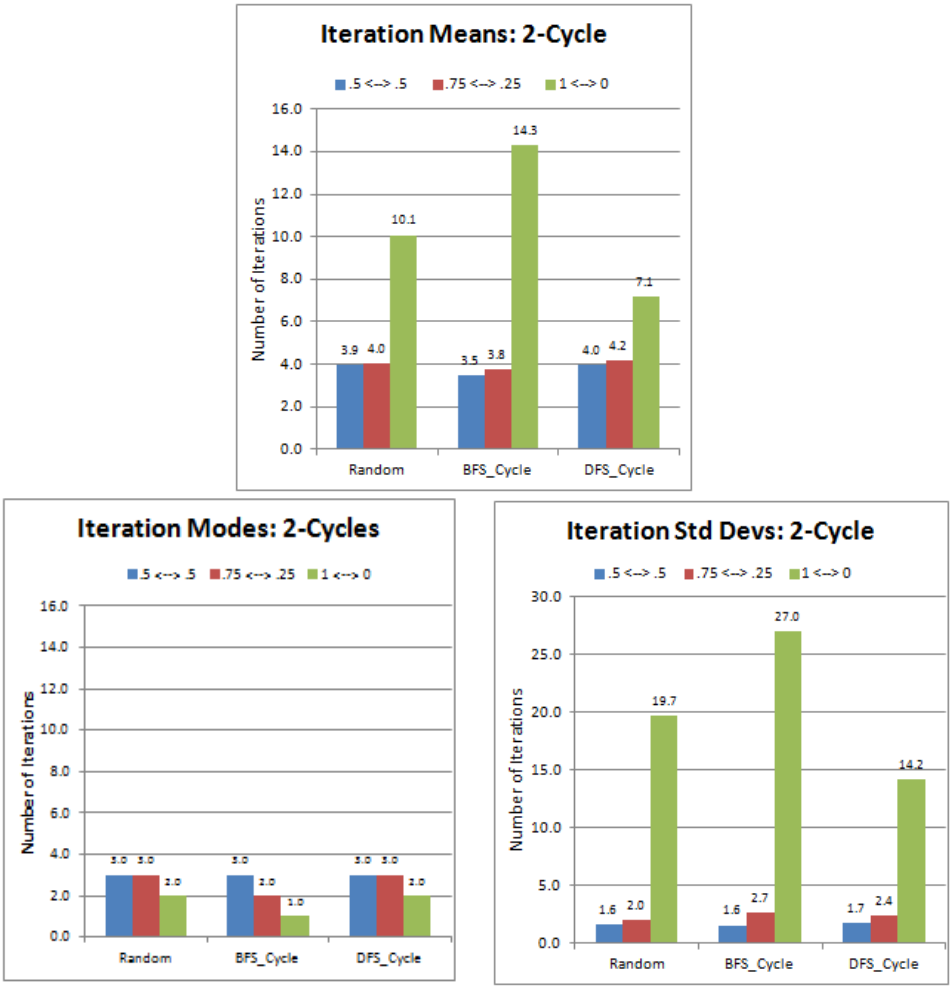


Figure 5.4: BFS_cycle and DFS_cycle 2-cycle graph results.

is better than the other weighting schemes.

Next, focusing on the modes, we can see that (1) DFS_cycle and Ragdoll have an identical mode of 3 for all weightings on 1-cycle graphs, (2) BFS_cycle solves the constraints in the lowest number of iterations most commonly, for all weighting techniques on 1-cycles and 2-cycles, and (3) BFS_cycle most commonly solves the constraint satisfaction problem in 1 iteration, for both 1-cycles and 2-cycles. From the means chart we saw

that BFS_cycle with a 1/0 weighting had a significantly worse mean than all the other techniques, however it most frequently solves the constraints in 1 iteration. This suggests that BFS_cycle with a 1/0 weighting has potential for graphs with cycles.

Finally, examining the standard deviation, we can recognize that (1) BFS_cycle with 1/0 weighting is extremely high, which was suggested by the high mean but low mode, (2) BFS_cycle, DFS_cycle,and Ragdoll with a .5/.5 weighting all have very low, almost indifferent standard deviations, suggesting that they are all very consistent techniques, and (3) BFS_cycle is inconsistent for both weightings .75/.25 and 1/0 on 1-cycles, and 1/0 for 2-cycles. This data implies that a .5/.5 weighting technique gives the most consistent results and that 1/0 gives the least consistent results.

From this information, we concluded that the weighting has a large impact on the efficiency of the algorithms, and that the ordering has a less of an effect. We also see that BFS_cycle has potential to outperform all the other techniques since it most commonly solves the constraints in 1 iteration, which is the best that any technique can do.

5.2.2 Reverse

BFS_cycle had inconsistent behavior with a weighting scheme of 1/0, however during its high performance, it was at or near perfect, solving the

constraints most commonly in one iteration. This led us to believe that the technique has great potential to be an extremely efficient algorithm. DFS_cycle was performing very consistent on a .5/.5 weighting method, however its average is still too high to be superior to Ragdoll.

Since both of these algorithms show room for improvement, but with BFS_cycle with 1/0 weighting seeming the best candidate, we have extended both methods further. We saw that there was oscillation in the motion, and thought it might be caused by iterating over the constraints, always in the same order. Thus we created a technique which traverses the constraints forward, while doing repairs, and then reverses itself, if another iteration is needed.

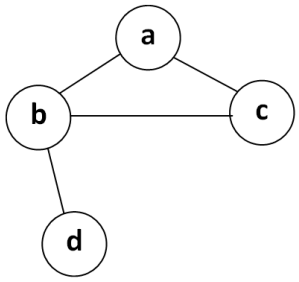


Figure 5.5: An example of a 1-cycle graph. BFS_reverse order produced from root a: {(a,b),(a,c),(b,d),(c,b),(c,b),(b,d),(a,c),(a,b)}. Problem constraint:{(c,b)}. DFS_reverse order produced from root a: {(a,b),(b,d),(b,c),(c,a),(c,a),(b,c),(b,d),(a,b)}. Problem constraint:{(c,a)}.

Approach. First, perform either a BFS_cycle or a DFS_cycle, generating a forward ordering, then reverse that returned ordering and concatenate it to the end of the original (forward) ordering. See Figure 5.5 for an example.

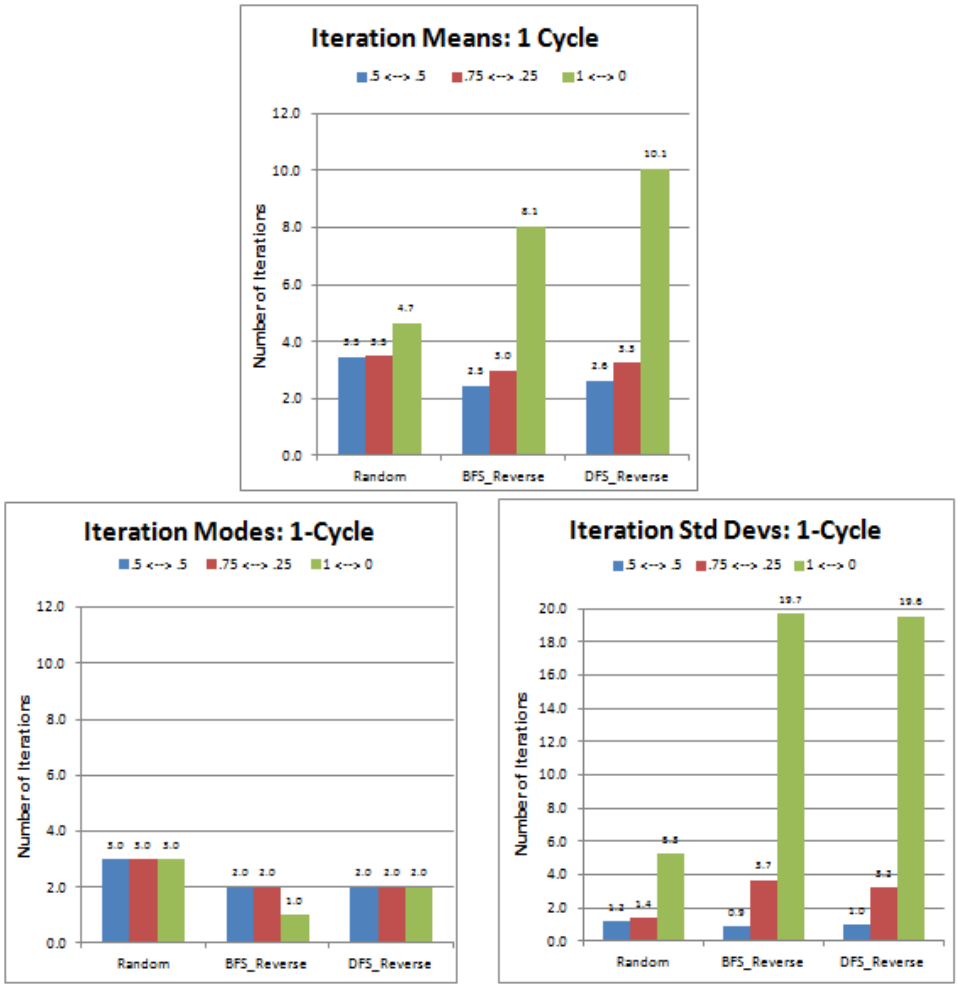


Figure 5.6: BFS_reverse and DFS_reverse 1-cycle graph results.

Results and Discussion. Our results after performing BFS_reverse and DFS_reverse on 1-cycle and 2-cycle graphs can be seen in the charts in Figures 5.6 and 5.7 which again will be compared to a Ragdoll and previous orderings.

First, looking at the means, observe that BFS_reverse, DFS_reverse, are performing very similarly to how BFS_cycle and DFS_cycle performed.

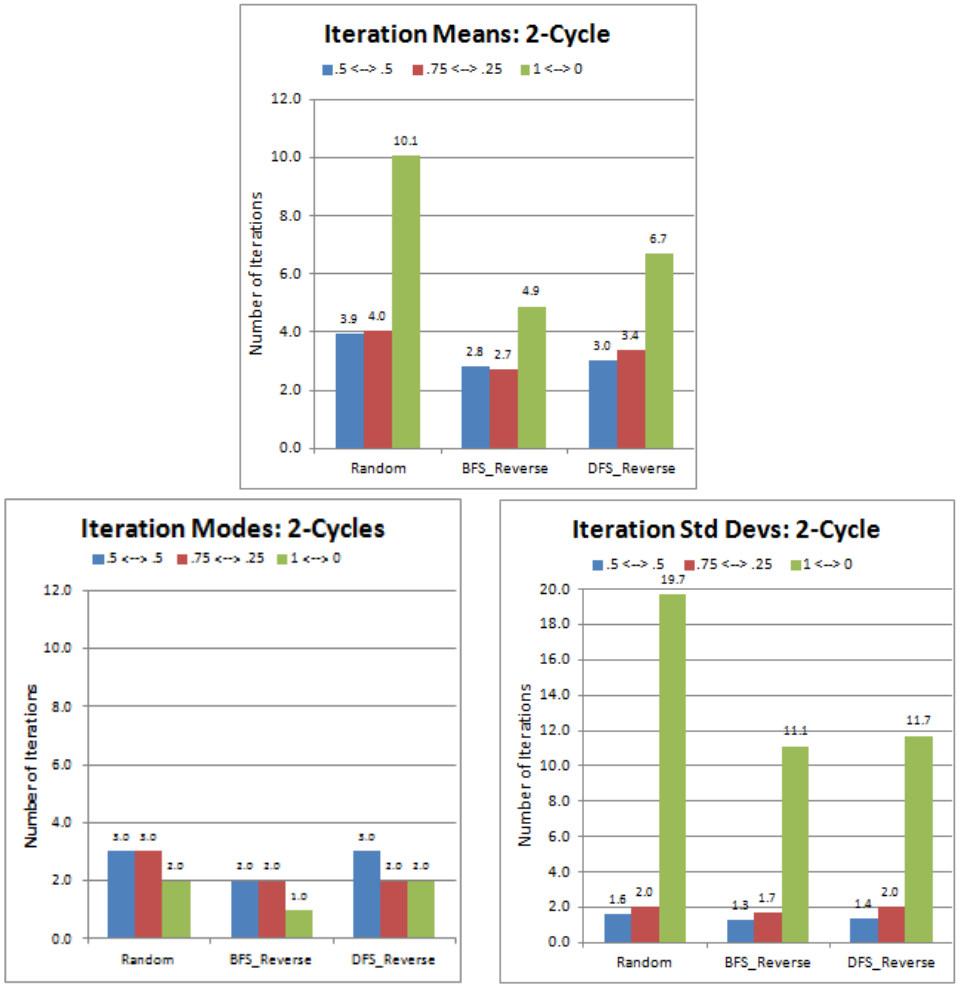


Figure 5.7: BFS_reverse and DFS_reverse 1-cycle graph results.

Also, note that BFS_reverse with .5/.5 weighting outperforms the other techniques, however, with a 1/0 weighting it performs much worse than the other techniques. BFS_cycle with a .5/.5 weighting is currently the technique with the lowest mean for both 1-cycles and 2-cycles.

Next, focusing on the modes, we can see that (1) DFS_reverse has dropped to a mode of 2 for both 1-cycles, and 2-cycles for all weighting schemes except .5/.5 for 2-cycles, (2) BFS_reverse has dropped to a mode

of 2 for both 1-cycles and 2-cycles for weightings .5/.5 and .75/.25, and (3) BFS_reverse still most commonly solves the constraints in 1 iteration, but also still has the worst mean. This suggests that BFS_reverse with a 1/0 weighting is still inconsistent in the same way that BFS_cycle was.

Finally, examining the standard deviations, we can recognize that (1) BFS_reverse with 1/0 weighting is extremely high, which was suggested by the high mean but low mode, (2) BFS_reverse and DFS_reverse with a .5/.5 weighting have even lower standard deviations, suggesting that they are even more consistent than BFS_cycle and DFS_cycle, and (3) BFS_reverse, and DFS_reverse are inconsistent for both weightings .75/.25 and 1/0 on 1-cycles and 2-cycles. This data empirically establishes that a .5/.5 weighting technique gives the most consistent results, and that 1/0 gives the least consistent results for graphs containing cycles.

From this information, we have found a new (albeit nominally) best technique, BFS_reverse, which performs best for 1-cycles and 2-cycles. Since BFS_reverse performs well at these complexities, then as the number of cycles in a graph increases, as in many real-world cases, BFS_reverse may do increasingly better.

5.3 Techniques unique to 1-cycles

In this section we will discuss our final constraint satisfaction techniques which are unique to 1-cycles. These techniques were designed to be adaptive in that they change the root joint during iteration. We will end this section with an analysis of the results presented in Figure 5.9.

5.3.1 Adapt

The reverse techniques showed a large improvement over our previous techniques, however, we would still like to improve upon them. We noticed that techniques ordered with DFS have not been performing as well as those with BFS, so for the rest of this chapter, we will be focusing only on a BFS ordering.

During the iteration phase, changing the constraint ordering, as reverse techniques did, had a positive impact on the results. Therefore, we have created an adaptive technique that will pseudo-learn where the break is rather than doing the same ordering forward and then backward. We hypothesised that the so-called *problem constraint* is where the break would be after an iteration. Therefore we concluded we should start the next iteration from the where the structure was last known to be broken, the problem constraint.

Approach. The ordering begins by calling BFS_cycle. However, the problem constraint is stored, and on the next iteration, rather than starting from the same root joint, it starts from the problem constraint and calls BFS_cycle from there. See Figure 5.8 for an example.

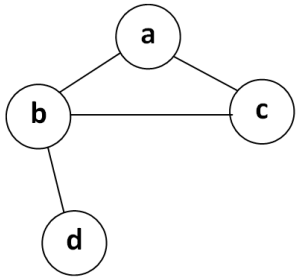


Figure 5.8: An example of a 1-cycle graph. BFS_adapt order produced from root a: {(a,b),(a,c),(b,d),(c,b)}. Problem constraint:{(c,b)}.

Results and Discussion. Our results after performing BFS_adapt on 1-cycle graphs can be seen in Figure 5.9 which will be compared to a random and previous orderings.

Observing the charts in Figure 5.9, it is easy to see that BFS_adapt is much worse than previous techniques, including Ragdoll. BFS_adapt has a high mean, a high mode with the exception of a 1/0 weighting, and a high standard deviation with the exception of a .5/.5 weighting. BFS_adapt is still getting similar results to the original BFS_cycle in that it is very inconsistent, sometimes performing exceptionally, and others very poorly.

BFS_reverse is still the best technique, however we are surprised by the results of BFS_adapt, since adapting the root joint during iteration

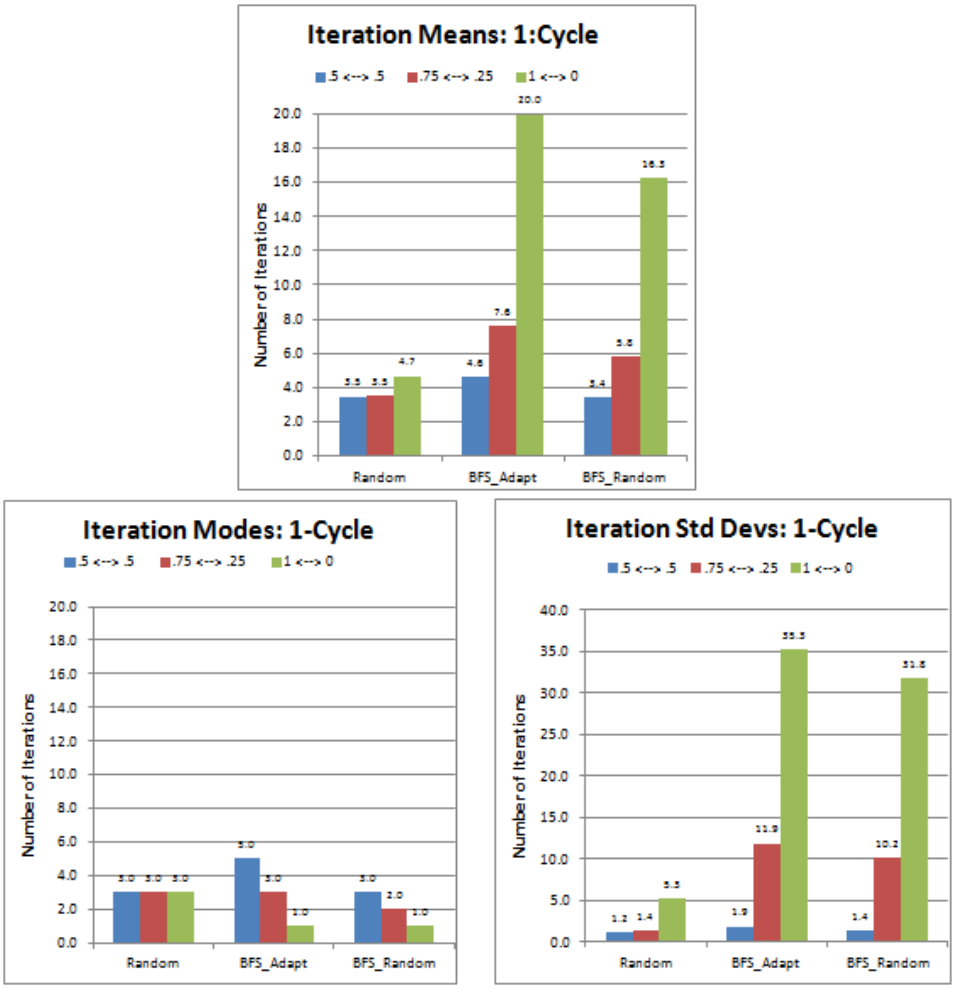


Figure 5.9: BFS_adapt and BFS_random 1-cycle graph results.

worsened the results.

5.3.2 BFS_Random

BFS_adapt did not perform nearly as well as we had hypothesized. Since both a random ordering and BFS ordering are performing well, the next approach we developed combines the advantages of Ragdoll physics and BFS_cycle. We still believe our initial intuition that changing the root

joint during iteration will give the best results.

Approach. The ordering begins by calling BFS_cycle. However, on the next iteration, rather than starting from the same root joint, it starts from a random joint in the graph. See Figure 5.10.

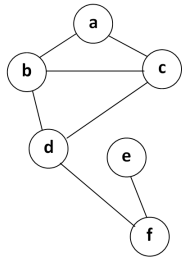


Figure 5.10: An example of a 2-cycle graph. BFS_random order produced from root a: {(a,b),(a,c),(b,d),(c,b),(d,f),(c,d),(f,e)}. Problem constraints:{(c,b),(c,d)}.

Results and Discussion. Our results after performing BFS_random on 1-cycle graphs can be seen in the charts in Figure 5.9 which will be compared to a Ragdoll and previous orderings.

Examining Figure 5.9, we observed that BFS_random was an improvement to BFS_adapt, but is very similar to BFS_cycle. BFS_random performed best with a .5/.5 weighting however, with a weighting of 1/0 it still has a mode of 1.

BFS_random is the better of our two techniques that try to learn where the breaks in the structure are, however BFS_reverse with a .5/.5 weighting is still the best technique overall for 1-cycles and 2-cycles.

Chapter 6

Conclusions

This thesis has striven to realize two main goals: 1) **real-time motion simulation** of *geometric constraint structures*, and 2) efficient **constraint satisfaction** after applied forces break the constraints of a structure.

The first problem has important applications to CAD, proteins, and robotics. We approached it by utilizing geometric constraint structures, specifically the bar-and-joint structure, to model structures and their interactions. This allowed the extension of methodologies from Ragdoll physics by incorporating classical search algorithms to develop new enhanced algorithmic approaches to the second problem. By utilizing these new algorithms in a motion simulation tool we developed, motionSim, we were able to evaluate the methodologies to determine the most efficient constraint resolution techniques for each our of our classes of graphs. Our analysis shows that BFS_reverse consistently outperforms other constraint satisfaction techniques, including Ragdoll physics, that the weighting of the constraints has a larger impact on the number of iterations than the

ordering does, and that our techniques may have a potential to improve motion simulation of molecular motion.

Future work would include (1) determining why BFS_cycle performed rather inconsistently, (2) experimenting further with constraint satisfaction technique that adjust the root, trying to learn where the breaks in the structure are, (3) validate our results on larger and more complex data sets, (4) extend our constraint satisfaction techniques to other classes of graphs, (5) compare the structure of proteins to our classes of structures to determine the feasibility of our techniques for protein folding and flexibility, and (6) develop and evaluate new constraint satisfaction techniques.

Appendix A

Physics Behind Verlet Integration

The Verlet Integration formula is Pos_{new}= 2*Pos_{cur} - Pos_{old}+ A*dt*dt and can be derived from Euler's formulas along with common physics formulas. Below are two derivations for the Verlet Integration formula.

Euler:

1a Vel_{new}= Vel_{cur} + A*dt

1b Pos_{new}= Pos_{cur} + Vel_{new} *dt

Merge and simplify:

1c Pos_{new}= Pos_{cur} + (Vel_{cur} + A*dt)*dt

1d Pos_{new}= Pos_{cur} + Vel_{cur}*dt + A*dt*dt

Position Verlet: Assumes a constant acceleration and time step and considers Vel_{cur}*dt to be approximated by (Pos_{cur}- Pos_{old}). Thus giving us:

1e Pos_{new}= Pos_{cur} + (Pos_{cur}- Pos_{old})+ A*dt*dt

OR

1f Pos_{new}= 2*Pos_{cur} - Pos_{old}+ A*dt*dt

Using physics and holding the acceleration constant:

a(t)=a, v(t)=at+v, $x(t)=0.5a*t^2+v*t+x_0$.

Thus:

2a Pos_{cur}= 0.5A*dt*dt+Vel_{old}*dt+ Pos_{old}

2b Vel_{cur}=A*dt+Vel_{old}

2c Vel_{old}= Vel_{cur} -A*dt

Also:

3a Pos_{new}= 0.5A*dt*dt+Vel_{cur}*dt+ Pos_{cur} and

3b Vel_{new}=A*dt+Vel_{cur} and thus

3c Vel_{cur}= Vel_{new} -A*dt

Following from (2a) we have:

2d Pos_{cur}- Pos_{old} = 0.5A*dt*dt+Vel_{old}*dt

Following from (3a) we have:

3d Pos_{new}- Pos_{cur} = 0.5A*dt*dt+Vel_{cur}*dt

Plugging (2c) into (2d) we get:

2e Pos_{cur}- Pos_{old} = 0.5A*dt*dt+ (Vel_{cur} -A*dt) *dt

Simplifying we get:

2f Pos_{cur}- Pos_{old} = -0.5A*dt*dt+ Vel_{cur}*dt

Adding (A*dt*dt) to each side of (2f) we get:

2g Pos_{cur}- Pos_{old} +A*dt*dt=0.5A*dt*dt+ Vel_{cur}*dt

The right side of (2g) is equal to the right side of (3d), thus we get:

2h Pos_{cur}- Pos_{old} +A*dt*dt=Pos_{new}- Pos_{cur}

Then (2h) becomes (1f) by combining terms:

1f and 2i: Pos_{new}= 2*Pos_{cur} - Pos_{old}+ A*dt*dt

Appendix B

BFS and DFS Pseudo-Code

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

Figure B.1: Pseudo-code for the Breadth-First Search algorithm. Figure reproduced from [8].

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
 1  time = time + 1              // white vertex u has just been discovered
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]        // explore edge (u, v)
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK              // blacken u; it is finished
 9  time = time + 1
10  u.f = time
```

Figure B.2: Pseudo-code for the Depth-First Search algorithm. Figure reproduced from [8].

Appendix C

Constraint Satisfaction Results

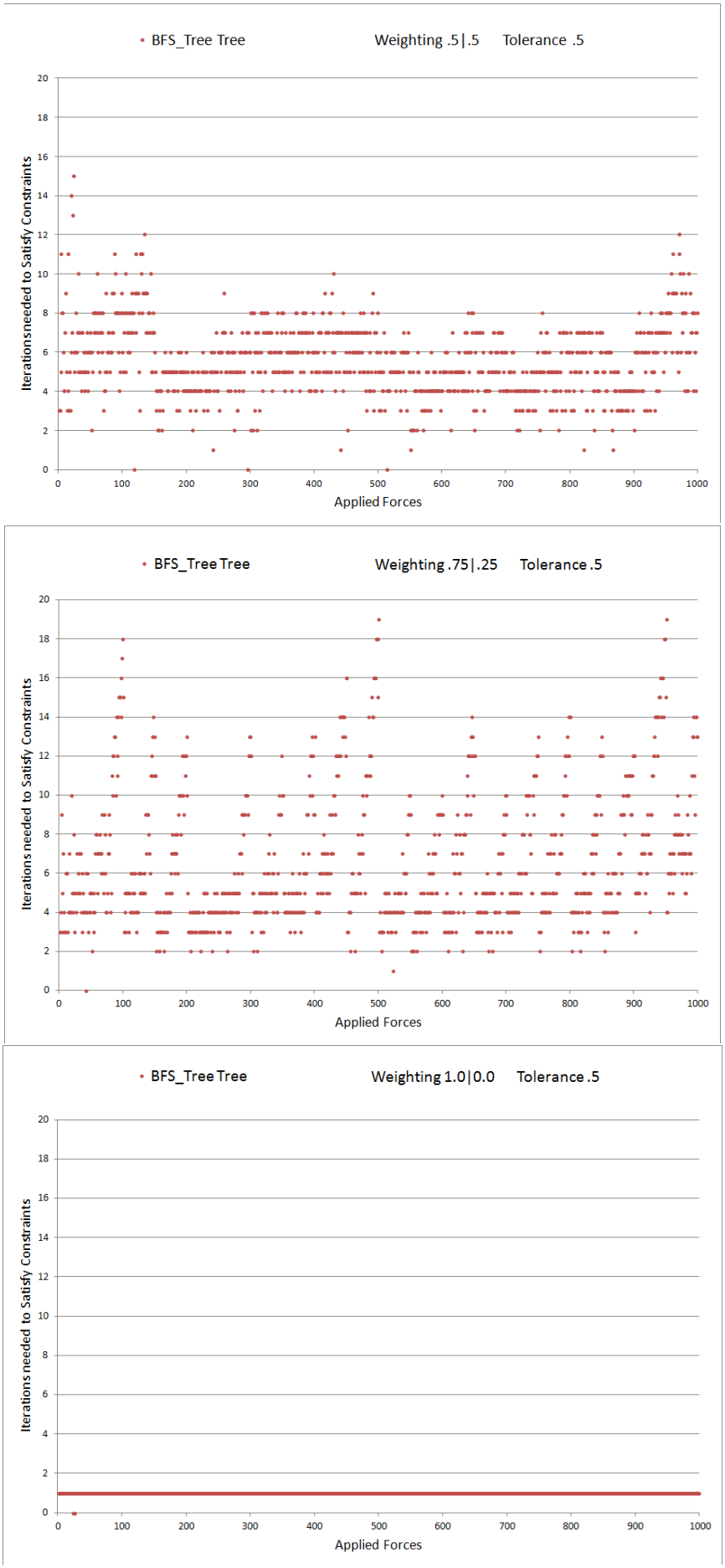


Figure C.1: BFS_tree tree graph results.

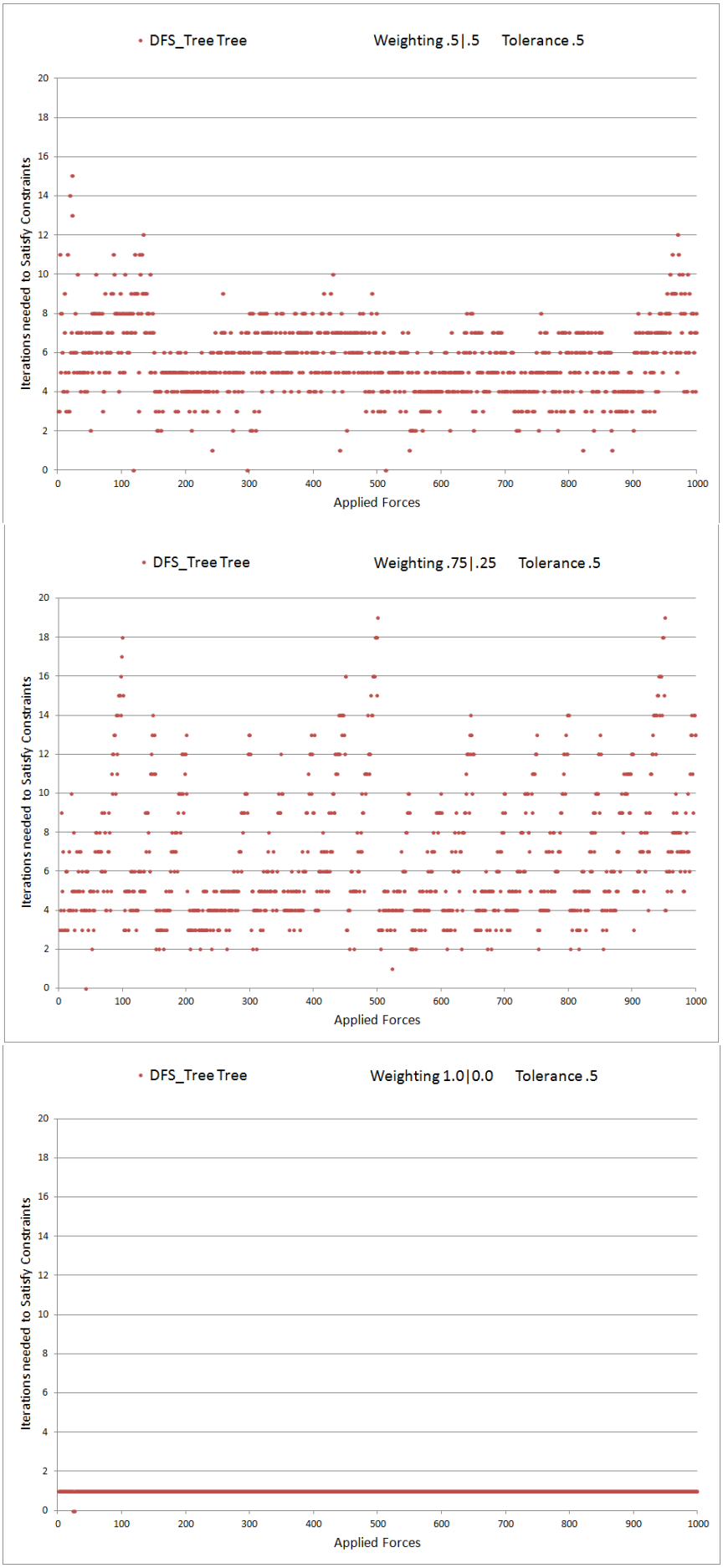


Figure C.2: DFS_tree tree graph results.

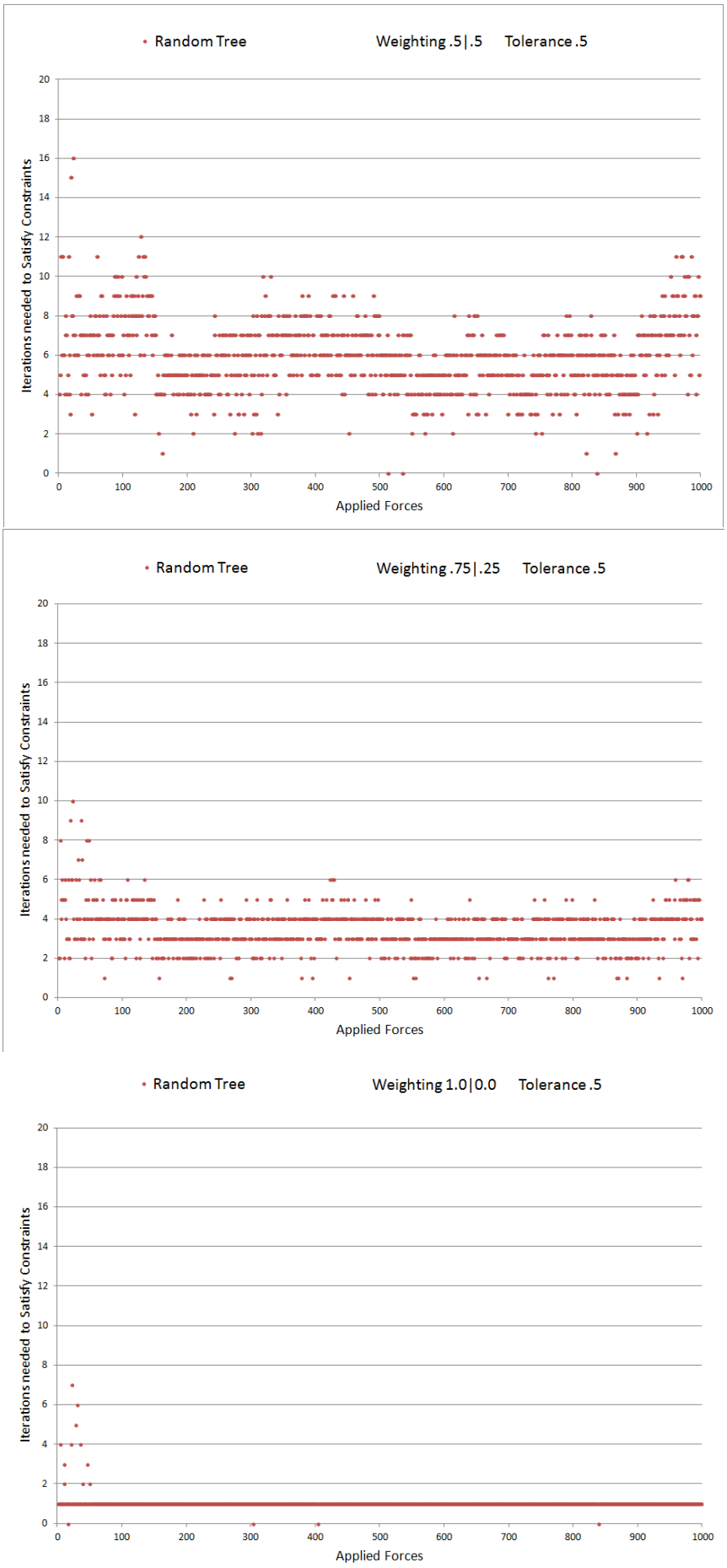
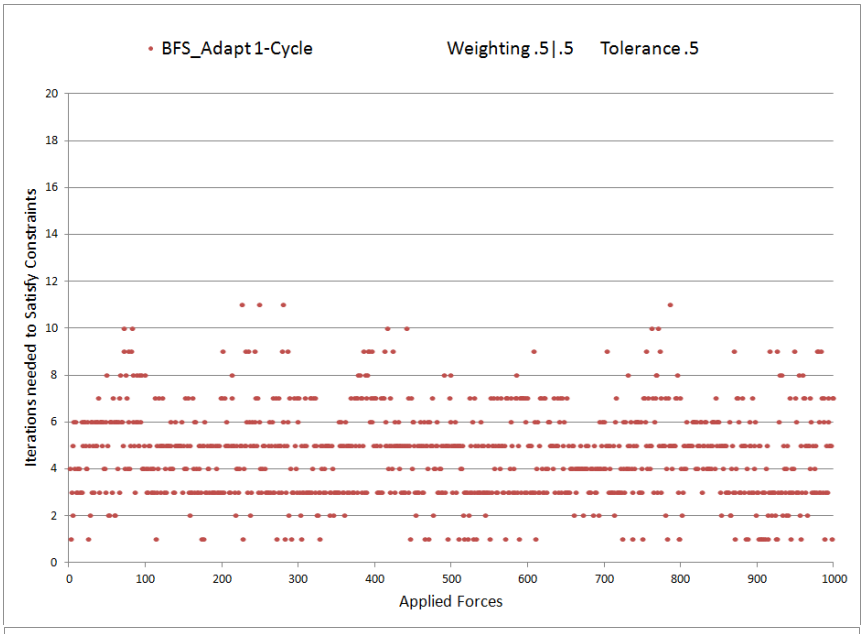
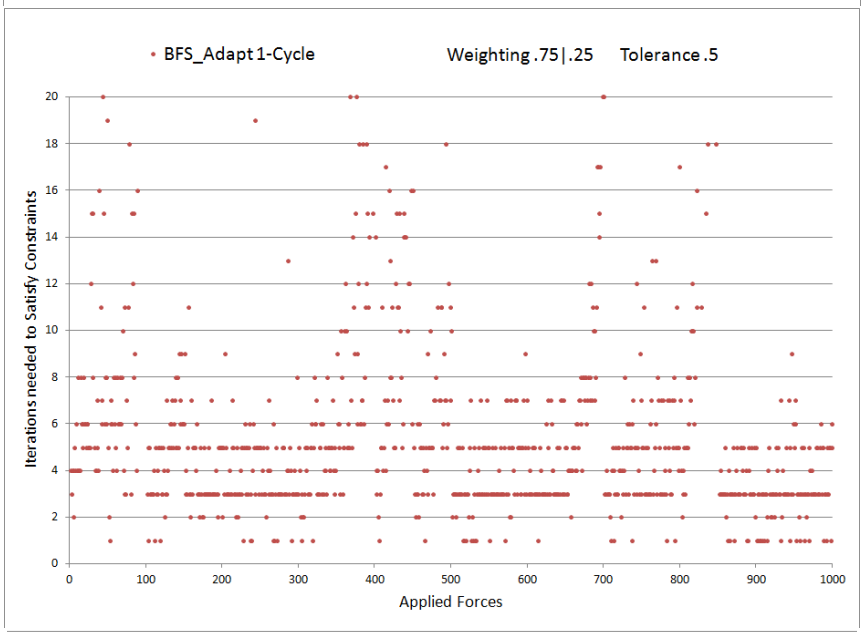


Figure C.3: Random tree graph results.





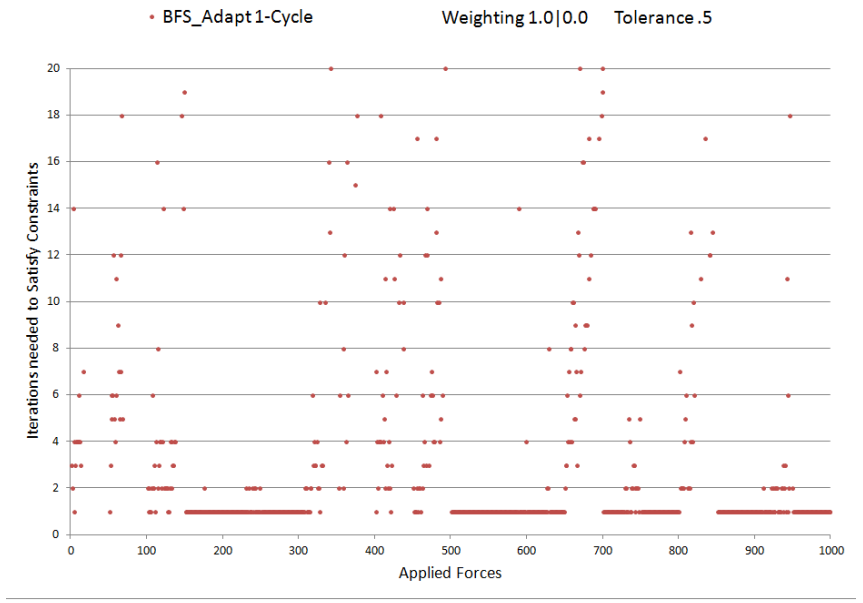
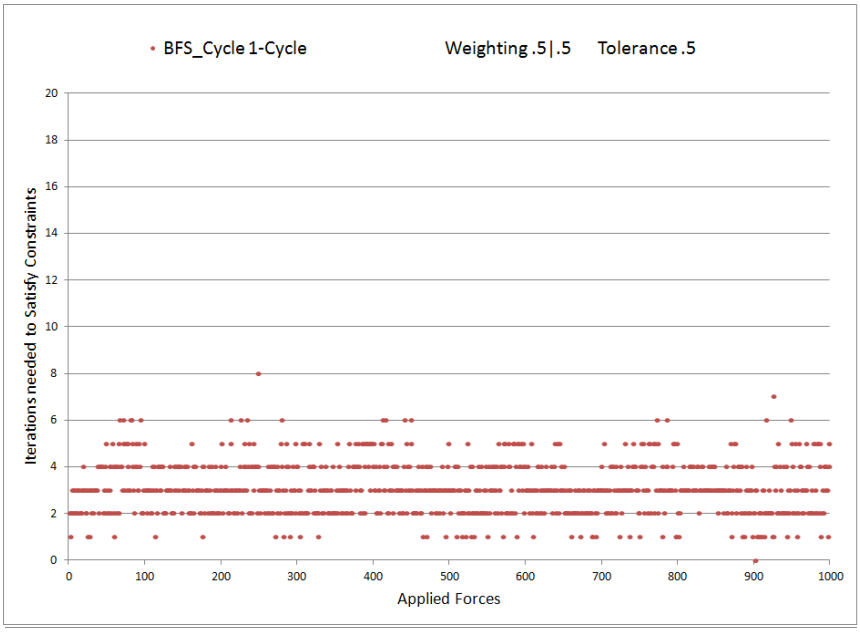
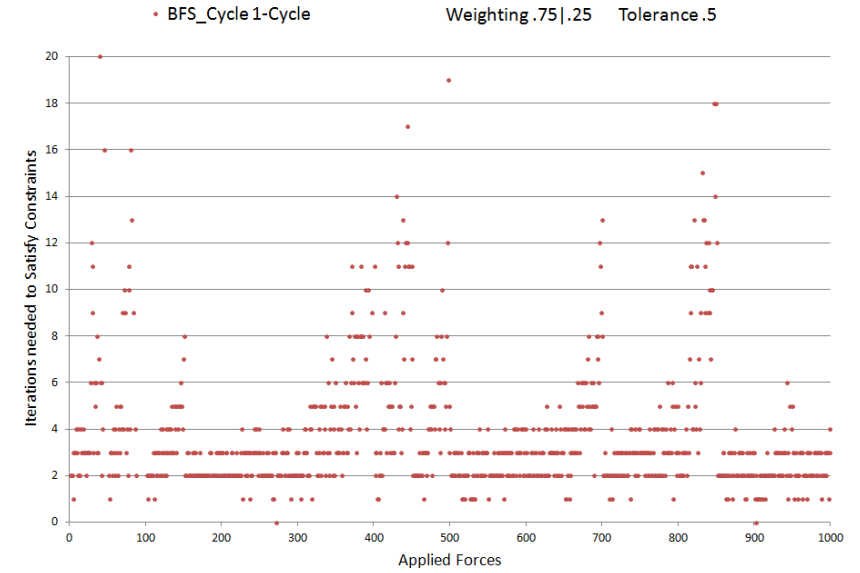


Figure C.4: BFS_adapt 1-cycle graph results.





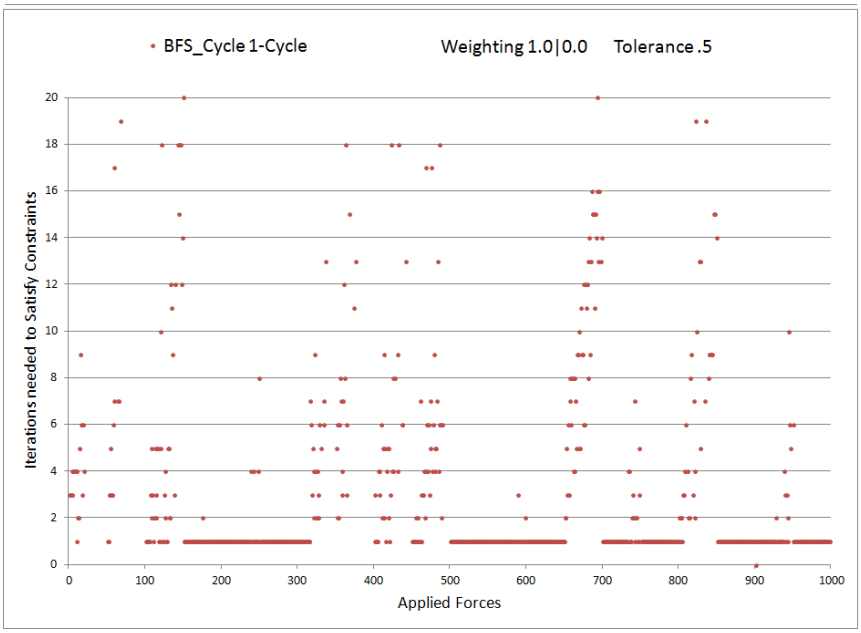


Figure C.5: BFS_cycle 1-cycle graph results.

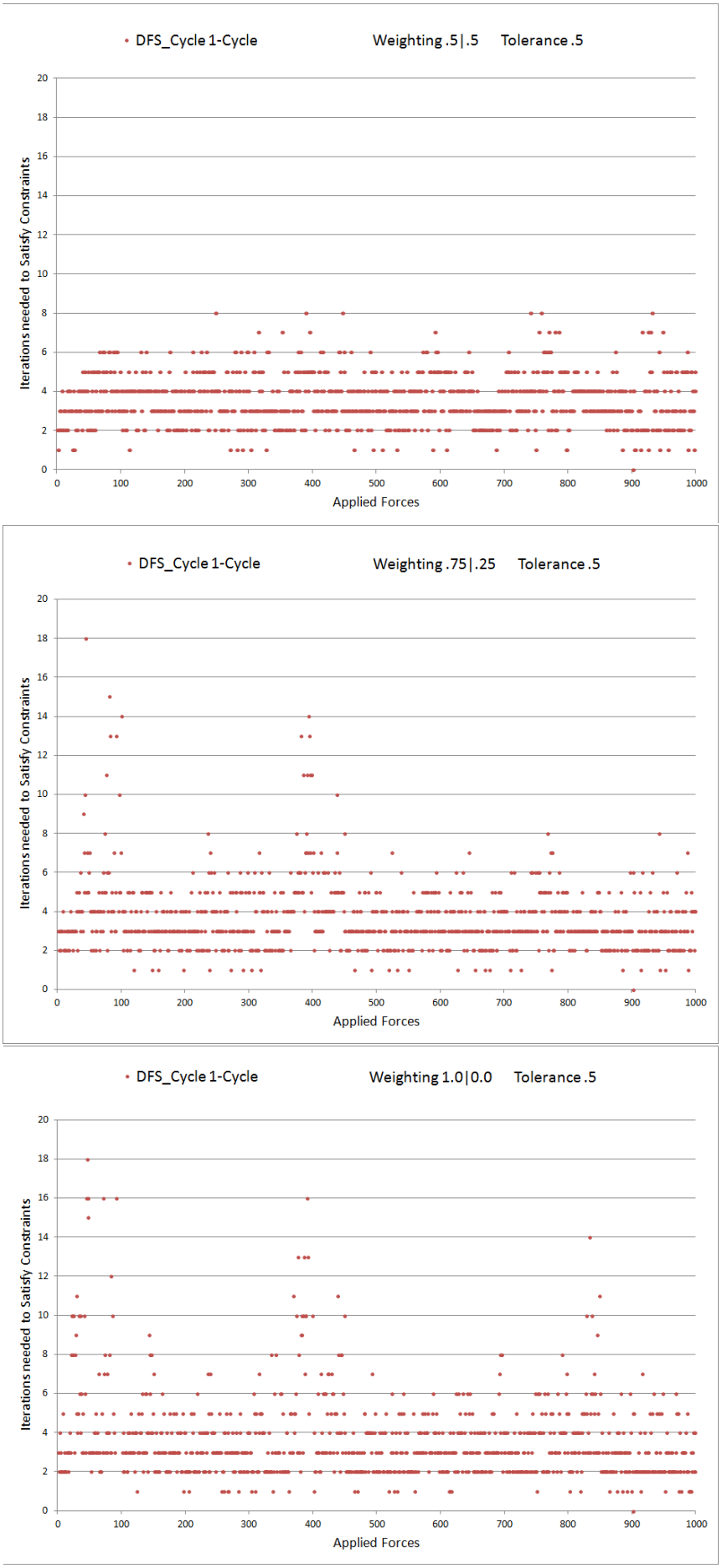


Figure C.6: DFS_cycle 1-cycle graph results.

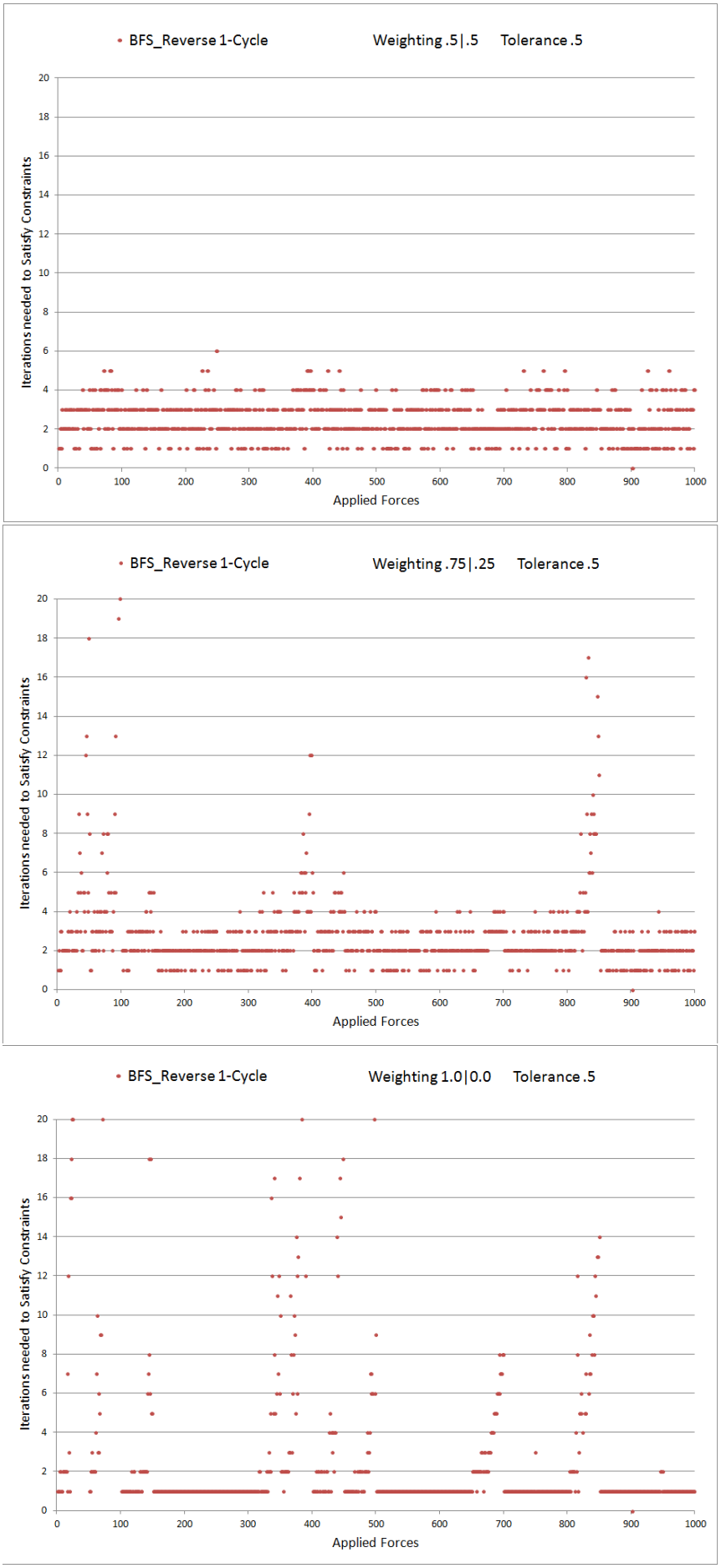
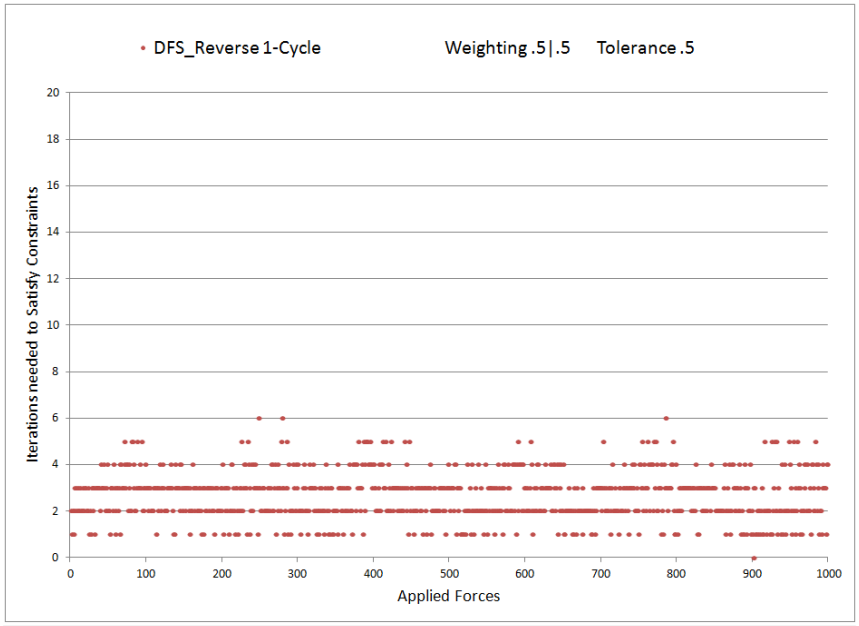
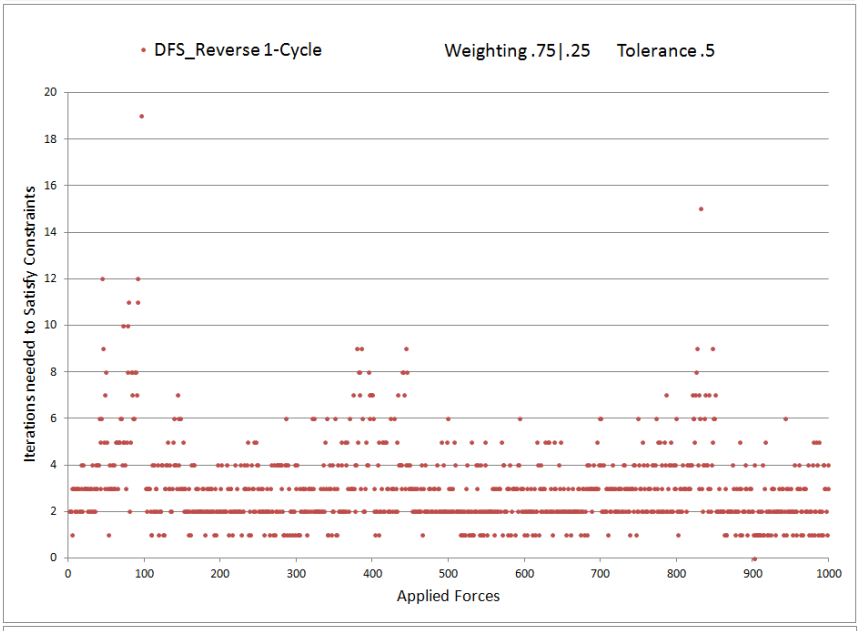


Figure C.7: BFS_reverse 1-cycle graph results.





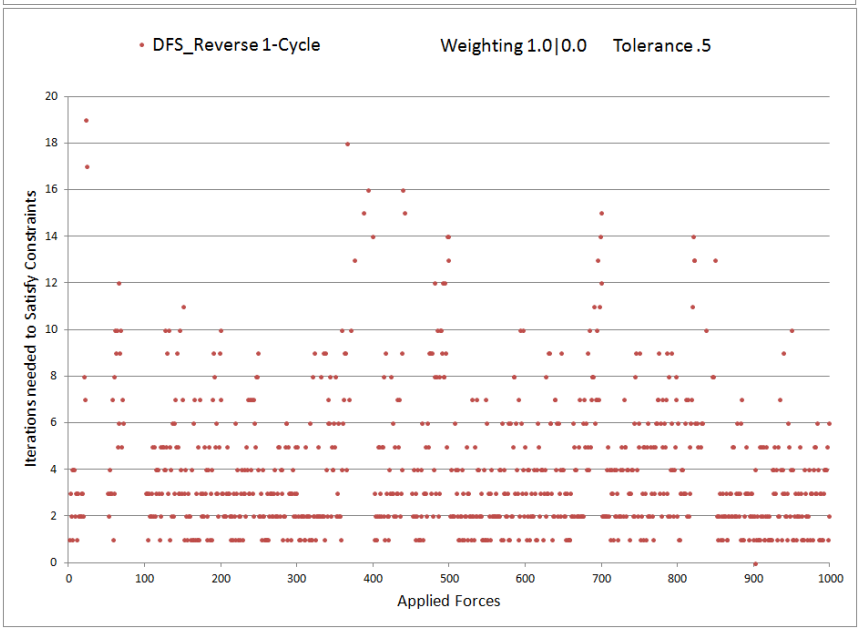


Figure C.8: DFS_reverse 1-cycle graph results.

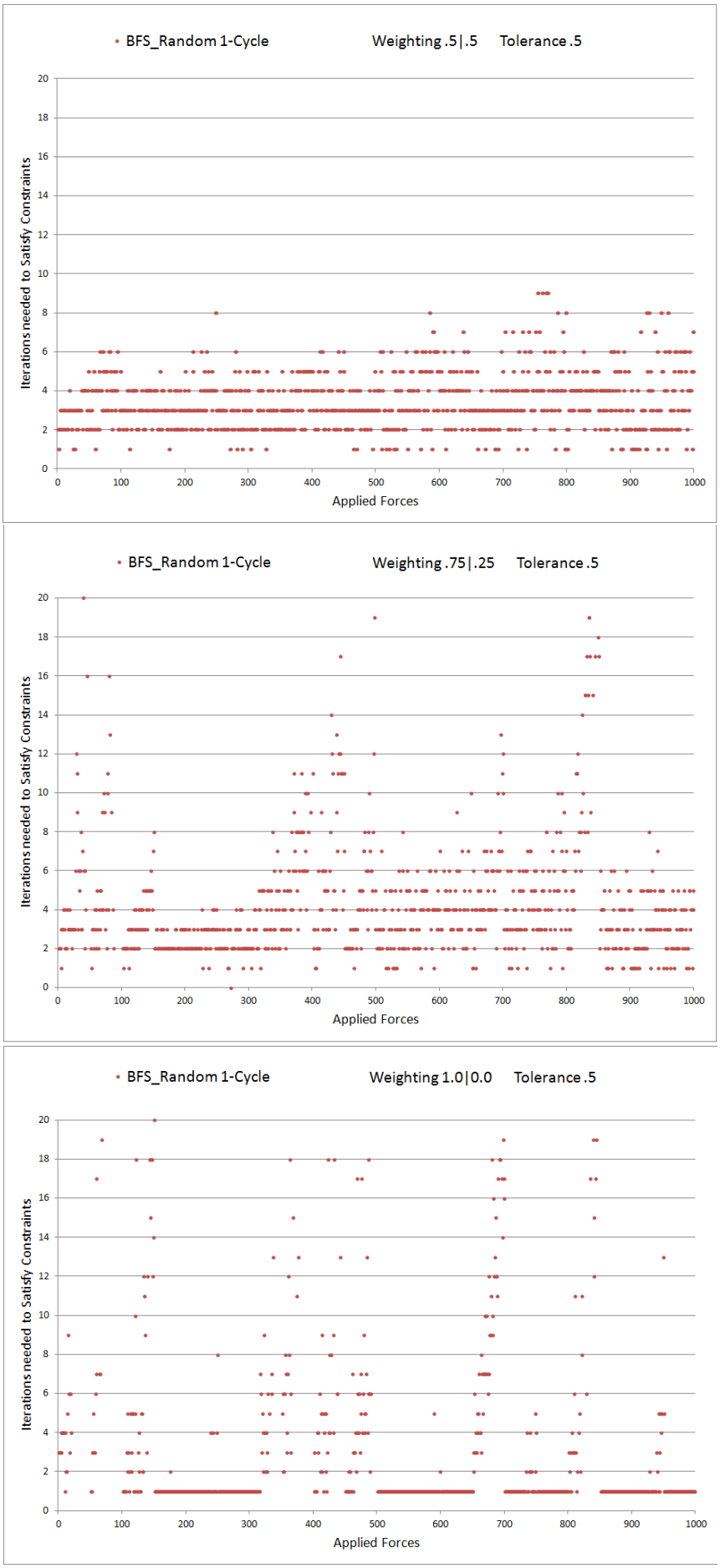
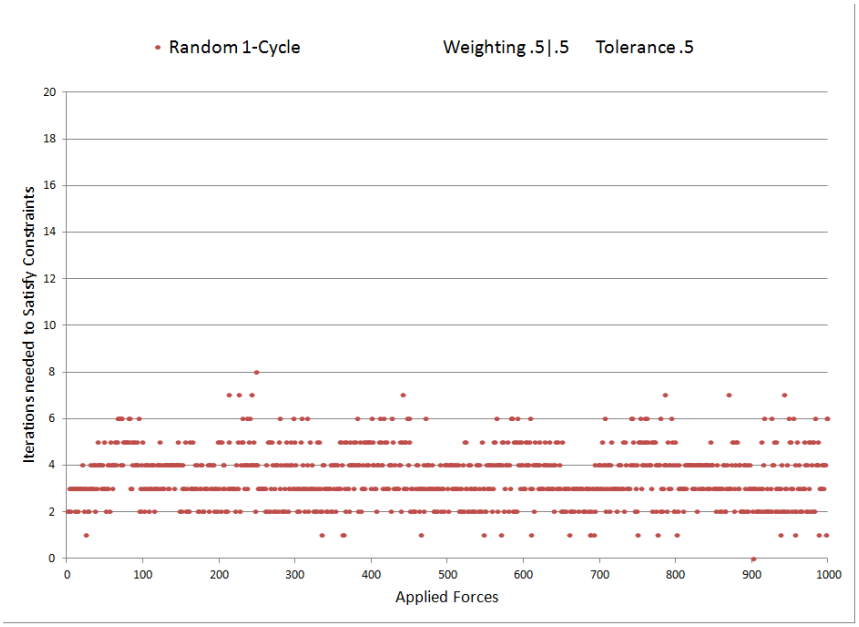
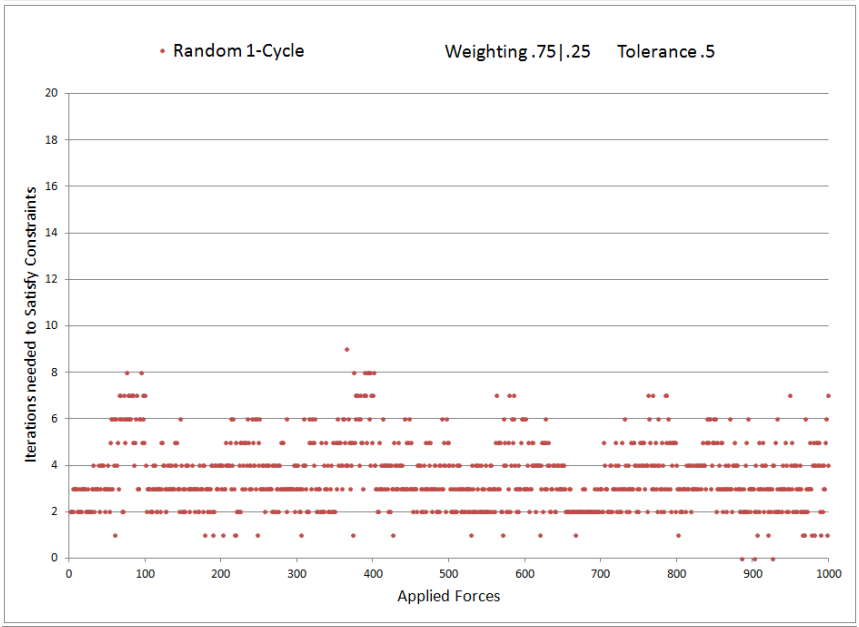


Figure C.9: BFS_random 1-cycle graph results.





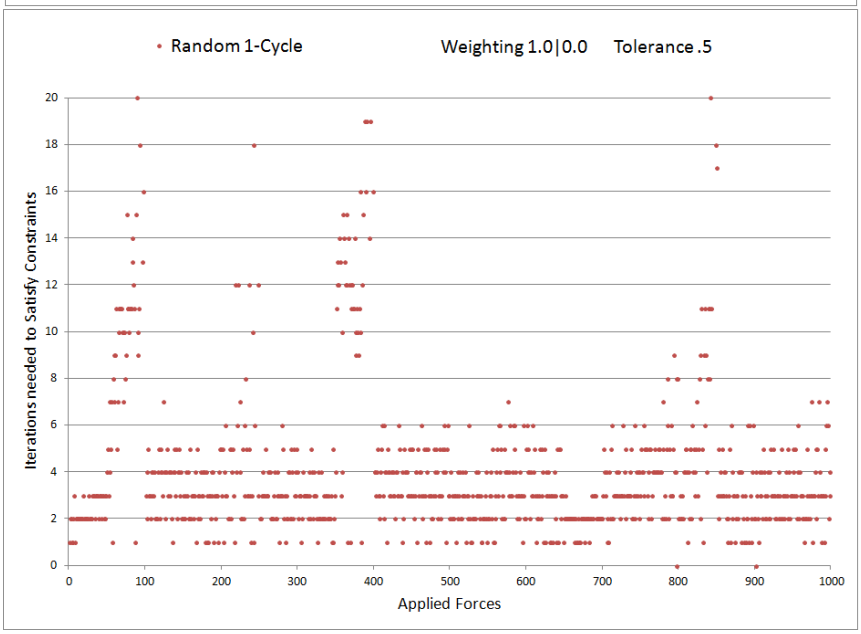
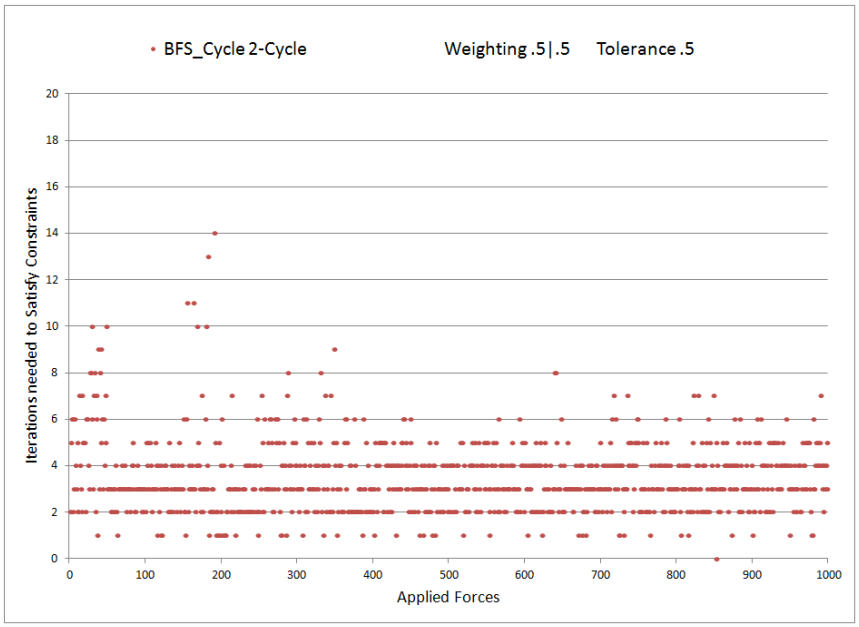
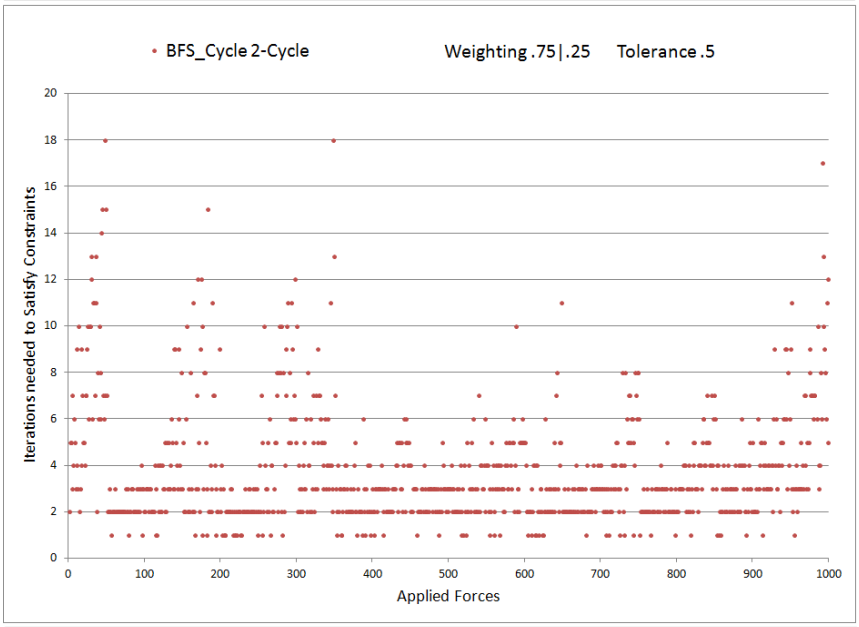


Figure C.10: Random 1-cycle graph results.





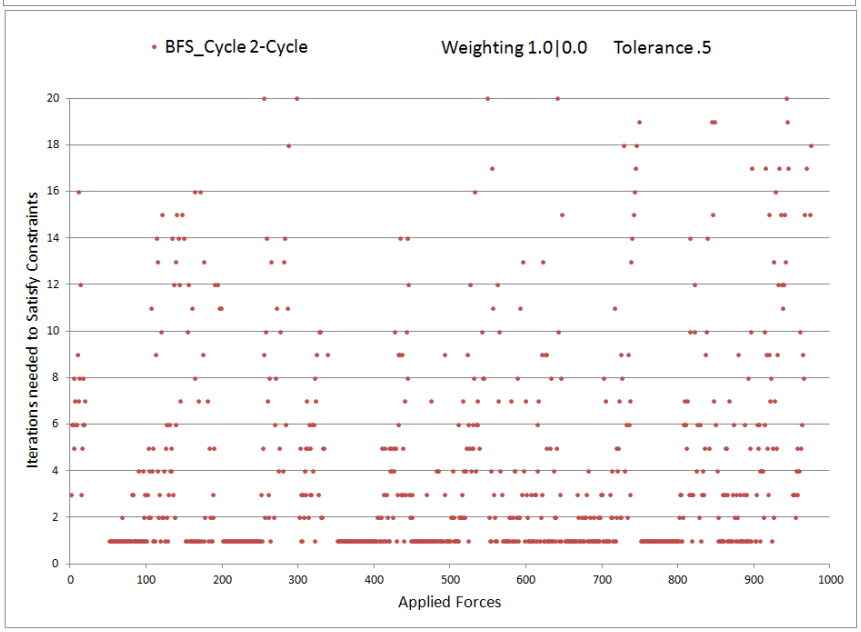


Figure C.11: BFS_cycle 2-cycle graph results.

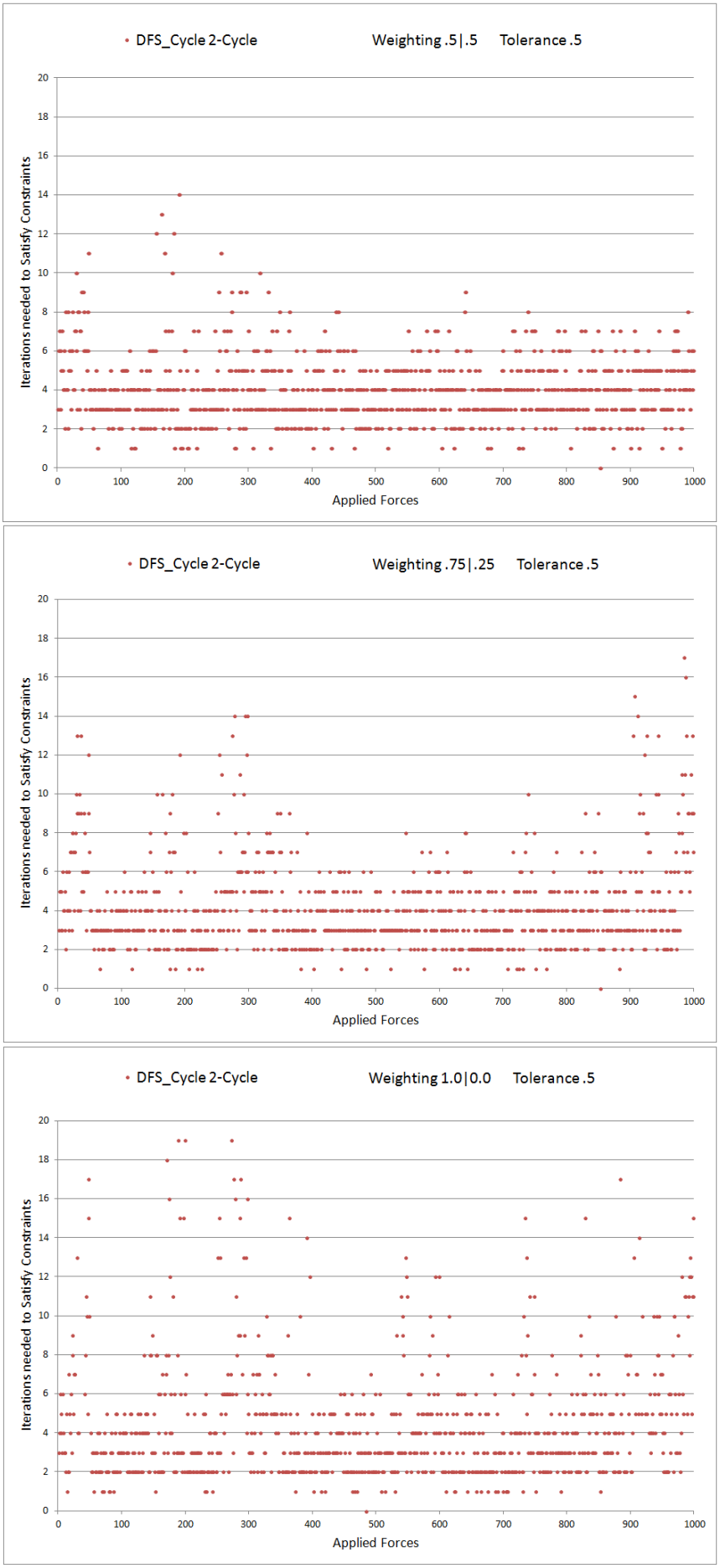
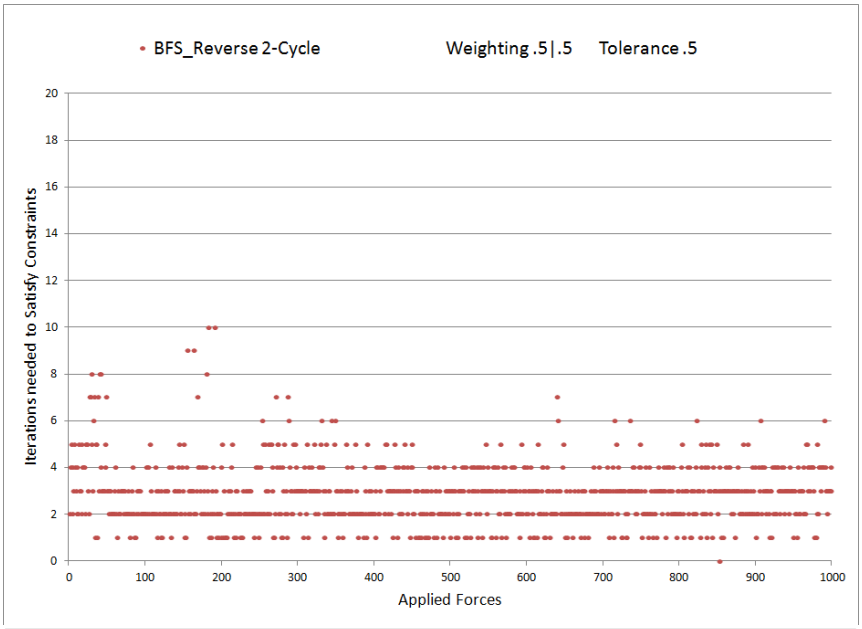
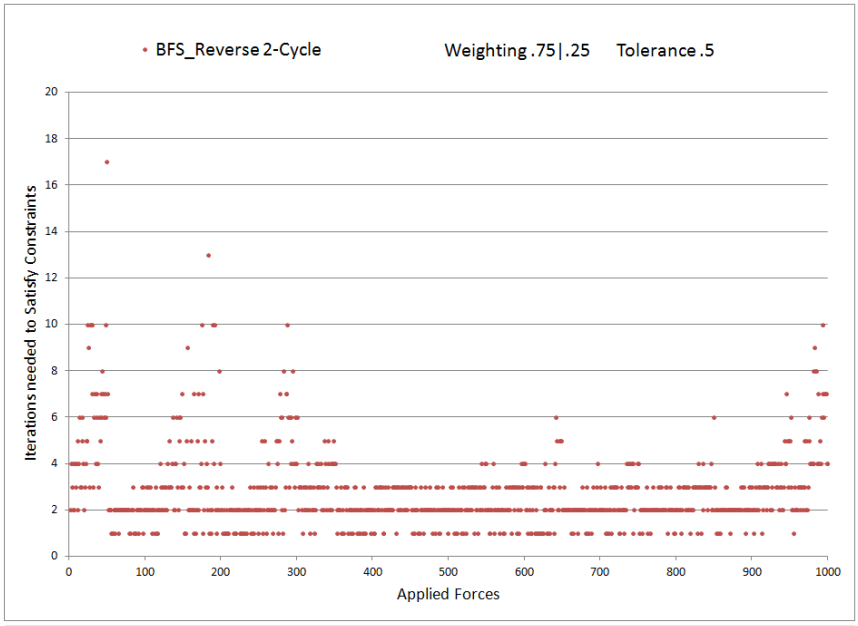


Figure C.12: DFS_cycle 2-cycle graph results.





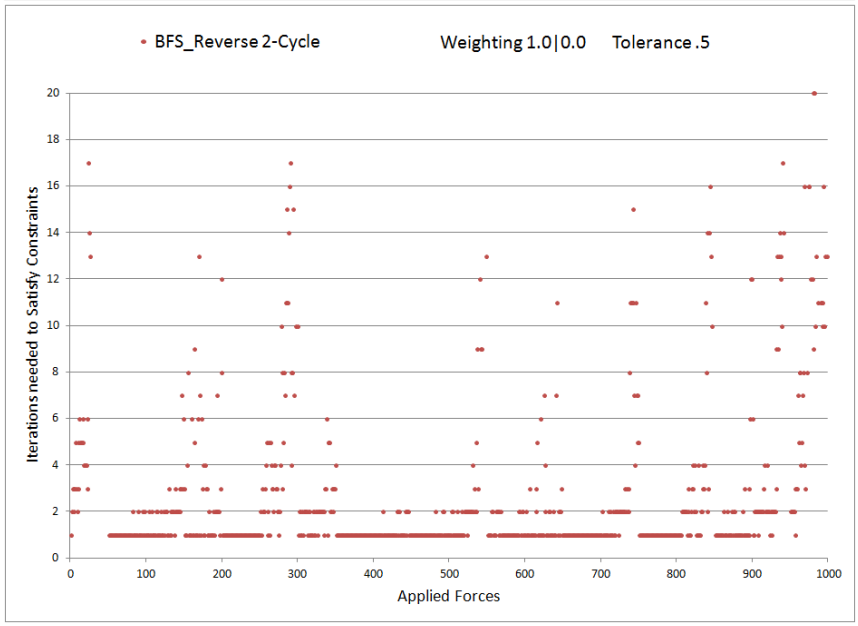


Figure C.13: BFS_reverse 2-cycle graph results.

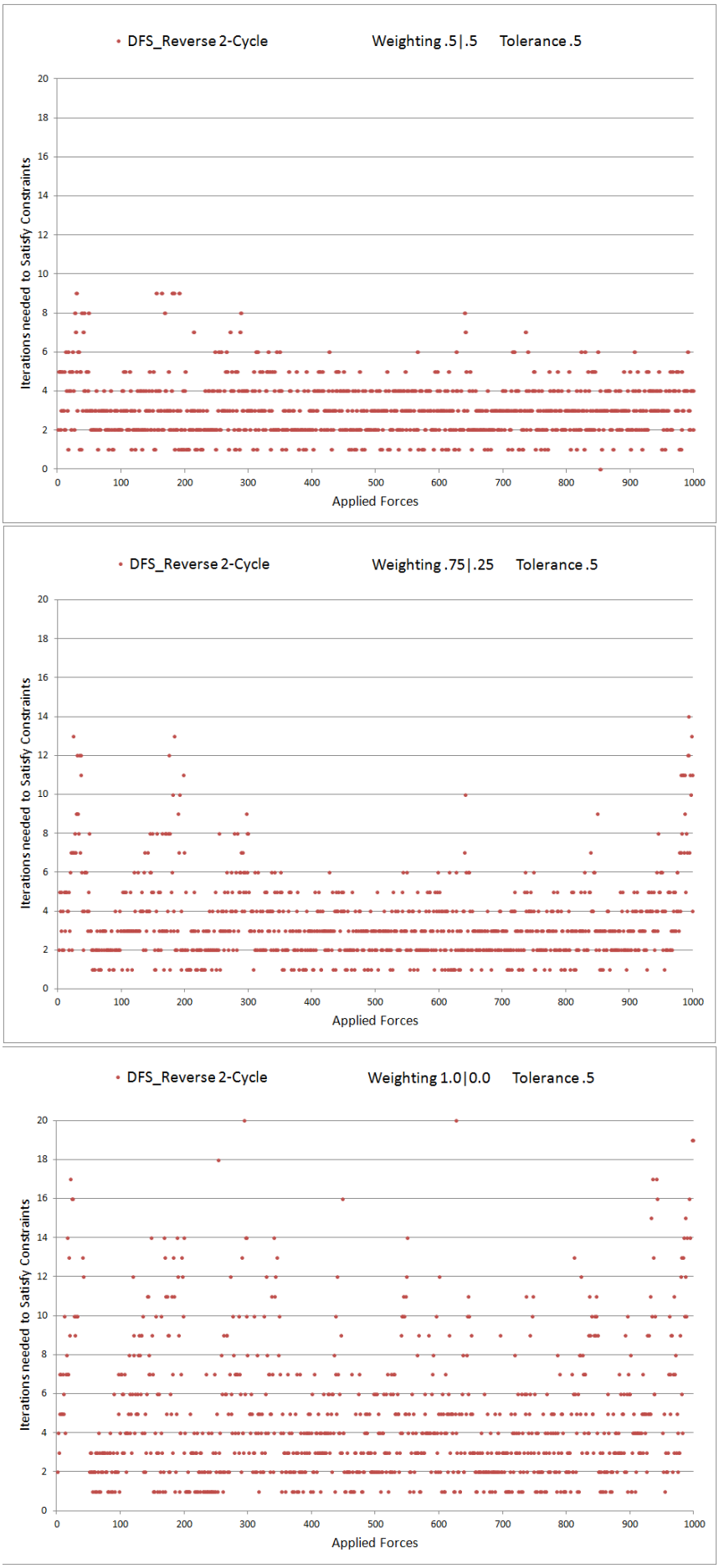
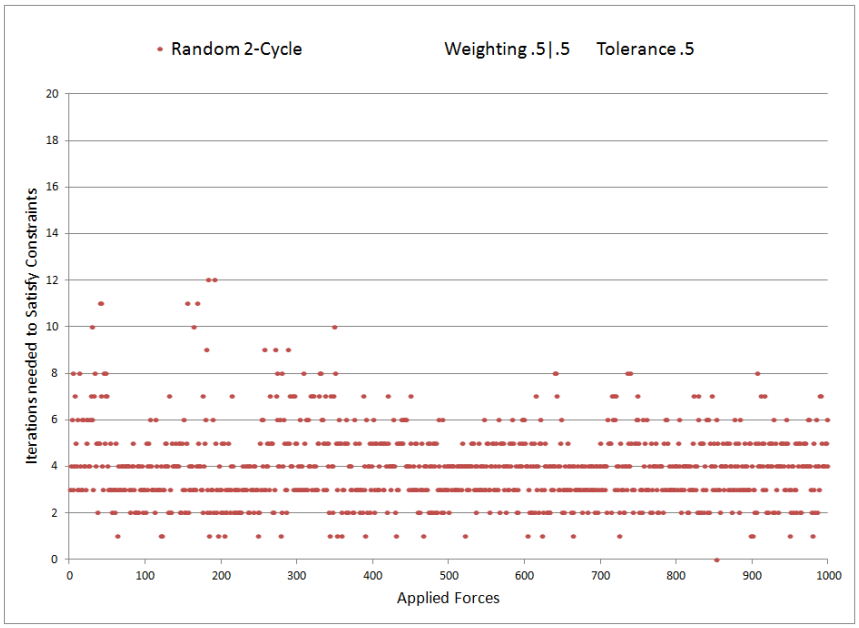
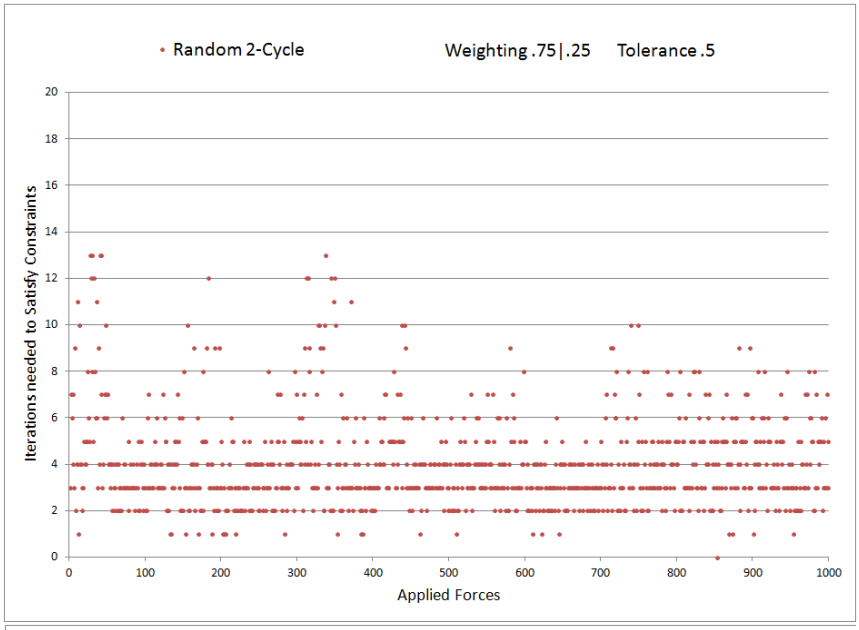


Figure C.14: DFS_reverse 2-cycle graph results.





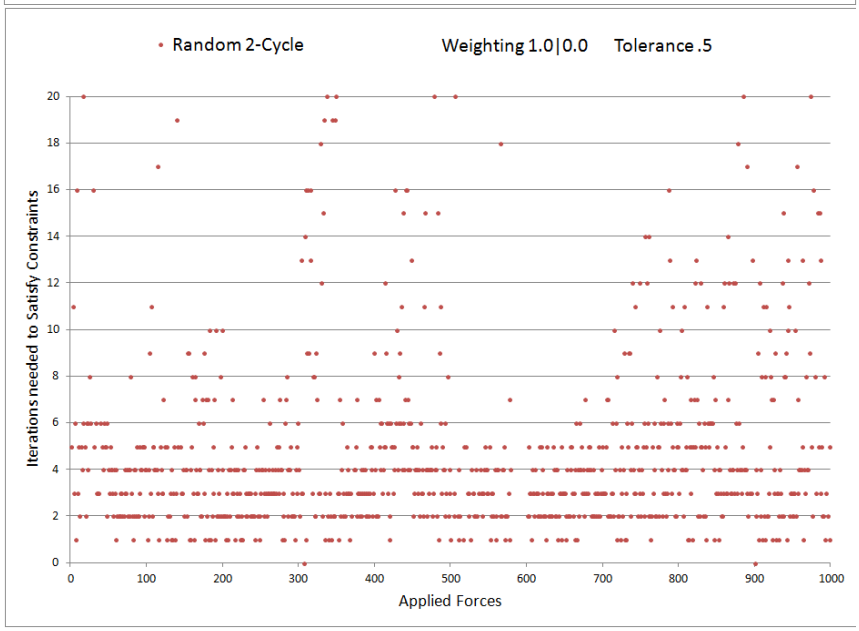


Figure C.15: Random 2-cycle graph results.

Bibliography

[1] Mechanical cad with multibody dynamic analysis based on modelica simulation. *Proceedings of the 44th Scandinavian Conference on Simulation and Modeling*, September 2003.

[2] Understanding motion simulation. Technical report, SolidWorks, 2008.

[3] Put your designs in motion with event-based simulation. Technical report, SolidWorks, 2009.

[4] J. E. Akin. *Finite Element Analysis Concepts: Via Solidworks*. World Scientific Pub Co Inc, 2010.

[5] Mehmet Serkan Apaydin, Douglas L. Brutlag, Carlos Guestrin, David Hsu, and Jean-Claude Latombe. Stochastic roadmap simulation: an efficient representation and algorithm for analyzing molecular motion. In *Proceedings of the sixth annual international conference on Computational biology*, RECOMB '02, pages 12–21, New York, NY, USA, 2002. ACM.

[6] Aude Bolopion, Barthélemy Cagneau, Stéphane Redon, and Stéphane Régnier. Haptic feedback for molecular simulation, 2009.

[7] J. S Chabura. Development of instructional software for demonstrating cad/fea integration best practices. Master's thesis, University of Illinois at Urbana-Champaign, 2004.

[8] Thomaas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition edition, 2009.

[9] Edgar Goodaire and Michael Parmenter. *Discrete Mathematics with Graph Theory*. Prentice Hall, 2nd edition, 2002.

[10] Michael Gourlay. Fluid simulation for video games. *Intel Software Network*, February 2010.

[11] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. Taylor & Francis, January 1989.

[12] T. Jakobsen. Advanced character physics. *Proc. Game Developer's Conf.*, 2001.

[13] Ales Krenek. Haptic rendering of molecular conformations, 2001.

[14] Audrey Lee. *Geometric constraint systems with applications in CAD and biology.* PhD thesis, University of Massachusetts Amherst, May 2008.

[15] Audrey Lee, Ileana Streinu, and Oliver Brock. A methodology for efficiently sampling the conformation space of molecular structures. *Physical Biology 2, SPECIAL FOCUS: Flexibility in biomolecules*, 2005.

[16] Audrey Lee-St John and Rittika Shamsuddin. The joint recognition problem: From cad constraints to kinematic joints, 2010.

[17] L.Ros, F. Thomas, J. M. Porta, C. Torras, V. Ruiz de Angulo, T. Creemers, J. Canto, F. Corcho, and A. Sabater. Geometric methods in robotics. *First Worksop on Automation, Vision and Robotics*, 2004.

[18] Prabhakar Raghavan and Rajeev Motwani. *Randomized Algorithms.* Cambridge University Press, 2000.

[19] J. M. Selig. *Geometrical Methods in Robotics.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

[20] John E. Stone, Justin Gullingsrud, and Klaus Schulten. A system for interactive molecular dynamics simulation. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, I3D '01, pages 191–194, New York, NY, USA, 2001. ACM.

[21] John E. Stone, David J. Hardy, Ivan S. Ufimtsev, and Klaus Schulten. Gpu-accelerated molecular modeling coming of age. *Journal of Molecular Graphics and Modelling*, 29(2):116 – 125, 2010.

[22] W. F. van Gunsteren, P. H. Hunenberger, A. E. Mark, P. E. Smith, and I. G. Tironi. Computer simulation of protein motion. *Computer Physics Communications*, 91(1-3):305–319, September 1995.

[23] Rachel Weinstein. *Simulation and Control of Articulated Rigid Bodies*. PhD thesis, Stanford University, 2007.

[24] S. Wells, S. Menor, B. M. Hespenheide, and M. Thorpe. Constrained geometric simulation of the diffusive motions in proteins. *Physical Biology*, 2, 2005.