

COMPUTATIONAL ANALYSIS OF STATICS AND DYNAMICS OF MACROMOLECULES

Milka Doktorova

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 2010

Acknowledgements

I would like to thank my thesis advisor, Professor Audrey St. John, for giving me a direction in life and for always encouraging me to do better.

I learned and accomplished a lot, thanks to your guidance.

I want to thank Professor Harriet Pollatsek for introducing me to the world of Lie groups and for helping me clarify certain concepts and definitions in the thesis.

I want to thank all my professors at the math and computer science departments at Mount Holyoke College for their continuous support.

I also want to extend a note of gratitude to Professor Ileana Streinu and the LinKaGe research group at University of Massachusetts Amherst for their help in generating some of the data for my project.

Last, but by no means least, I want to thank my family and friends for their unconditional care and love.

Abstract

Macromolecules, such as proteins, play a salient part in biological processes. Conducting analysis on the statics and dynamics of such structures is a key to understanding naturally occurring phenomena and designing drugs for the prevention and cure of diseases. This thesis presents research that applies tools from math and computer science to study two problems in computational biology: 1) the flexibility of proteins, and 2) the motion of macromolecular structures in general.

The first part of the thesis focuses on a phenomenon called the *allosteric effect*, which has been observed in nature, but is not well-understood. Proteins generally perform their functions by binding other molecules at designated locations called active sites. In some proteins, it has been found that small molecules can bind at a different *allosteric* location, thus causing a conformational (structural) change that affects the functionality of the protein. Gaining insight into how these conformational changes occur could lead to great advances in drug design. We take a computational approach by applying concepts from theoretical computer science, geometry and linear algebra to study the structural properties of macromolecules. We model proteins as discrete structures whose movements are restricted by specific geometric constraints. Then we apply techniques from rigidity theory to perform analysis on these constrained structures and display the results via an interactive web application.

Conformational changes are essential to macromolecular function, but actual motions cannot be observed experimentally. Standard simulation approaches, such as molecular dynamics, are too computationally expensive to perform on the timescale in which “interesting” motions occur, due to the complexity of the macromolecules. As a result, novel techniques are being developed to improve simulation efficiency. However, the implementation of such techniques is not standardized, so there is no effective method for comparing the performance of various approaches. The second part of this thesis applies standard software engineering principles to the design of an infrastructure that provides for a robust development and comparison of motion simulation techniques.

Contents

1	Introduction	1
1.1	Problem statements	3
1.2	Related work	4
1.2.1	Tools for static structural analysis	4
1.2.2	Tools for dynamic analysis	5
1.3	Contributions and results	6
1.4	Structure of thesis	7
2	Preliminaries	8
2.1	Definitions	8
2.2	Introduction to matrix Lie groups	9
2.2.1	Brief overview	10
2.2.2	The special Euclidean group	12
2.3	Introduction to rigidity: bar-and-joint structures	14
2.3.1	Algebraic rigidity	16
2.3.2	Infinitesimal rigidity	17
2.3.3	Combinatorial rigidity	20

3	Rigidity in 3D: Body-Bar-Hinge Structures	21
3.1	Algebraic rigidity	22
3.2	Infinitesimal rigidity	24
3.2.1	Theory of screws and infinitesimal rigid body motions	24
3.2.2	Grassmann-Cayley algebra and Plücker coordinates	27
3.2.3	Infinitesimal motions and bar constraints	30
3.2.4	3D body-and-hinge systems	31
3.3	Combinatorial rigidity	33
3.4	Contributions	36
4	Static Analysis for Understanding Allosteric Effect	39
4.1	Preliminaries	39
4.2	Overview of steps	41
4.3	Implementation details	43
4.4	Identifying relevant bodies	48
4.5	Identifying relevant null space vectors	49
4.6	Identifying relevant infinitesimal motions	54
4.7	Results	56
4.7.1	CHK1	56
4.7.2	PDK1	59
5	Infrastructure Design for Motion Simulation Software	70
5.1	Data representation	73
5.1.1	Geometric representation	74
5.1.2	Part and Assembly	76

5.1.3	Discussion	81
5.2	Motion simulation	83
5.2.1	SimulationStep and Motion Manager	83
5.2.2	StructureAnalyzer	85
5.2.3	PartMover	85
5.2.4	Resolver	86
5.2.5	MovieMaker	86
5.2.6	Discussion	87
6	Conclusions and Future Work	88
	Bibliography	90

List of Figures

1.1	Structure of an amino acid. Adapted from [12].	2
1.2	Serendipitous allosteric site identified in F16BPase (black atoms). Reprinted from [14].	4
2.1	A visual representation of a Lie group \mathcal{G} and its tangent space g	11
2.2	Examples of rigid and flexible frameworks	15
2.3	Different types of rigidity	15
2.4	Different embeddings of a framework	16
2.5	Example: a bar-and-joint framework and its corresponding rigid- ity matrix	19
3.1	Body-bar-hinge structure. Reprinted from [21].	21
3.2	Screws. Reprinted from [21].	26
3.3	Minimally rigid body-bar-hinge structure. Reprinted from [21]. . .	35
4.1	Steps of the analysis	41
4.2	Snapshot from the MotionSpace interface	46
4.3	Colors corresponding to the velocity of the atoms	46
4.4	Initialization stage	47
4.5	Interactive stage	47
4.6	Information for the active site of CHK1	49

4.7	Data from applying heuristic 1 to protein PDK1	55
4.8	(protein 3JVR; null space vector applied: 302) In this coloring, both the active and allosteric sites are flexible while the rest of the protein seems mostly rigid. Thus, we cannot make any conclusions about the relationship between the two sites.	64
4.9	(protein 3JVR; null space vector applied: 396) Here we can see again that the active and allosteric sites are flexible. However, most of the atoms in the rest of the protein are also flexible and thus, we cannot justify the existence of a relationship precisely between the two sites.	65
4.10	(protein 3JVR; null space vector applied: 6) In this coloring most of the protein is either rigid or slightly flexible. However, the magnitudes of the velocities of the atoms at the active and allosteric sites are larger, and, if we look carefully, we can identify a potential “path” that leads from one of the sites to the other, as illustrated in the bottom picture	66
4.11	(protein 3HRF; null space vector applied: 3) In this coloring most of the protein is rigid while some of the atoms at the active an allosteric sites have velocities different than 0. Thus, we cannot make any conclusions about the relationship between the two sites.	67
4.12	(protein 3HRF; null space vector applied: 337) Here we can see again that the active and allosteric sites are flexible. However, most of the atoms in the rest of the protein are also flexible and thus, we cannot justify the existence of a relationship precisely between the two sites.	68

4.13	(protein 3HRF; null space vector applied: 375) In this coloring most of the protein is either rigid or slightly flexible. However, the magnitudes of the velocities of the atoms at the active and allosteric sites are clearly larger, and we can identify a potential “path” that leads from one of the sites to the other. Note that the structure is 3-dimensional and the part of the path that seems to go through “blue” atoms actually goes behind them.	69
5.1	Primary and secondary structure of a protein. Adapted from http://commons.wikimedia.org/	72
5.2	Data representation for the infrastructure	73
5.3	GeometricRepresentation class and its children	74
5.4	GeometricElement class and its children	74
5.5	Point class and its children	75
5.6	Body class and its children	77
5.7	Structure of the Part class	78
5.8	Constraint class and its children	80
5.9	Motion Simulation part of the infrastructure	83
5.10	SimulationStep, Motion Manager and their children	84
5.11	Resolver	86

Chapter 1

Introduction

Proteins are the most adaptable macromolecules in living systems. Purposefully called the “building blocks of life,” they play an important role in essentially all biological processes. They catalyze reactions such as metabolism, DNA replication and digestion; they transport signals from one cell to another while controlling and coordinating activities throughout the organism; they provide mechanical support and immune protection, and are required for the building and repair of body tissues, muscle contractions, water balancing, nutrient transport and many other vital processes in the human body [25].

Protein versatility is the key factor that makes these macromolecules such an appealing object of research for many scientists. Understanding how proteins function provides a powerful tool for obtaining control over biological processes and thus, curing diseases.

Studying the structure of proteins is important for understanding how they function. In fact, there is a whole branch of bioinformatics, called structural bioinformatics, that examines precisely the structure-function relationship in macromolecules.

All proteins, whether very simple or extremely complex, are constructed from the same set of 20 amino acids. The structure of a prototypical amino acid is shown in Figure 1.1. The R-group distinguishes one amino acid from another and represents any of the 20 possible side chains, each with distinctive chemical characteristics. Amino acids form *peptide* bonds between each other.

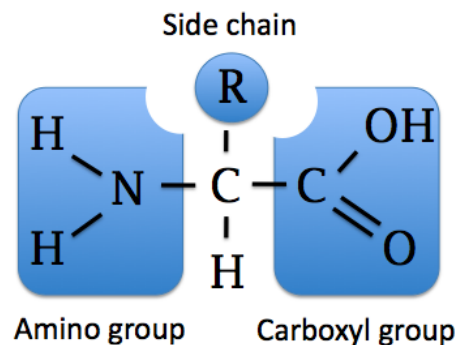


Figure 1.1: Structure of an amino acid.
Adapted from [12].

Multiple amino acids linked by such bonds are referred to as a *peptide*. Once incorporated into a peptide, the amino acid becomes an amino acid *residue*. Various combinations and sequences of amino acid residues produce proteins with different sizes and properties. Examples of such diverse protein products are enzymes, hormones, antibodies, transporters, muscle fibers, feathers, spider webs, milk proteins, mushroom poisons, antibiotics, etc. The most varied and specialized among them are the enzymes which serve as catalyzers of virtually all cellular reactions [25].

Proteins function by interacting with, or *binding*, other molecules (e.g., *ligands*). The locations where these interactions occur are called *active sites*. The binding process was initially thought to be a simple lock-and-key mechanism in which the lock is the active site and the key is the binding ligand. However, scientists now understand that this process is dynamic, involving changes in the *conformations* (i.e., the spatial arrangements of atoms in a molecule) of both the protein and the ligand.

1.1 Problem statements

Certain proteins also have *allosteric sites* that are different than, and usually far from, the active sites, but serve as binding docks for particular molecules. A phenomenon, called the *allosteric effect*, is observed when binding occurring at an allosteric site causes a conformational change at the active site, which may render the protein active or inhibit it from performing its function.

The allosteric effect is not well understood and therefore has become an active area of research. In particular, understanding how motion at the allosteric site causes motion at the active site is an open question. We will refer to this problem as **understanding the allosteric effect**.

Studying the mechanisms behind this phenomenon would give biologists valuable information that may help them to identify undiscovered allosteric sites in proteins. While a limited number of experimental approaches exist, **identifying allosteric sites** is still an open area of research. Providing tools towards this identification would foster advances in drug design.

Proteins are not static structures. Their flexibility is the key to their function. Conformational changes, essential to the proteins function, cannot be observed experimentally. Computational tools to help simulate these changes would give invaluable insight into how the “building blocks of life” work. Standard methods exist for biochemically accurate **motion simulation software**, but are too computationally expensive to simulate on the timescale in which “interesting” motions occur. This is due to the complexity of the protein molecules which are often composed of tens or hundreds of thousands of atoms.

1.2 Related work

In this section we present a brief overview of related work on tools for structural and dynamic analysis of proteins. The results described further in this thesis are based on rigidity theory which is discussed in Chapter 2 and Chapter 3. For a comprehensive history and review of rigidity theory, see [11].

1.2.1 Tools for static structural analysis

Different tools can be used to analyze the structure of proteins. One such tool is FIRST (Floppy Inclusions and Rigid Substructure Topography). FIRST is a software for analyzing the rigidity properties of biomolecular structures. Its first implementation was developed by Thorpe and Jacobs at Michigan State

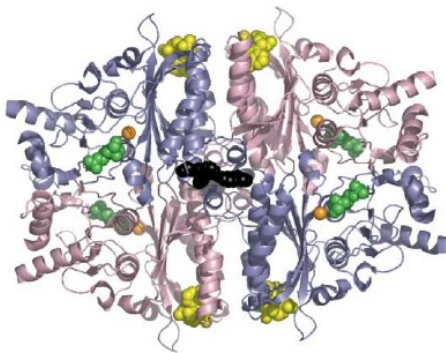


Figure 1.2: Serendipitous allosteric site identified in F16BPase (black atoms).

Reprinted from [14].

University and initially used a generalized “pebble game” approach for *bar-and-joint* structures (to be defined in Chapter 2.3) to identify the rigid and flexible parts of a molecule. A later version of FIRST based on a *body-and-hinge* model (to be described in Chapter 3.2.4) was developed at Arizona State University [2, 17, 18].

For fundamental literature on the naturally observed allosteric effect phenomenon, see [24]. Novel work by Hardy et al. [15] used computational techniques to assist in finding so-called *serendipitous allosteric sites* in pro-

teins. A serendipitous allosteric site is a site that “may not interact with any natural ligand and has only adventitiously been exploited by the small molecule that binds there.” The work of Hardy et al. revealed the existence of serendipitous allosteric sites in enzymes [14]. Their approach required the use of a software called HotPatch that identifies unusual patches, or cavities, on the surface of the protein and colors the macromolecule according to how “unusual” these patches are [3]. Based on the coloring, potential allosteric sites were first identified manually. Then chemical expertise was used to generate a list of potential ligands for each of these sites and finally, wet lab experiments were conducted to identify the ligands that successfully bound to a site (see Figure 1.2).

1.2.2 Tools for dynamic analysis

Motion simulation techniques have a long history [12]. The classical method of molecular dynamics (MD) is rooted in standard energy calculations, based on the chemical properties of particular atoms [31]. Some commonly used programs for MD simulation are AMBER, CHARMM, ENCAD, and GRO-MOS. While considered the most biologically accurate technique, current MD approaches can simulate motion up to the nanosecond timescale. Since functional motions may occur at the microsecond timescale, MD is simply too computationally expensive to produce precise simulations [12].

Other techniques for analyzing the flexibility of proteins include *normal mode analysis* (NMA) [1, 28] and *elastic network models* (ENM) [8] such as the *Gaussian network model* (GNM) [13]. They study mostly the fluctuation dynamics of the proteins near their native state conformations. NMA constructs a harmonic approximation of the potential well around an energy

minimized conformation and involves Hessian computation. GNM is based on polymer network mechanics and models each protein by an elastic network.

The Yale morph server [10], another simulation tool, performs a linear interpolation between initial and final protein conformations, generating a “movie” of the motion; however, such movies are biologically inaccurate as they do not respect chemical constraints during the interpolated motion. Other computational techniques include FRODA [35] and ROCK [23], which generate new conformations for complex networks while maintaining all constraints.

1.3 Contributions and results

The first part of the thesis addresses the problem of **understanding the allosteric effect**. We approach it by modeling proteins as discrete structures whose movements are restricted by geometric constraints. We analyze the rigidity properties of the model and apply various computational tools to explore its infinitesimal motion space. Along the way, we present an approach for properly **representing infinitesimal hinge constraints**. Our results and analysis give an intuition about the mechanisms of the allosteric effect and provide a strong foundation for future research in this area.

The second part of the thesis focuses on the development of **motion simulation software**. We apply standard software engineering principles to design an infrastructure that provides for a robust development and comparison of motion simulation techniques. The future implementation of this infrastructure would not only provide an effective method for comparing the performance of various approaches, but could also foster the development of new motion simulation techniques and improve the efficiency of already existing ones.

1.4 Structure of thesis

The subsequent chapters describe the theory behind the analysis and provide a discussion of the results. Chapters 2, 3 and 4 focus on the problem of **understanding the allosteric effect**. Chapter 2 gives the preliminaries of the work, introducing basic notation and terminology (Section 2.1), the theory of matrix Lie groups (Section 2.2), and the concepts of algebraic, infinitesimal, and combinatorial rigidity for 2-dimensional bar-and-joint structures (Section 2.3). Chapter 3 builds upon the basics from Chapter 2 and discusses the development of the rigidity theory for body-and-hinge frameworks, then presents a new contribution for handling special cases of **hinge axes**. Chapter 4 applies the theory from Chapter 3 to perform static analysis on proteins: it begins with preliminaries and methodology (Sections 4.1 and 4.2), then presents the implementation details and data collection process (4.3 through 4.6), and ends with a discussion of the results for two protein structures (4.7).

Chapter 5 addresses the problem of the **development of motion simulation software** by introducing the design of an infrastructure for application and comparison of motion simulation techniques. Section 5.1 presents the *data representation* part of the infrastructure: it describes the organization of the main classes in the library, and discusses the benefits and limitations of the design decisions. Similarly, Section 5.2 focuses on the *motion simulation* part of the infrastructure by first introducing, then discussing, its design.

Finally, Chapter 6 presents conclusions of the thesis results and future directions for research.

Chapter 2

Preliminaries

Proteins can be abstractly represented as a set of objects (atoms) with certain constraints (chemical bonds) between them. Such representation is analogous to a *graph* structure whose rigidity properties can further be analyzed. We begin by defining the basic combinatorial objects that will be used throughout the paper, then give an overview of the theory of Lie groups and use it to introduce the concept of rigidity in the context of *bar-and-joint structures*.

2.1 Definitions

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. Depending on the set of edges, a graph can be a *simple graph* (a graph that does not have any loops or parallel edges) or a *multigraph* (a graph that is permitted to have parallel edges).

A graph $G = (V, E)$ along with a distance function on the edges $L : E \rightarrow \mathbb{R}$, defines the *framework* (G, L) . A framework describes a *bar-and-joint* structure,

composed of universal joints connected by fixed-length bar constraints. We use bar-and-joint structures in Section 2.3 to introduce the concept of rigidity.

The theoretical analysis of a framework is related to the inspection of whether the framework can be realized in Euclidean space. This problem is called the **realization problem**. We use \mathbb{R}^d to denote d -dimensional Euclidean space. In mathematical terms, $\mathbb{R}^d = \{(a_1, \dots, a_d) : a_j \in \mathbb{R}\}$. The realization of a framework (G, L) in \mathbb{R}^d is an assignment $P = (p_1, p_2, \dots, p_n)$ of $n = |V|$ distinct points in \mathbb{R}^d . The points assign coordinates to the vertices of V and are constrained by the edge lengths defined by L . Realizing a framework may also be viewed as assigning a single point in (dn) -dimensional space. The subspace of \mathbb{R}^{dn} comprising all possible realizations of a framework forms the *configuration*, or *motion*, space of the framework.

2.2 Introduction to matrix Lie groups

Before we explore the rigidity properties of a framework, we first introduce the theory of Lie groups. Formally speaking, Lie groups have the structure of *differentiable manifolds* with associated *Lie algebras*, defined as tangent spaces. These are very complicated structures and studying them carefully is outside the scope of this thesis. We will note, however, that when a matrix group is a closed subset of Euclidean space, we can do calculus on the group the same way we do calculus on Euclidean space. For the purpose of this research, we restrict our attention only to matrix Lie groups which “live” entirely in Euclidean space and are thus “well-behaved”: they satisfy all conditions of being Lie groups, we can do calculus on them, and their tangent spaces at the identity are Lie algebras. It is important to note that not all matrix groups

are Lie groups, but those that usually appear in various scientific applications, are.

This section gives a very brief overview of matrix Lie groups and their tangent spaces, then describes the special Euclidean group and its dimension as both play a fundamental role in the rigidity analysis. We refer the interested reader to the classroom-adapted book written by Pollatsek [27] on Lie groups.

2.2.1 Brief overview

As we mentioned, the Lie groups that we will explore are matrix groups on which we can do calculus. To illustrate what exactly we mean by this, suppose that we have a positive integer n , and \mathcal{G} is a group of $n \times n$ matrices with real entries:

$$\mathcal{G} = \left\{ \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} : a_{ij} \in \mathbb{R} \right\}.$$

Now let α be a function that maps real numbers to matrices in \mathcal{G} , i.e., $\alpha : \mathbb{R} \rightarrow \mathcal{G}$

$$\alpha(t) = \begin{bmatrix} a_{11}(t) & \cdots & a_{1n}(t) \\ \vdots & \ddots & \vdots \\ a_{n1}(t) & \cdots & a_{nn}(t) \end{bmatrix} \in \mathcal{G}.$$

We can intuitively think of each entry $a_{ij}(t)$ in $\alpha(t)$ as a function that changes over time ($a_{ij} : \mathbb{R} \rightarrow \mathbb{R}$). Thus, if we differentiate α with respect to t we get

$$\frac{d}{dt}(\alpha) = \begin{bmatrix} a_{11}'(t) & \cdots & a_{1n}'(t) \\ \vdots & \ddots & \vdots \\ a_{n1}'(t) & \cdots & a_{nn}'(t) \end{bmatrix} \quad \text{where} \quad a_{ij}'(t) = \frac{d}{dt}(a_{ij}).$$

If all entries of $\alpha(t)$ are smooth, or equivalently, infinitely differentiable, functions (i.e., their first, second, third, ..., n^{th} , ... derivatives with respect to t all exist), then we say that α is a *smooth curve* in \mathcal{G} . Moreover, if $\alpha(0) = I_n$,

$$\alpha(0) = \begin{bmatrix} a_{11}(0) & \cdots & a_{1n}(0) \\ \vdots & & \vdots \\ a_{n1}(0) & \cdots & a_{nn}(0) \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix},$$

then, we say that α *passes through* the identity of \mathcal{G} . We know that \mathcal{G} contains the identity, because it is a group, and thus we can visualize $\alpha(t)$ as a curve in the space, that passes through I_n as shown in Figure 2.1. It is easy to see that there can actually be many other smooth curves that pass through the identity (one example being the dotted curve on Figure 2.1), and all of them are in the neighborhood of I_n .

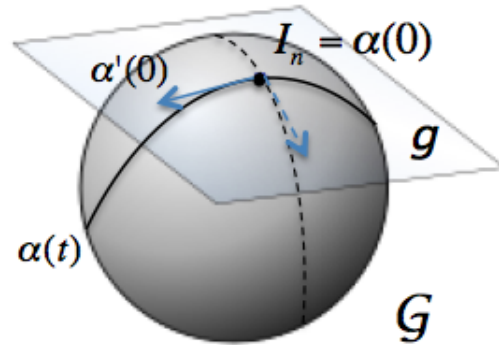


Figure 2.1: A visual representation of a Lie group \mathcal{G} and its tangent space \mathfrak{g}

As is the case with curves in n -dimensional space, we can examine the vector tangent to $\alpha(t)$ at the identity, namely $\alpha'(0)$. If we do the same for all other smooth curves passing through the identity in \mathcal{G} , we will get a vector space called the **tangent space at the identity** of \mathcal{G} , denoted by g , as shown in Figure 2.1. Thus, we can formally define g as:

$$g = \{ \alpha'(0) \mid \alpha : \mathbb{R} \rightarrow \mathcal{G}, \alpha(0) = I_n, \alpha \text{ smooth} \}.$$

It turns out that the dimension of the tangent space at the identity of a group defines the dimension of the group itself.

2.2.2 The special Euclidean group

The rigidity analysis that we describe later relies on the **special Euclidean group** which belongs to the larger class of matrix Lie groups. Before we define it, however, we first introduce the *orthogonal* and *special orthogonal groups*.

The *orthogonal group* is the group of all $n \times n$ orthogonal matrices whose transpose is equal to their inverse:

$$O(n) = \{ M \in \mathcal{M}(n) : M^T M = I_n \}$$

where $\mathcal{M}(n)$ denotes the set of all $n \times n$ matrices. The elements of the group have the property of preserving distances between points. In 2-dimensional space, for instance, they represent all rotations and reflections in the x - y plane.

The *special orthogonal group*, $SO(n)$, is a subgroup of the orthogonal group in which all matrices have determinant 1:

$$SO(n) = \{ M \in O(n) : \det(M) = 1 \}.$$

The elements of $SO(2)$ represent all rotations in the x - y plane.

The tangent space at the identity of $SO(n)$, denoted by $so(n)$, is a space of *skew-symmetric*, or equivalently, *antisymmetric* matrices of the form

$$A = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ -a_{12} & 0 & & \vdots \\ \vdots & & \ddots & a_{(n-1)n} \\ -a_{1n} & \cdots & -a_{(n-1)n} & 0 \end{bmatrix}.$$

Thus, we can define $so(n)$ as

$$so(n) = \{A \in \mathcal{M}(n) : A^T + A = \mathbf{0}_n\}$$

where $\mathbf{0}_n$ is the $n \times n$ matrix of all zeros. Note that, due to the structure of its skew-symmetric elements, the dimension of $so(n)$ is

$$\dim(so(n)) = \frac{n^2 - n}{2} = \frac{n(n-1)}{2}.$$

The **special Euclidean group**, denoted by $SE(n)$, is the group of all $(n+1) \times (n+1)$ matrices of the form

$$\left(\begin{array}{c|c} R & \mathbf{v} \\ \hline \mathbf{0} & 1 \end{array} \right)$$

where R is an $n \times n$ rotation matrix, $\mathbf{v} \in \mathbb{R}^n$ is a translation vector, and $\mathbf{0}$ is a $1 \times n$ vector of zeros. A curve in $SE(n)$ that passes through the identity of the group can be written as

$$\alpha(t) = \left\{ \begin{bmatrix} \gamma(t) & \mathbf{v}(t) \\ \mathbf{0} & 1 \end{bmatrix} : \gamma(t) \in SO(n), \mathbf{v}(t) \in \mathbb{R}^n \right\}$$

where $\gamma(0) = I_n$ and $\mathbf{v}(0)$ is the zero vector in \mathbb{R}^n . Thus, the tangent space at the identity of $\text{SE}(n)$, denoted by $\text{se}(n)$, has the form:

$$\text{se}(n) = \left\{ \begin{bmatrix} \gamma'(0) & \mathbf{v}'(0) \\ \mathbf{0} & 0 \end{bmatrix} : \gamma'(0) \in \text{so}(n), \mathbf{v}'(0) \in \mathbb{R}^n \right\}.$$

Since $\gamma'(0)$ is an arbitrary element in $\text{so}(n)$ and $\mathbf{v}'(0)$ is an arbitrary element in \mathbb{R}^n , the dimension of $\text{se}(n)$ becomes

$$\dim(\text{se}(n)) = \dim(\text{so}(n)) + \dim(\mathbb{R}^n) = \frac{n(n-1)}{2} + n = \frac{n(n+1)}{2}.$$

2.3 Introduction to rigidity: bar-and-joint structures

Rigidity theory studies the possible deformations (or motion space) that a framework may have while preserving its constraints. Analysis may categorize a framework to be rigid, flexible or flexible with a rigid component, as shown in Figure 2.2. We begin by studying *bar-and-joint* structures, composed of *universal joints* (illustrated by the green points in the figure) with distance constraints between them (the purple bars connecting the points). While we cannot deform a triangular structure as in 2.2(a) without changing the distances between the joints, we can “flex” a rectangular framework while maintaining all of its constraints, as in 2.2(b). In this case, we say that the structure has 1 *degree of freedom* (intuitively, a degree of freedom can be associated with a flex in the framework’s motion space). Therefore the structure in 2.2(a) is said to be *rigid*, and the structure in 2.2(b) *flexible*.

It is possible, however, to have a flexible structure as in Figure 2.2(c) with

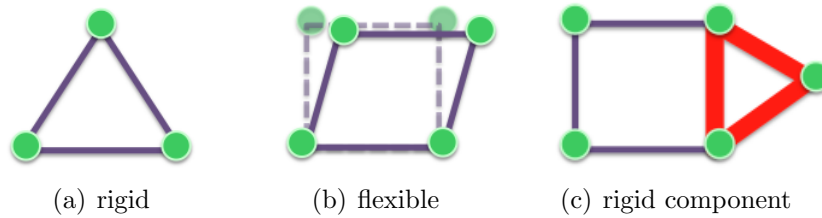


Figure 2.2: Examples of rigid and flexible frameworks

a rigid component (illustrated by the thicker bars). In this case, when the flexible part undergoes deformations, the rigid part “moves” along with it, but stays locally rigid.

Rigid frameworks can further be classified as either *minimally rigid* or *overconstrained*. A minimally rigid framework is one such that, if we remove any of its bars, the framework becomes flexible (Figure 2.3(a)). On the other hand, an overconstrained framework contains at least one bar whose removal preserves the rigidity of the structure (Figure 2.3(b)). In fact, any bar could be removed from 2.3(b) without the structure becoming flexible.

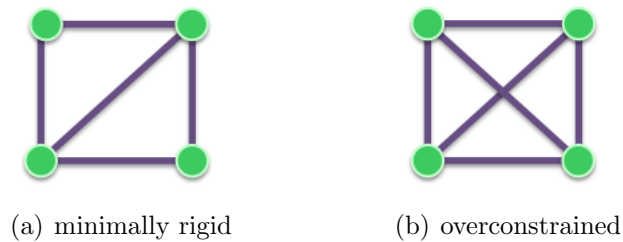


Figure 2.3: Different types of rigidity

The development of the rigidity theory for a model can be broken into three general steps: *algebraic*, *infinitesimal*, and *combinatorial*. The rest of this chapter presents brief descriptions of each as they pertain to the bar-and-joint model.

2.3.1 Algebraic rigidity

Algebraic rigidity is closely related to the **realization problem** which asks if a framework can be embedded in d -dimensional space. In other words, we are interested in whether such an embedding exists in \mathbb{R}^d or not. In \mathbb{R}^2 , for instance, each point p_i has two coordinates (x_i, y_i) . If e is an edge such that $e = ij \in E$, and L_{ij} is the length of edge e , then the realization problem is equivalent to finding a solution to the system consisting of $|E|$ equations of the following form:

$$(x_i - x_j)^2 + (y_i - y_j)^2 = L_{ij}^2.$$

Every framework has *trivial* motions, i.e., rotations and translations. A trivial motion preserves the distances between any two points, whether they are adjacent or not. Intuitively, a framework is considered *rigid* if it cannot be deformed (modulo the trivial motions); otherwise, it is *flexible*.

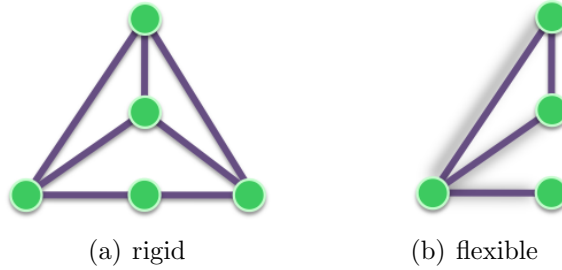


Figure 2.4: Different embeddings of a framework

Note, however, that this intuitive notion of rigidity is associated with a particular embedding in space satisfying the distance function on the edges L . Consider, for instance, the two embeddings of the structure in Figure 2.4. The one in 2.4(a) consists only of triangles and is thus, rigid. However, if we “flip” the right half of the structure on top of the left half (so that the bottom

rightmost and leftmost joints overlap as in 2.4(b)), the structure becomes flexible.

From now on we abuse terminology by assuming that a framework is always given with a particular embedding in space. We will then study rigidity properties for that embedding.

2.3.2 Infinitesimal rigidity

Solving systems of algebraic equations quickly becomes computationally infeasible. Therefore, we consider the *infinitesimal* rigidity of the framework. Infinitesimal rigidity defines a framework to be rigid or flexible depending on the type of its infinitesimal motion and can be viewed as an approximation for algebraic rigidity, as infinitesimal rigidity implies algebraic rigidity, while the opposite is not always true [7].

Let $P = (p_1, p_2, \dots, p_n)$ be the embedding of a framework (G, L) consisting of n vertices in d space and constrained by the edge lengths L . Then, for every edge $e = ij \in E$, we have:

$$\langle (p_i - p_j), (p_i - p_j) \rangle = L_{ij}^2$$

where $\langle (p_i - p_j), (p_i - p_j) \rangle$ denotes the dot product of the vector $p_i - p_j$ with itself. The motion of the framework can be represented as a continuous curve through the configuration space. Hence, at any particular moment t , P can be written as $P(t) = (p_1(t), p_2(t), \dots, p_n(t))$. Thus, a more general form of the equation above is:

$$\langle p_i(t) - p_j(t), p_i(t) - p_j(t) \rangle = L_{ij}^2.$$

Since the product rule generalizes to dot products, assuming that the motion is differentiable, we obtain

$$\left\langle p_i(t) - p_j(t), \frac{d}{dt}(p_i(t) - p_j(t)) \right\rangle = \langle p_i(t) - p_j(t), p'_i(t) - p'_j(t) \rangle = 0.$$

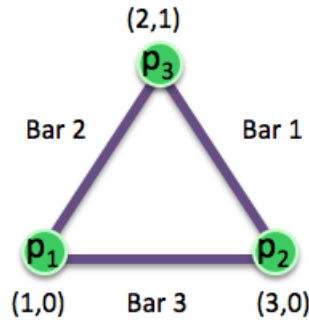
Note that the first derivative of a function modeling a structure's motion assigns instantaneous, or *infinitesimal*, velocities to the points. The first derivative $p'(t)$ of the configuration path of an embedding p , i.e., $p(t)$, at any given moment t , represents the *infinitesimal velocity vector* of the framework at t . An infinitesimal velocity vector is a vector $\mathbf{v} = (v_1, v_2, \dots, v_n)$ with each $v_{i \in [1, n]} \in \mathbb{R}^d$, whose components satisfy the equation:

$$\langle p_i(t) - p_j(t), v_i(t) - v_j(t) \rangle = 0 \quad \text{for all } i, j \in E. \quad (1)$$

If the only infinitesimal motions correspond to the trivial ones (translations and rotations), the framework is *infinitesimally rigid*; otherwise, it is *infinitesimally flexible*.

Rigidity matrix

The system of linear equations defined by (1) can be represented by a matrix M with $|E|$ rows and $d|V|$ columns (one column for each vertex coordinate) where $M\mathbf{v} = 0$ must hold for all velocity vectors \mathbf{v} . For every edge $ij \in E$, there is a row in M in which the d column entries associated with $v_i(t)$ are equal to the d coordinates of the difference $p_i(t) - p_j(t)$, and the d column entries associated with $v_j(t)$ are equal to the d coordinates of the difference $p_j(t) - p_i(t)$. This matrix is called the *rigidity matrix* for bar-and-joint structures. A sample structure embedded in space and its rigidity matrix are shown below.



	p_1		p_2		p_3	
	x_1	y_1	x_2	y_2	x_3	y_3
Bar 1	0	0	1	-1	-1	1
Bar 2	-1	-1	0	0	1	1
Bar 3	-2	0	2	0	0	0

Rigidity matrix

Figure 2.5: Example: a bar-and-joint framework and its corresponding rigidity matrix

The space of *infinitesimal motions* of a framework forms the kernel of the framework's rigidity matrix. Note that the trivial motions of the framework must then be in the kernel. The trivial motions correspond to the rigid transformations of the system, which in turn, correspond to the elements of the Lie algebra $se(d)$. Recall from Section 2.2.2 that $se(d)$ has dimension $d(d+1)/2$. If the dimension of the kernel is exactly $d(d+1)/2$, then the only possible transformations of the system are the trivial ones and thus, the framework is infinitesimally rigid; if the dimension of the kernel is greater than $d(d+1)/2$, then there is a non-trivial infinitesimal motion, or *flex*, and the framework is infinitesimally flexible.

In the example from Figure 2.3.2 the kernel of the rigidity matrix (denoted by $\ker(A)$, where A is a matrix) is:

$$\ker \left(\begin{bmatrix} 0 & 0 & 1 & -1 & -1 & 1 \\ -1 & -1 & 0 & 0 & 1 & 1 \\ -2 & 0 & 2 & 0 & 0 & 0 \end{bmatrix} \right) = \left\{ \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ 2 \\ -1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right\}.$$

The dimension of the kernel is $3 = 2(2 + 1)/2$ and corresponds to the trivial motions of the framework in \mathbb{R}^2 (i.e., translations along the x and y axes, and rotation). Thus, the structure is infinitesimally rigid.

2.3.3 Combinatorial rigidity

In certain models, the pattern of the rigidity matrix may lead to a combinatorial characterization for infinitesimal rigidity. For 2D bar-and-joint rigidity, Laman proved the following theorem.

Theorem 2.3.1. (*Laman's theorem*) *A graph is generically¹ minimally rigid as a 2D bar-and-joint framework if and only if it has $2n - 3$ edges and any subset of n' vertices spans at most $2n' - 3$ edges.*

This theorem leads to combinatorial algorithms for decomposing a given graph into rigid components as well as computing the degrees of freedom for the graph. In particular, Jacobs and Hendrickson developed the *2D pebble game*. Although the hereditary count condition appears to require exponential time, the pebble game is an efficient and simple algorithm that runs in $O(n^2)$ time [16].

While 2D bar-and-joint is thus well-understood, a combinatorial characterization for 3D bar-and-joint rigidity is arguably the biggest open question in rigidity theory.

¹A framework is *generic* if the associated rigidity matrix achieves its highest rank over all embeddings.

Chapter 3

Rigidity in 3D: Body-Bar-Hinge Structures

Proteins can be modeled as 3D bar-and-joint structures, but there are no known efficient algorithms for analyzing the rigidity of that model. In this chapter, we consider a different model of rigidity for *body-bar-hinge* structures that can be used instead to model proteins in 3-dimensional space. A body-bar-hinge structure consists of rigid bodies and constraints between them, as shown in Figure 3.1. The constraints can be in the form of *bars* and *hinges*.

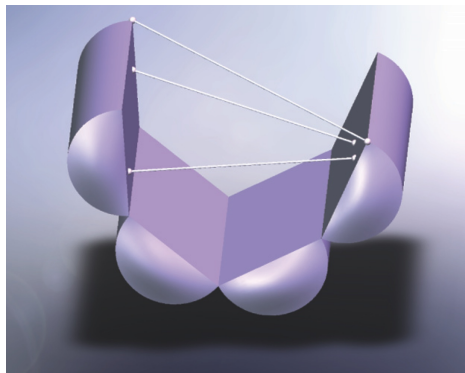


Figure 3.1: Body-bar-hinge structure. Reprinted from [21].

A bar imposes a distance constraint between two bodies and is attached to them by universal joints positioned at specified locations, while a hinge allows only a single rotational degree of freedom (for more details, see Section 3.2.4). If the motion space of a body-bar-hinge structure consists solely of the trivial motions (i.e., rotations and translations), the structure is said to be *rigid*; otherwise, it is *flexible*.

The development of the rigidity theory in 3D can again be broken into three steps. The foundation was presented in [32, 38] (see also [21] for a complete development including the algebraic theory). In Sections 3.1-3.3 we give an overview of the rigidity theory for first, *body-and-bar*, then, *body-bar-hinge* structures in 3-space. The last section, 3.4, presents a new contribution for handling special cases of hinge axes.

3.1 Algebraic rigidity

Let $G=(V,E)$ be a multigraph with n vertices and m edges, where each vertex represents a body and each edge represents a bar. Then, let \mathbf{p} and \mathbf{q} be two m -tuples of points in \mathbb{R}^3 defining the positions of each bar's two attachment universal joints. Thus, a bar e between two bodies would attach to each body at the points $\mathbf{p}_e \in \mathbf{p}$ and $\mathbf{q}_e \in \mathbf{q}$. If $L : E \rightarrow \mathbb{R}$ is the distance function for each bar, the framework $(G, \mathbf{p}, \mathbf{q}, L)$ describes a body-and-bar structure.

We begin studying the algebraic rigidity properties by considering the **realization problem**. The realization problem asks whether the framework $(G, \mathbf{p}, \mathbf{q}, L)$ has an *embedding* satisfying L . However, since the vertices represent bodies, the term *embedding* becomes more subtle. In order to define it precisely, we first need to clarify the notion of a *body*.

A body is an abstract structure defined by a frame and is represented by a *transformation* matrix. Such transformation matrices in 3D are elements of the special Euclidean group $SE(3)$ (described in Section 2.2). Each element of $SE(3)$ is a 4×4 matrix having the form:

$$\left(\begin{array}{c|c} \mathbf{R} & \mathbf{c} \\ \hline \mathbf{0} & 1 \end{array} \right)$$

where \mathbf{R} is a 3×3 rotation matrix; \mathbf{c} is a translation vector in \mathbb{R}^3 ; and $\mathbf{0}$ is a 1×3 vector of zeros. Applying this matrix to a point is equivalent to rotating the point by the matrix \mathbf{R} , and translating it by the vector \mathbf{c} .

Now let $\mathbf{T} \in (SE(3))^n$ be an assignment of frames to all n bodies in V , with \mathbf{T}_u being the transformation matrix for the frame assigned to body u . Consider a bar $e = uv \in E$ with attachment points $\mathbf{p}_e \in \mathbb{R}^3$ and $\mathbf{q}_e \in \mathbb{R}^3$ in bodies u and v , respectively. Then, solving the realization problem becomes equivalent to solving the quadratic system of equations, in which every bar is described by the following equation:

$$\left\| \mathbf{T}_u \begin{pmatrix} \mathbf{p}_e \\ 1 \end{pmatrix} - \mathbf{T}_v \begin{pmatrix} \mathbf{q}_e \\ 1 \end{pmatrix} \right\|^2 = (L(e))^2 \quad (3.1)$$

The notation $\begin{pmatrix} \mathbf{p}_e \\ 1 \end{pmatrix}$ is used to represent the *homogenous coordinates* of the point \mathbf{p}_e which simply add an additional dimension to the point without changing its position and geometric properties (homogenous coordinates are necessary since all \mathbf{T} matrices are 4×4 and all points are in \mathbb{R}^3).

Recall our assumption that a framework is always given with an embedding satisfying L . Then, the framework $(G, \mathbf{p}, \mathbf{q}, L)$ is considered rigid if it cannot be deformed (modulo the trivial motions); otherwise, it is flexible.

3.2 Infinitesimal rigidity

The infinitesimal theory for body-and-bar frameworks is a study of the deformations of these frameworks over time. If $\mathbf{T} \in (\text{SE}(3))^n$ is the set of transformation matrices defining the bodies in a framework, then $\mathbf{T}(t)$ represents the set of deformations of the structure over time. In mathematical terms, $\mathbf{T}(t)$ can be viewed as the map $\mathbf{T} : \mathcal{I} \rightarrow (\text{SE}(3))^n$, where \mathcal{I} is a time interval containing zero. Without loss of generality, assume that $\mathbf{T}_i(0) = I_4$ (the identity of $\text{SE}(3)$) for all $i = 1, \dots, n$. If we differentiate Equation 3.1, we obtain:

$$\left\langle \mathbf{T}_u(t) \begin{pmatrix} \mathbf{p}_e \\ 1 \end{pmatrix} - \mathbf{T}_v(t) \begin{pmatrix} \mathbf{q}_e \\ 1 \end{pmatrix}, \mathbf{T}'_u(t) \begin{pmatrix} \mathbf{p}_e \\ 1 \end{pmatrix} - \mathbf{T}'_v(t) \begin{pmatrix} \mathbf{q}_e \\ 1 \end{pmatrix} \right\rangle = 0.$$

The first order derivative $\mathbf{T}'(t)$ represents the *infinitesimal motions* for the bodies at time t . Since, for each body u , $T'_u(0) \in se(3)$ (the space tangent to the identity of $\text{SE}(3)$), an infinitesimal motion for a body is an element of $se(3)$. Thus, $\mathbf{T}'(t) \in (se(3))^n$. In order to further study the infinitesimal motions for a body-and-bar framework, we introduce rigid body motions in 3D.

3.2.1 Theory of screws and infinitesimal rigid body motions

The theory of screws was first introduced by Sir Robert Ball to describe rigid body motion [9]. Intuitively, a *screw* is defined by an axis, called the *screw axis*, and represents a displacement about that axis as illustrated in Figure 3.2(a). As a consequence of Chasles' Theorem, the elements of the Lie algebra

$se(3)$ can be identified exactly with Ball's instantaneous screws (Figure 3.2(b)). Here we present only a brief overview of the connection between them. A more thorough development can be found in [20, 21, 29].

Elements of $se(3)$

As we described in Section 2.2.2, $se(3)$ can be defined mathematically as:

$$se(3) = \left\{ \begin{bmatrix} \Omega & \mathbf{v} \\ \mathbf{0} & 0 \end{bmatrix} : \Omega \in \mathbb{R}^{3 \times 3}, \mathbf{v} \in \mathbb{R}^3, \Omega^T + \Omega = \mathbf{0}_3 \right\}$$

where Ω is an element of $so(3)$. Since Ω is a skew-symmetric matrix, we can write it as:

$$\Omega = \begin{pmatrix} 0 & -\omega^z & \omega^y \\ \omega^z & 0 & -\omega^x \\ -\omega^y & \omega^x & 0 \end{pmatrix}$$

such that, for any vector $\mathbf{x} \in \mathbb{R}^3$, $\Omega\mathbf{x} = \boldsymbol{\omega} \times \mathbf{x}$ where $\boldsymbol{\omega} = (\omega^x, \omega^y, \omega^z) \in \mathbb{R}^3$ and $\boldsymbol{\omega} \times \mathbf{x}$ denotes the cross product of $\boldsymbol{\omega}$ and \mathbf{x} .

Instantaneous screws

Let \mathbf{p} be a point in \mathbb{R}^3 and \mathbf{p}' its velocity resulting from an instantaneous screw. Then, if \mathbf{a} is a point on the axis of rotation of the screw, \mathbf{p}' can be written as

$$\mathbf{p}' = \Omega(\mathbf{p} - \mathbf{a}) + \tau \tag{3.2}$$

where τ is the translational velocity component of the screw and $\begin{bmatrix} \Omega & \tau \\ \mathbf{0} & 0 \end{bmatrix}$ is an element of $se(3)$. Thus, $\boldsymbol{\omega}$ represents the *angular velocity vector* of the

screw defining the direction of the rotation axis and the rotational speed.

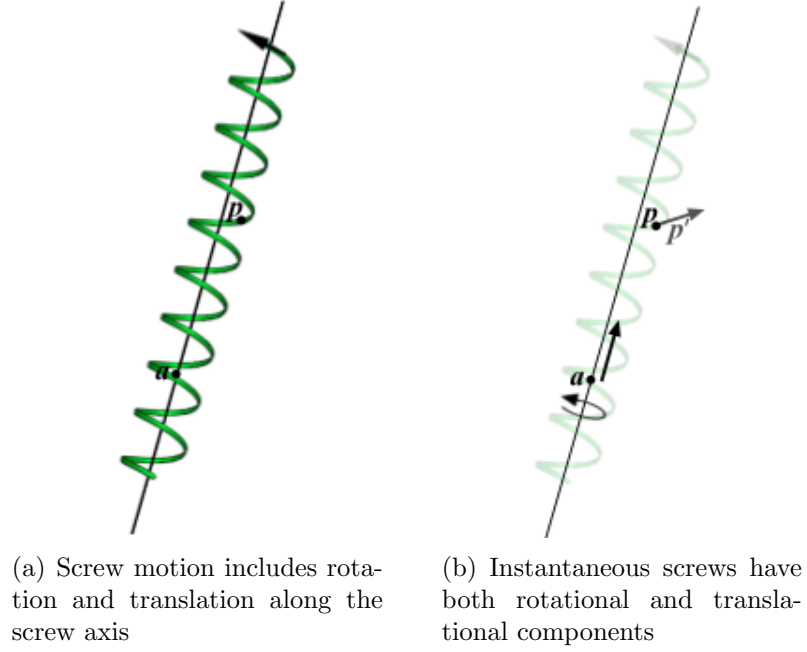


Figure 3.2: Screws. Reprinted from [21].

Now let $\mathbf{v} = \mathbf{a} \times \boldsymbol{\omega} + \tau$. Then, the velocity becomes:

$$\begin{aligned}
 \mathbf{p}' &= \boldsymbol{\Omega}(\mathbf{p} - \mathbf{a}) + \tau \\
 &= \boldsymbol{\Omega}\mathbf{p} - \boldsymbol{\Omega}\mathbf{a} + \tau \\
 &= \boldsymbol{\omega} \times \mathbf{p} - \boldsymbol{\omega} \times \mathbf{a} + \tau \\
 &= \boldsymbol{\omega} \times \mathbf{p} + \mathbf{a} \times \boldsymbol{\omega} + \tau \\
 \mathbf{p}' &= \boldsymbol{\omega} \times \mathbf{p} + \mathbf{v}
 \end{aligned} \tag{3.3}$$

This result allows us to associate the instantaneous screw with the 6-vector

$$(-\boldsymbol{\omega}, \mathbf{v})^1 = (-\omega^x, -\omega^y, -\omega^z, v^x, v^y, v^z).$$

¹The negative sign in front of $\boldsymbol{\omega}$ is necessary for the interpretation of the result from the *join* (in the Grassmann-Cayley algebra) of the 6-vector with a point as the instantaneous velocity of the point

Hence, every element $s \in \mathbb{R}^6$ can be identified with an instantaneous screw by interpreting its coordinates to be $(-\boldsymbol{\omega}, \mathbf{v})$.

3.2.2 Grassmann-Cayley algebra and Plücker coordinates

Grassmann-Cayley algebra is defined by a *bracket ring*, where a bracket is a $k \times k$ determinant of the homogenous coordinate vectors of k points in a $(k - 1)$ -dimensional projective space. This algebra is required to develop infinitesimal rigidity theory and here we present only a brief overview of some basic concepts. For a detailed explanation, see [36, 37].

Grassmann-Cayley algebra has two operators - *join* and *meet*. They can be intuitively thought of as “union” and “intersection” respectively. For example, the join of a point and a line not containing the point is the plane spanned by them, and the meet of two intersecting non-identical lines is a point.

Extensors

An *extensor* of step k , also called a k -extensor, is defined by k linearly independent points a_1, \dots, a_k , and denoted by $a_1 \vee \dots \vee a_k$, where the symbol \vee represents the Grassmann-Cayley join operator. We can compute the Plücker coordinates of the subspace spanned by the k points of an extensor by calculating all $k \times k$ minors (i.e., determinants) of the transpose of the matrix whose columns are the homogenous coordinates of the k points (see example below). Note that the subspace spanned by the k points of an extensor (each point being in \mathbb{R}^d) is a k -dimensional linear subspace of \mathbb{R}^{d+1} that can be identified with a $(k-1)$ -dimensional projective subspace of \mathbb{P}^d , where \mathbb{P}^d denotes

d -dimensional projective space (for instance, a 1-dimensional subspace of \mathbb{R}^3 corresponds to a point in \mathbb{P}^2 , and a 2-dimensional subspace of \mathbb{R}^3 corresponds to a line in \mathbb{P}^2). Thus, each k -extensor is identified with a linear subspace of dimension $k - 1$.

Plücker coordinates of lines in 3D

Let $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$ be two points in 3D with coordinates (a^x, a^y, a^z) and (b^x, b^y, b^z) respectively. To compute their join $\mathbf{a} \vee \mathbf{b}$ (whose geometric representation is a line), we form the 2×4 matrix M :

$$M = \begin{pmatrix} a^x & a^y & a^z & 1 \\ b^x & b^y & b^z & 1 \end{pmatrix}$$

The Plücker coordinates of the line determined by \mathbf{a} and \mathbf{b} are the six 2×2 minors of M . Since there are various ways of writing down the 6 minors, we will use the following convention $(M_{14}, M_{24}, M_{34}, M_{23}, -M_{13}, M_{12})$. In the example above, this is equivalent to:

$$\left(\begin{vmatrix} a^x & 1 \\ b^x & 1 \end{vmatrix}, \begin{vmatrix} a^y & 1 \\ b^y & 1 \end{vmatrix}, \begin{vmatrix} a^z & 1 \\ b^z & 1 \end{vmatrix}, \begin{vmatrix} a^y & a^z \\ b^y & b^z \end{vmatrix}, \begin{vmatrix} a^x & a^z \\ b^x & b^z \end{vmatrix}, \begin{vmatrix} a^x & a^y \\ b^x & b^y \end{vmatrix} \right).$$

This order conveniently encodes the direction of the line in the first 3 Plücker coordinates, and the *angular momentum* of the line about the origin in the last 3 coordinates (for a definition of angular momentum, see [40]).

Hence, the Plücker coordinates of a line in 3D is a 6-dimensional vector. However, not every 6-dimensional vector can be identified with the Plücker coordinates of a line. Let $s = (a, b, c, d, e, f)$ and $s^* = (d, e, f, a, b, c)$. Then, $s \in \mathbb{R}^6$ is said to satisfy the Grassmann-Plücker relation if the dot product of

s and s^* is zero; that is, $\langle s, s^* \rangle = 0$. Note that if $\mathbf{u} = (a, b, c)$ and $\mathbf{v} = (d, e, f)$, then $\langle s, s^* \rangle = s \cdot s^* = \mathbf{u} \cdot \mathbf{v} + \mathbf{v} \cdot \mathbf{u} = 0$ if and only if $\mathbf{u} \cdot \mathbf{v} = 0$. Vectors in \mathbb{R}^6 that are identified with the Plücker coordinates of a 3D line are precisely those satisfying this relation.

There is an infinite number of 6-vector representations of a specific line in 3D (since scalar multiples of the vector preserve the line). For the further analysis, however, it is important to note that different pairs of points along the line may also produce identical Plücker coordinates. This is due to the fact that the distance between the points is directly related to the magnitude of the direction vector, found in the first 3 coordinates of the line.

Tensors

A k -tensor is defined by the sum of k -extensors. If a k -tensor is itself a k -extensor, it is referred to as a *decomposable tensor*; otherwise it is an *indecomposable tensor*. In fact, 2-tensors are identified with \mathbb{R}^6 .

Connecting screws and tensors

Every 6-vector is a 2-tensor and also an instantaneous screw. We can classify the decomposable tensors as either pure rotations or pure translations, and the indecomposable tensors as screws with both rotational and translational components.

It turns out that the vectors that satisfy the Grassmann-Plücker relation are exactly the decomposable tensors. Screws that include both a rotation and translation are, as tensors, the sum of a pure rotation 2-extensor and a pure translation 2-extensor.

3.2.3 Infinitesimal motions and bar constraints

We are now ready to examine the infinitesimal motions of a body-and-bar framework. Let $(G, \mathbf{p}, \mathbf{q}, L)$ be a framework with the embedding $\mathbf{T} \in (\text{SE}(3))^n$ and a distance function L calculated by \mathbf{p} and \mathbf{q} . We define the *infinitesimal motion* of this framework to be an assignment $\mathbf{s} \in (\mathbb{R}^6)^n$ where each \mathbf{s}_i is a 6-vector associated with an instantaneous screw. A trivial infinitesimal motion assigns the same screw to all \mathbf{s}_i .

Let \mathbf{p}_{ij} and \mathbf{q}_{ij} be the attachment points of the bar $\mathbf{p}_{ij}\mathbf{q}_{ij}$ in bodies i and j respectively. Now let \mathbf{u} and \mathbf{v} be the instantaneous velocities assigned to these attachment points by the screw motions associated with \mathbf{s}_i and \mathbf{s}_j . Then the constraint imposed by bar $\mathbf{p}_{ij}\mathbf{q}_{ij}$ is maintained infinitesimally if the relative velocity of \mathbf{p}_{ij} and \mathbf{q}_{ij} is orthogonal to the direction of the line determined by $\mathbf{p}_{ij}\mathbf{q}_{ij}$, or, in mathematical terms, $\langle \mathbf{u} - \mathbf{v}, \mathbf{p}_{ij} - \mathbf{q}_{ij} \rangle = 0$. After a short derivation, one can conclude that the bar constraint is maintained infinitesimally if and only if $\langle \mathbf{s}_i^*, \mathbf{p}_{ij} \vee \mathbf{q}_{ij} \rangle - \langle \mathbf{s}_j^*, \mathbf{p}_{ij} \vee \mathbf{q}_{ij} \rangle = 0$, where $\mathbf{p}_{ij} \vee \mathbf{q}_{ij}$ are the Plücker coordinates of the line defined by the bar.

Rigidity matrix

Now we can build the rigidity matrix for body-and-bar frameworks. If m is the number of bars and n the number of bodies in one such framework, then the rigidity matrix for the framework is an $m \times 6n$ matrix in which each bar is defined by one row, and each body is associated with 6 columns. For instance, bar ij corresponds to the following row in the rigidity matrix:

$$\begin{array}{ccccccc}
 & \dots & \xrightarrow{\mathbf{s}_i^*} & \dots & \xrightarrow{\mathbf{s}_j^*} & \dots & \\
 \boxed{\dots \mathbf{0} \dots} & \boxed{\mathbf{p}_{ij} \vee \mathbf{q}_{ij}} & \boxed{\dots \mathbf{0} \dots} & \boxed{-(\mathbf{p}_{ij} \vee \mathbf{q}_{ij})} & \boxed{\dots \mathbf{0} \dots} & &
 \end{array}$$

The null space of the matrix (i.e., its kernel) is the space of infinitesimal motions. The trivial infinitesimal motions correspond to a subspace of the kernel determined by the 6 vectors

$$\begin{aligned}
&(1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, \dots, 1, 0, 0, 0, 0, 0) \\
&(0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, \dots, 0, 1, 0, 0, 0, 0) \\
&(0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, \dots, 0, 0, 1, 0, 0, 0) \\
&(0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, \dots, 0, 0, 0, 1, 0, 0) \\
&(0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, \dots, 0, 0, 0, 0, 1, 0) \\
&(0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, \dots, 0, 0, 0, 0, 0, 1)
\end{aligned}$$

Therefore, the kernel has dimension at least $6 = 3(3 + 1)/2$, and, if the rigidity matrix achieves its maximum rank of $6n - 6$, then the structure is *infinitesimally rigid*.

3.2.4 3D body-and-hinge systems

We now turn to *body-hinge* structures and discuss their rigidity properties. In the context of this work, we will move directly to the infinitesimal theory.

Infinitesimal rigidity

Since the elements in $\text{se}(3)$ are the instantaneous screws in 3D, infinitesimally, a hinge constraint is preserved if the relative screw motion is a pure rotation about the hinge axis, i.e., the hinge axis must be the same as the screw axis. Since a screw can be represented by a 6-vector interpreted as $(-\boldsymbol{\omega}, \mathbf{v})$, where $\boldsymbol{\omega}$ corresponds to the angular velocity vector of the screw, we can conclude that the screw written as $(-w_x, -w_y, -w_z, v_x, v_y, v_z)$, must be equal to the Plücker coordinates of a hinge axis, (a, b, c, d, e, f) , up to a scalar α :

$$(-w_x, -w_y, -w_z, v_x, v_y, v_z) = \alpha(a, b, c, d, e, f).$$

There are many different ways to express this dependency. Consider the following method:

$$\begin{array}{ccc} -w_x = \alpha a & & -w_y = \alpha b \\ \Downarrow & & \Downarrow \\ -bw_x = \alpha ab & & -aw_y = \alpha ab \\ & \Downarrow & \\ & -bw_x = -aw_y & \\ & -bw_x + aw_y = 0 & \end{array}$$

Similarly,

$$\begin{array}{ccc} -w_y = \alpha b & & -w_z = \alpha c \\ \Downarrow & & \Downarrow \\ -cw_y = \alpha bc & & -bw_z = \alpha bc \\ & \Downarrow & \\ & -cw_y = -bw_z & \\ & -cw_y + bw_z = 0 & \end{array}$$

By expressing the correlation between the rest of the coordinates in the same way, we obtain the following five linear equations:

$$\begin{aligned} -bw_x + aw_y &= 0 \\ -cw_y + bw_z &= 0 \\ -dw_z - cv_x &= 0 \\ ev_x - dv_y &= 0 \\ fv_y - ev_z &= 0 \end{aligned}$$

Using these equations we can represent each hinge in the rigidity matrix with the corresponding 5 rows:

\mathbf{s}_i^*							\mathbf{s}_j^*							
\dots	v_x	v_y	v_z	$-w_x$	$-w_y$	$-w_z$	\dots	v_x	v_y	v_z	$-w_x$	$-w_y$	$-w_z$	\dots
$\dots \mathbf{0} \dots$	0	0	0	b	$-a$	0	$\dots \mathbf{0} \dots$	0	0	0	$-b$	a	0	$\dots \mathbf{0} \dots$
$\dots \mathbf{0} \dots$	0	0	0	0	c	$-b$	$\dots \mathbf{0} \dots$	0	0	0	0	$-c$	b	$\dots \mathbf{0} \dots$
$\dots \mathbf{0} \dots$	c	0	0	0	0	$-d$	$\dots \mathbf{0} \dots$	$-c$	0	0	0	0	d	$\dots \mathbf{0} \dots$
$\dots \mathbf{0} \dots$	e	$-d$	0	0	0	0	$\dots \mathbf{0} \dots$	$-e$	d	0	0	0	0	$\dots \mathbf{0} \dots$
$\dots \mathbf{0} \dots$	0	f	$-e$	0	0	0	$\dots \mathbf{0} \dots$	0	$-f$	e	0	0	0	$\dots \mathbf{0} \dots$

Since a hinge allows only a single rotational degree of freedom and a framework in 3D has a total of 6 trivial infinitesimal motions, we expect to obtain 5 linearly independent rows in the rigidity matrix. However, to ensure this we must carefully choose the method for expressing the constraint between the screw and the Plücker coordinates of the hinge axis (see Section 3.4). Determining whether the structure is infinitesimally rigid is, again, equivalent to determining if the dimension of the kernel of the rigidity matrix is 6.

3.3 Combinatorial rigidity

Combinatorial rigidity for body-and-bar structure has been characterized by Tay [32].

Theorem 3.3.1. (*Tay’s Theorem*) *A 3-dimensional body-and-bar structure is generically rigid if and only if the associated multigraph with vertices instead of bodies and edges instead of bars is composed of 6 edge-disjoint spanning trees.*

Note that Tay’s theorem was stated originally in dimension d , but we focus on its interpretation in 3D since it is relevant for our work.

The edge-disjoint spanning trees referred to in the theorem are a special combinatorial object for which we give an intuitive description. A *spanning tree* of a graph $G=(V,E)$ is a subgraph whose edges touch every vertex of G , without forming any cycles. Thus, a graph may have zero, one, or many spanning trees. Now imagine that we draw the graph and color as many spanning trees as we can by using a different color for every spanning tree and not recoloring any edge. By doing so, we will actually identify a set of *edge-disjoint* spanning trees of that graph (i.e., spanning trees that do not share common edges). The maximum number of colors that we need is exactly the number of the

graph's edge-disjoint spanning trees. As illustrated by Theorem 3.3.1, this number plays an important role in determining the rigidity properties of a body-and-bar framework.

Nash-Williams and Tutte's theorem

Tay's theorem characterizes body-and-bar rigidity with edge-disjoint spanning trees. Nash-Williams and Tutte (independently) proved a connection between the number of edge-disjoint spanning trees and a “counting” property of the graph. This relation is expressed in the following theorem.

Theorem 3.3.2. (*Nash-Williams, Tutte Theorem*) *Let $G=(V, E)$ be a multigraph with $|V| = n$ and $|E| = m$. Then G has exactly k edge-disjoint spanning trees if and only if the following two conditions are true:*

1. $m = k(n - 1) = kn - k$ and
2. *for all subsets of n' vertices spanned by m' edges, $m' \leq k(n' - 1)$.*

As a consequence of Theorems 3.3.1 and 3.3.2, we can characterize the rigidity properties of a body-and-bar structure described by this combinatorial “counting” property. In fact, this property belongs to a more generalized condition called (k, l) -*sparsity*.

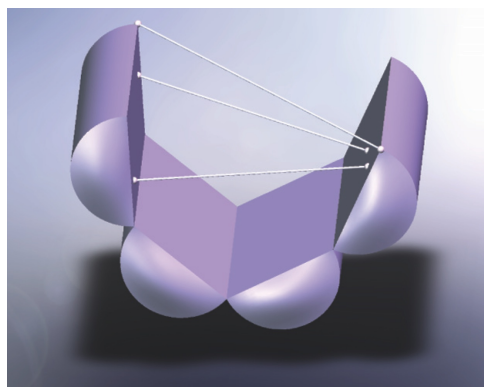
Definition 3.3.3. A multigraph G with n vertices and m edges is called (k, l) -sparse if for all subsets of n' vertices spanned by m' edges, $m' \leq kn' - l$. If, in addition, $m = kn - l$, G is called (k, l) -*tight*.

Lee and Streinu [22] extended the 2D pebble game algorithm of Jacobs and Hendrickson to address (k, l) -sparsity. Thus, we can apply those pebble game algorithms to analyze body-and-bar rigidity. In fact, we can use them to study

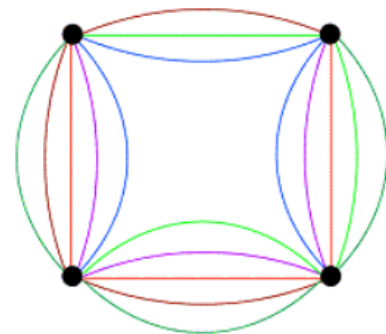
the rigidity properties of body-bar-hinge systems as well, due to Whiteley and Tay who introduce the representation of a hinge as 5 bars [39, 33].

To demonstrate the application of these theorems, consider the body-bar-hinge structure illustrated at the beginning of the chapter and reprinted in Figure 3.3(a) for convenience. It consists of 4 bodies connected by 3 hinges and 3 bars. We can build the multigraph associated with this structure by representing each body with a vertex and each bar with an edge as shown in Figure 3.3(b). Note that a hinge (corresponding to 5 bars) is modeled as 5 edges.

The graph has 6 edge-disjoint spanning trees, each of which is illustrated with a different color in 3.3(b). Since every hinge is modeled with bars, we can use Theorem 3.3.1 to prove that the structure is generically rigid. Furthermore, following the notation from Theorem 3.3.2, $n = 4$, $m = 18$ and $k = 6$, and the reader can easily verify that both conditions stated in the theorem are satisfied. Note that the graph is also $(6, 6)$ -tight by Definition 3.3.3.



(a) A body-bar-hinge structure composed of 4 bodies, 3 hinges and 3 bars.



(b) Associated multigraph containing a vertex for each body and 1 edge for each bar.

Figure 3.3: Minimally rigid body-bar-hinge structure. Reprinted from [21].

3.4 Contributions

According to the theory explained in Section 3.2, every non-degenerate hinge (i.e., a hinge that allows only 1 rotational degree of freedom) is associated with five rows in the rigidity matrix. However, in certain cases, we found that the approach from Section 3.2.4 caused dependencies among the rows representing a single hinge. We present an approach that addresses this problem and always produces 5 linearly independent rows. To the best of our knowledge, this is the first careful analysis to be performed for body-and-hinge rigidity.

As in Section 3.2.4, infinitesimally, a hinge constraint is preserved if the relative screw motion is a pure rotation about the hinge axis. Furthermore, a screw can be represented by a 6-vector interpreted as $(-\boldsymbol{\omega}, \mathbf{v})$ where $\boldsymbol{\omega}$ corresponds to the angular velocity vector of the screw. Thus, the screw must be equal to the Plücker coordinates of a hinge axis up to a scalar:

$$(-w_x, -w_y, -w_z, v_x, v_y, v_z) = \alpha(a, b, c, d, e, f). \quad (3.4)$$

In Section 3.2.4 we describe one way for expressing the hinge constraint with five linear equations. In particular, we compare the correlations between every two adjacent vector elements imposed by Equation 3.4. However, this method does not always produce 5 linearly independent rows. The following example illustrates this.

Consider a hinge between points $(1, 1, 0)$ and $(0, 1, 0)$. Its Plücker coordinates are $(1, 0, 0, 0, 0, 1)$ and using the method from Section 3.2.4, we get three rows of zeros in the rigidity matrix:

\mathbf{s}_i^*							\mathbf{s}_j^*							
\dots	v_x	v_y	v_z	$-w_x$	$-w_y$	$-w_z$	\dots	v_x	v_y	v_z	$-w_x$	$-w_y$	$-w_z$	\dots
$\dots \mathbf{0} \dots$	0	0	0	0	-1	0	$\dots \mathbf{0} \dots$	0	0	0	0	1	0	$\dots \mathbf{0} \dots$
$\dots \mathbf{0} \dots$	0	0	0	0	0	0	$\dots \mathbf{0} \dots$	0	0	0	0	0	0	$\dots \mathbf{0} \dots$
$\dots \mathbf{0} \dots$	0	0	0	0	0	0	$\dots \mathbf{0} \dots$	0	0	0	0	0	0	$\dots \mathbf{0} \dots$
$\dots \mathbf{0} \dots$	0	0	0	0	0	0	$\dots \mathbf{0} \dots$	0	0	0	0	0	0	$\dots \mathbf{0} \dots$
$\dots \mathbf{0} \dots$	0	1	0	0	0	0	$\dots \mathbf{0} \dots$	0	-1	0	0	0	0	$\dots \mathbf{0} \dots$

In order to avoid such behavior, we present another method in which we formulate the five equations in the following way:

$$\begin{array}{ll}
 -w_x = \alpha a & -w_y = \alpha b \\
 \Downarrow & \Downarrow \\
 -bw_x = \alpha ab & -aw_y = \alpha ab \\
 & \Downarrow \\
 & -bw_x = -aw_y \\
 & -bw_x + aw_y = 0
 \end{array}$$

Similarly,

$$\begin{array}{ll}
 -w_x = \alpha a & -w_z = \alpha c \\
 \Downarrow & \Downarrow \\
 -cw_x = \alpha ac & -aw_z = \alpha ac \\
 & \Downarrow \\
 & -cw_x = -aw_z \\
 & -cw_x + aw_z = 0
 \end{array}$$

and

$$\begin{array}{ll}
 -w_x = \alpha a & v_x = \alpha d \\
 \Downarrow & \Downarrow \\
 -dw_x = \alpha ad & av_x = \alpha ad \\
 & \Downarrow \\
 & -dw_x = av_x \\
 & -dw_x - av_x = 0
 \end{array}$$

By expressing the correlations between the rest of the coordinates in the same way, we obtain the following five linear equations:

$$\begin{aligned}
-bw_x + aw_y &= 0 \\
-cw_x + aw_z &= 0 \\
-dw_x - av_x &= 0 \\
-ew_x - av_y &= 0 \\
-fw_x - av_z &= 0
\end{aligned}$$

The resulting rigidity matrix has the form:

			\mathbf{s}_i^*									\mathbf{s}_j^*								
\dots	v_x	v_y	v_z	$-w_x$	$-w_y$	$-w_z$	\dots	v_x	v_y	v_z	$-w_x$	$-w_y$	$-w_z$	\dots						
$\dots\mathbf{0}\dots$	0	0	0	b	$-a$	0	$\dots\mathbf{0}\dots$	0	0	0	$-b$	a	0	$\dots\mathbf{0}\dots$						
$\dots\mathbf{0}\dots$	0	0	0	c	0	$-a$	$\dots\mathbf{0}\dots$	0	0	0	$-c$	0	a	$\dots\mathbf{0}\dots$						
$\dots\mathbf{0}\dots$	a	0	0	$-d$	0	0	$\dots\mathbf{0}\dots$	$-a$	0	0	d	0	0	$\dots\mathbf{0}\dots$						
$\dots\mathbf{0}\dots$	0	a	0	$-e$	0	0	$\dots\mathbf{0}\dots$	0	$-a$	0	e	0	0	$\dots\mathbf{0}\dots$						
$\dots\mathbf{0}\dots$	0	0	a	$-f$	0	0	$\dots\mathbf{0}\dots$	0	0	$-a$	f	0	0	$\dots\mathbf{0}\dots$						

In this method we take the correlation between $-w_x$ and a (we will call it the “main” correlation) and compare it to the correlations of all five other vector elements. Thus, using the example above, we would get five linearly independent rows.

			\mathbf{s}_i^*									\mathbf{s}_j^*								
			v_x	v_y	v_z	$-w_x$	$-w_y$	$-w_z$				v_x	v_y	v_z	$-w_x$	$-w_y$	$-w_z$			
\cdots			0	0	0	0	-1	0	\cdots			0	0	0	0	1	0	\cdots		
\cdots			0	0	0	0	0	-1	\cdots			0	0	0	0	0	1	\cdots		
\cdots			1	0	0	0	0	0	\cdots			-1	0	0	0	0	0	\cdots		
\cdots			0	1	0	0	0	0	\cdots			0	-1	0	0	0	0	\cdots		
\cdots			0	0	1	-1	0	0	\cdots			0	0	-1	1	0	0	\cdots		

This result comes from the fact that $a = 1$, and since a appears on every row and in different columns in each row, there are no zero rows, and thus no dependencies. In fact, we get five independent rows for any $a \neq 0$.

We generalize this approach to address any hinge axis. As long as the Plücker coordinates contain a nonzero vector element, we can take this vector element and use its correlation with the corresponding screw element, as a “main” correlation, comparing it with all other five correlations. In this way, we avoid undesired dependencies in the rigidity matrix.

Chapter 4

Static Analysis for Understanding Allosteric Effect

In this chapter we focus on the problem of **understanding the allosteric effect**: how is it possible for a motion occurring at an allosteric site (due to ligand binding) to cause motion at the active site, thus altering the function of the protein? In order to investigate this question, we use the theory from Chapter 3 and present a piece of software to conduct the analysis. We describe the steps to find and “prepare” a protein for our analysis, the software that we use, and details of the implementation of the project. Then we discuss different approaches for analyzing the data and present some preliminary results. As a further reference on biological concepts and definitions, see [25].

4.1 Preliminaries

The static analysis of the protein is conducted using information about its rigidity. This information is obtained by the *FIRST: Floppy Inclusions and*

Rigid Substructure Topography, software [2] which models the molecule as a body-bar-hinge structure. Here we explain how FIRST works, since the further analysis is based on the results from the application.

FIRST allows users to specify a protein from the Protein Data Bank or upload their own. The Protein Data Bank (PDB) [5] is an online database containing information about experimentally-determined structures of molecules. Having a protein, the software performs the following steps:

1. It adds hydrogen atoms to the PDB file (a file describing the protein structure). Not all proteins in the Protein Data Bank have hydrogens (H atoms) included in the PDB files. However, these H atoms may form hydrogen bonds that impose constraints on the molecule. FIRST computes the position of the hydrogens in order to model the rigidity of the protein.
2. Having the hydrogens in the PDB file, FIRST then computes and adds hydrogen bonds. An optional parameter lets the user specify an energy cut-off distance which regulates the number of hydrogen bonds included in the model. The default cut-off distance is -1 kcal/mol.
3. In order to model the protein as a body-bar-hinge structure, FIRST uses knowledge of chemical interactions between the atoms. A covalent bond, for instance, is a very strong bond represented by a *hinge* in the model, while a hydrogen bond may be weaker and is modeled by FIRST with a variable number of *bars* (usually less than 5). FIRST also models hydrophobic interactions with “hydrophobic tethers,” associated with 2 bars.

4. Having the body-bar-hinge model of the protein, the software uses a pebble game algorithm to find all rigid components (bodies) in the molecule. The results from the analysis are output through a set of text files.

4.2 Overview of steps

We begin by giving an overview of the steps of the analysis. Figure 4.1 illustrates their sequence.



Figure 4.1: Steps of the analysis

1. **Literature Search** - First, we need to find candidate proteins for our analysis. Since we are analyzing the effect of the allosteric site on the active site, we need to know the exact locations of these sites. Therefore, we look for proteins for which this information is known. The process for identifying such a protein involves searching the literature for publications that list the location of its allosteric site, such as, for instance, publications discussing the discovery of novel allosteric sites (we must also search for a second structure in order to “extract” information about the location of the active site as explained in Section 4.4). Once identified a candidate protein structure, we find its PDB file in the Protein Data Bank.

2. **SWISS-MODEL** - Very often, however, protein structures in the Protein Data Bank have missing residues. Since these missing residues are important for the rigidity analysis of the structures, we use another piece of software to properly prepare a PDB file for analysis.

2.1 *Adding missing residues* - The software we use is called SWISS-MODEL [6, 19, 26]; this application builds homology models of proteins at different levels of complexity. It has a web-based interface, the SWISS-MODEL Workplace, where users submit a protein sequence (taken from the Protein Data Bank); in response, the software produces a new PDB file with the missing residues added.

2.2 *Adding chain ID* - In an original PDB file, next to each atom there is a letter (usually A, B, C, etc.), called the *chain ID*, denoting which chain the atom belongs to. The chain IDs, however, are missing in the PDB file produced by SWISS-MODEL. Since FIRST needs them to perform the rigidity analysis, we add them to the new PDB file via a Perl script.

3. **FIRST** - Having prepared the PDB file, we run it through FIRST, which produces a number of output files describing the rigidity of the molecular structure. In particular, it identifies the rigid components (or bodies) in the structure. We use the output files to generate a single XML file that contains information about each atom (ID and position in 3D), body (ID and a list of the atoms in it), hinge (the 2 atoms and bodies that it connects), and bar (the 2 atoms that identify it). The generation of this file is done via another Perl script.

4. **Infinitesimal Rigidity** - The next step is to perform the infinitesimal rigidity analysis and our contribution starts here. We use the XML file

described above to build the rigidity matrix for the protein (see Chapter 3.2.4). Once we have the rigidity matrix, we compute its null space. These computations are performed via MATLAB as explained in Section 4.3.

5. **Analysis** - The last step is to use the computed null space to study the range of possible motions of the protein, which includes:

5.1 *Identifying relevant bodies* - The relevant bodies are usually the bodies that contain atoms found in the active or the allosteric sites as their motion could potentially tell us something about the allosteric effect. Section 4.4 describes how we find such bodies.

5.2 *Identifying relevant null space vectors* - A relevant null space vector is a vector that produces an “interesting” motion when every body is acted upon by the screw described in the body’s corresponding 6 entries in the vector. By “interesting” motion, we intuitively mean one that reveals a relationship or a dependency between the active and allosteric sites. We have developed three heuristics for finding relevant null space vectors, described in detail in Section 4.6.

4.3 Implementation details

In order to conduct the infinitesimal rigidity analysis and test different heuristics, we developed a MATLAB program for computing the rigidity matrix and its null space, as well as a web application for visualizing the subsequent results. We call this project MotionSpace as it is a tool for examining the motion space of the protein. In this section, we describe briefly the structure

and function of both the MotionSpace program and the MotionSpace web interface. We conclude with a discussion of implementation details and open issues.

MotionSpace program

The purpose of the MotionSpace program is to compute the rigidity matrix of a protein structure, then find its null space. In order to do this the program first reads the contents of the XML file generated from the output of FIRST (for more information about the file, see step 4 in Section 4.2). For each hinge, the program computes the Plücker coordinates of the two atoms to which the hinge is connected (as described in Section 3.2.2), then uses infinitesimal rigidity theory (see Section 3.2) to compute the 5 rows of the rigidity matrix corresponding to this hinge. The program then reads the data for the bars and computes the rows in the rigidity matrix associated with them (as described in Section 3.2.3). This completes the formation of the rigidity matrix. We proceed by computing its null space and saving it in a file accessible by the MotionSpace interface.

Since the trivial motions are included in the null space, we let the user choose one body to “pin down.” By doing so, we add 6 more rows at the bottom of the rigidity matrix which, if extracted at the columns of that body, form the identity matrix. This removes the trivial degrees of freedom from the null space.

Note that, due to the limited computational power of MATLAB, we cannot compute the null space of structures with very large number of bodies. In order to regulate this number, we specify an *optimal* energy cut-off distance when we run FIRST. This optimal distance results in a number of bodies that is

sufficiently small to make computation feasible, and sufficiently large to allow for the exploration of a bigger range of possible motions.

MotionSpace web interface

The MotionSpace interface is a web application combining scripts written in different languages. Its main purpose is to allow users to explore the effect of various linear combinations of null space vectors on the range of motions of the bodies (or, equivalently, on the instantaneous velocities of the atoms). Figure 4.2 shows a snapshot of the application. The panel on the left consists of drop-down menus and buttons letting the user select different combinations of null space vectors. The protein on the right is always the original protein, while the protein on the left displays a coloring according to the calculated new velocities of the atoms. The two proteins are visualized via Jmol applets and can be easily manipulated on the screen (i.e., rotated, zoomed in, etc.). Jmol is an open-source molecular viewer that can be customized with Jmol scripts [4].

One of the main components of the web application is a Java program that calculates the new velocities of the atoms. It reads the null space vectors from the file produced by the MotionSpace program and, using the coefficients selected by the user, computes the summation vector of their linear combination. By reading the XML file with the results from FIRST, the program loads the data for the coordinates of the atoms and the bodies they are in. Every 6-entries in the summation vector are associated with a particular body and represent the screw that acts upon it. Based on this screw, we calculate the infinitesimal velocity of each of the atoms in the body as explained in Section 3.2.1.

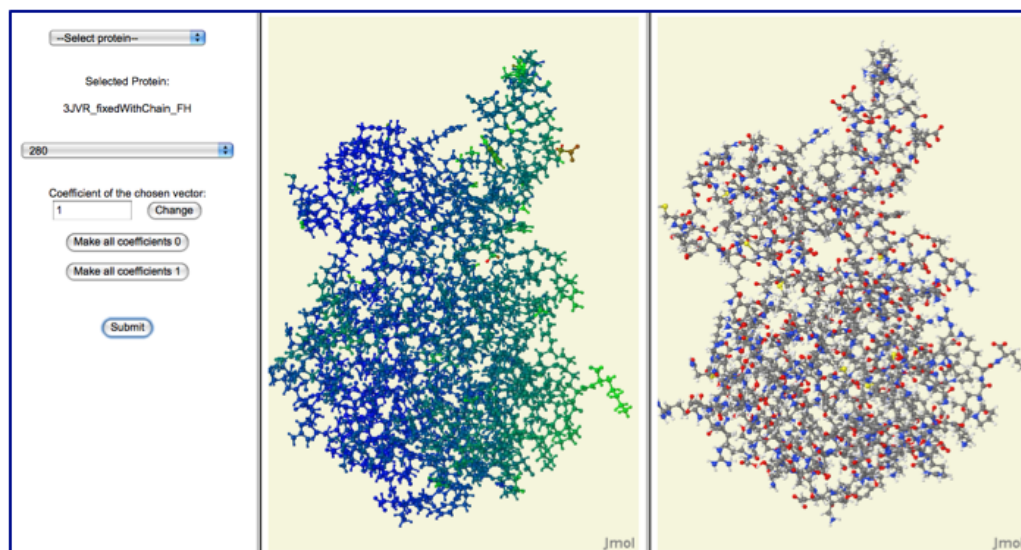


Figure 4.2: Snapshot from the MotionSpace interface

Once we have the velocities of all atoms, we compute and normalize their magnitudes, and represent them with RGB colors ranging from blue to red on a scale where blue denotes a magnitude of 0 and red a magnitude of 1; Figure 4.3 illustrates the scale. For visualization, we create a Jmol script that applies the new coloring to the protein.



Figure 4.3: Colors corresponding to the velocity of the atoms

There are two main stages of running the web application. First is the *initialization stage* illustrated in Figure 4.4. Before starting the application, it is important that the null space files have been created by the MotionSpace program. When we open the MotionSpace interface in the browser, Java Script (JS) is used to populate a directory drop-down menu for selecting a protein. Once we choose a protein, we use PHP to read some of the files for that

protein and save the information in JS arrays that populate the second drop-down menu on the screen. HTML loads and displays the 2 proteins on the right using the Jmol applet.



Figure 4.4: Initialization stage

The second stage is the *interactive stage* illustrated in Figure 4.5. Selecting null space vectors and changing their coefficients results in a loop between JS and the screen, in which JS is constantly updating its arrays and the information displayed on the screen. When we click the “Submit” button, AJAX is used to pass the selected coefficients to PHP, which in turn starts the Java program. The program reads the XML and null space files as described above, and calculates the magnitudes of the atom velocities. Then, with the help of PHP, the Jmol applet reads the Jmol script file produced by Java and reloads the protein on the left with the new coloring. For more information, see the ReadMe.txt file included in the application package.

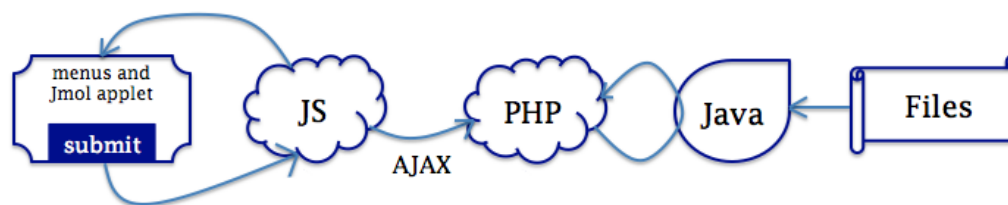


Figure 4.5: Interactive stage

4.4 Identifying relevant bodies

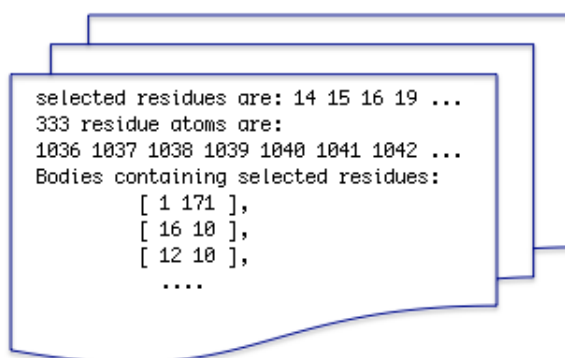
In order to perform the analysis and test the results of the theory, we need to know the exact locations of the active and allosteric sites (by exact location, we mean the residue IDs that form these sites). However, publications found during the literature search step in Section 4.2 do not always provide us with all of the desired information for that protein. Such publications usually list only a small number of the residues forming the allosteric site while we need all of them together with the residues forming the active site. In order to find this information, we use one of the structure features on the Protein Data Bank website, called Analysis of Ligand-Protein Contacts (LPC). This feature gives detailed information about each ligand bound to the protein and all residues forming the site where it binds. Our protein candidate usually has a ligand bound to an allosteric site, so, using LPC, we can find the exact location of the site.

We must also find the residues forming the active site. This involves searching through the Protein Data Bank for the same protein with a ligand bound to the active site rather than the allosteric site. When we find such an entry in the online database, we again use LPC to manually extract the numbers of the residues at the location (in this case, the active site) where the ligand binds.

Here we must note that the residue IDs that we obtain from the LPC feature correspond to the residue IDs in the original PDB files. After we run SWISS-MODEL to add missing residues, the residue numbers may change and, since we use the protein from SWISS-MODEL in the further analysis, we must record the correct data for the active and allosteric sites. We do this by

aligning the original protein and the protein produced by SWISS-MODEL in a software called Pymol.

Once we know where exactly a particular site (active or allosteric) is in the protein, we run a perl script that: 1) reads the PDB file for the protein and finds the atom numbers corresponding to the residues forming the site, and 2) goes through the XML file generated from the output of FIRST (describing the rigid components in the protein), and finds all bodies in which these atoms occur. The output from the script consists of the residues forming the given active or allosteric site, the IDs of the atoms in these residues, and a set of $[i, j]$ pairs in which i represents the id of the body that contains j number of atoms from the ones just listed. Figure 4.6 shows an extract from the output file produced by the script.



```

selected residues are: 14 15 16 19 ...
333 residue atoms are:
1036 1037 1038 1039 1040 1041 1042 ...
Bodies containing selected residues:
[ 1 171 ],
[ 16 10 ],
[ 12 10 ],
....

```

Figure 4.6: Information for the active site of CHK1

4.5 Identifying relevant null space vectors

The goal of our analysis is to examine the range of possible motions of the protein which could give us insight on the mechanisms of the allosteric effect. Every vector in the kernel of the rigidity matrix introduces an additional degree

of freedom to the system, thus contributing to the framework’s motion by assigning instantaneous screws to each body. Analyzing a body’s possible motions becomes equivalent to analyzing its corresponding 6 entries in each of the null space vectors as they describe the screw that acts upon the body.

Here we discuss 3 different heuristics that we use for identifying relevant infinitesimal motions, or equivalently, finding “interesting” null space vectors as described in Section 4.2. All heuristics are based on the observation that the allosteric effect involves a correlated movement of the allosteric and active sites. Our goal is to find a “path” between them. Note that, since there is more than one body that includes atoms from a particular site, we have many different options for choosing an *allosteric site body* and an *active site body* in the heuristics described below.

Heuristic 1: Counting zeros

Since we expect the motions of the allosteric and active sites to be related regardless of the motions of other parts of the protein, we search for the null space vectors that assign zero velocities to the largest number of bodies while keeping the velocities of the allosteric site body and active site body nonzero. By doing so, our goal is to find an interaction between the active and allosteric sites that minimizes motion from the rest of the protein. Following is a pseudocode of the heuristic.

```

Let N be a matrix whose columns are the null space vectors of
the rigidity matrix
Let body_active be the active site body
Let body_allosteric be the allosteric site body

```

```

Initialize an empty vector zeros_count of length the number of
columns of N
For each column i in N
    If the norm of the 6-vector in i corresponding to body_active
    is not 0 and the norm of the 6-vector in i corresponding to
    body_allosteric is not 0
        count = 0
        For each body j different than body_active and body_allosteric
            If the norm of the 6-vector in i corresponding to j is 0
                Increment count with 1
            Endif
        Endfor
        If count is greater than 0
            Set zeros_count[i] = count
        Else
            Set zeros_count[i] = 0
        Endif
    Endfor
Sort the vectors in a decreasing order of their corresponding
entries in zeros_count
Print the indices of the first 10 vectors

```

Heuristic 2: Finding maxima I

Another way to examine the interactions between the active and allosteric sites is to look for null space vectors associated with simultaneous large movements in the two sites. The intuition behind this heuristic is the hypothesis

that a big motion at the allosteric site should cause a big motion at the active site. We find the relevant null space vectors by performing the following steps:

1. Rank the null space vectors by the degree to which they maximize the active site motion
2. Rank the null space vectors by the degree to which they maximize the allosteric site motion
3. For each vector, add the two numbers assigned to it by the two rankings
4. Sort the vectors according the these sums and print the indices of the first 10 of them

Following is a pseudocode of the heuristic (note that when we say “maximum body” we mean that the norm of the 6-entries vector associated with that body in a null space vector is maximum).

Let N be a matrix whose columns are the null space vectors of the rigidity matrix

Let n and m denote the number of rows and columns in N

Let body_active be the active site body

Let body_allosteric be the allosteric site body

Initialize a vector act_site_ranking with m zero-elements

Initialize a vector all_site_ranking with m zero-elements

Initialize an empty sum_vector

$\text{act_rank} = 1$

While there are elements in act_site_ranking that are equal to 0

Find the vector with maximum active site body and index

such that $\text{act_site_ranking}[\text{index}] = 0$

```

    Set act_site_ranking[index] = act_rank
    Increment act_rank with 1
Endwhile
all_rank = 1
While there are elements in all_site_ranking that are equal to 0
    Find the vector with maximum allosteric site body and index
    such that all_site_ranking[index]=0
    Set all_site_ranking[index] = all_rank
    Increment all_rank with 1
Endwhile
sum_vector = act_site_ranking + all_site_ranking
Sort the vectors in an increasing order of their corresponding
entries in sum_vector
Print the indices of the first 10 vectors

```

Heuristic 3: Finding maxima II

We based heuristic 2 on the hypothesis that a big motion at the allosteric site should cause a big motion at the active site. However, it is possible that one of the motions is smaller, but the correlated movement of the two sites still causes a conformational change in the active site that changes the behavior of the protein. In order to examine this possibility, we search for null space vectors that maximize the velocity of the allosteric site bodies and active site bodies separately. By applying either of these vectors, we can examine the “effect” that one of the sites has on the other (for instance, we can check whether a big motion at the allosteric site induces a conformational change in the active site).

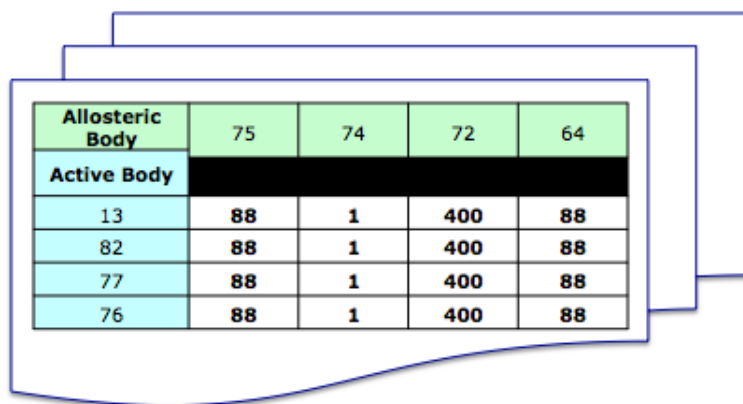
4.6 Identifying relevant infinitesimal motions

Once we identify the bodies that contain atoms from the active and allosteric sites, we apply the heuristics described above in order to find null space vectors that produce “interesting” motion (as explained in Section 4.3). We do this by performing the following 3 steps:

1. *Select bodies* - The perl script that we run in order to find the bodies at the active and allosteric sites usually gives us a large group of bodies containing different number of atoms from the sites (for convenience we will call these atoms *site atoms*). Since we manually analyze different combinations of these bodies (as explained in the subsequent steps), we select a smaller set of the bodies that will be used as a representative body-set for that particular site. In the output file from the script, the bodies are listed in a decreasing order by the number of site atoms that they contain. For the representative body-set, we select the bodies with the most number of site atoms.
2. *Apply heuristics* - Once we have the bodies at the active and allosteric sites, we apply each of the heuristics described in Section 4.6 to obtain a set of potentially relevant null space vectors. We say *potentially* relevant, because not all of them necessarily produce colorings of the protein that depict a clear relationship between the active and allosteric sites.

We create tables (such as the one in Figure 4.7) that list the active and allosteric bodies for the protein, and, for each pair, record the computed output from applying a particular heuristic. Note that the resulting null space vectors in the table in Figure 4.7 follow a nice pattern (which makes the identification of relevant null space vectors much easier). Often,

however, our data does not follow any patterns (i.e., we get different numbers in the table entries), and in this case we start searching for the null space vectors that appear most frequently.



Allosteric Body	75	74	72	64
Active Body				
13	88	1	400	88
82	88	1	400	88
77	88	1	400	88
76	88	1	400	88

Figure 4.7: Data from applying heuristic 1 to protein PDK1

3. *Analyze* - After we identify relevant null space vectors we visualize each of them via the web application discussed in Section 4.3. This process consists of setting the coefficient of the identified null space vector to 1, and the coefficients of all other null space vectors to 0, then visually examining the resulting coloring of the protein. We may also visualize various combinations of vectors as well (i.e., set the coefficients to more than one vectors to equal non-zero numbers on the scale 0 to 1). The next section shows the colorings that we have found for two proteins by following this methodology.

4.7 Results

We have analyzed 2 proteins - checkpoint kinase 1 (CHK1) and phosphoinositide-dependent protein kinase 1 (PDK1). The choice of these two proteins was entirely based on the availability of the information required for the analysis. The following sections list the steps described in Section 4.2 for each of the proteins: step 1 gives the names of the PDB entries used to identify the residues at the active and allosteric sites; step 2 lists the residues before and after the application of SWISS-MODEL, as aligned by Pymol; step 3 outputs the user-specified energy cutoff and a summary of the results from FIRST; step 4 shows the dimensions of the computed rigidity matrix and its null space, and step 5 displays the discovered relevant bodies and illustrates the results via three protein colorings.

4.7.1 CHK1

Checkpoint kinase 1 (CHK1) is a key element in the DNA damage response pathway and plays a crucial role in the S-G2-phase checkpoint [34]. Inhibiting CHK1 is a therapeutic strategy involving abrogation of the G2/M mitotic checkpoint defense of tumor cells toward lethal damage induced by DNA-directed chemotherapeutic agents.

1. **Literature search** - The protein structure that we use to identify the residues forming the allosteric site appears as 3JVR in the Protein Data Bank. It is the CHK1 protein bound to the ligand AGX. For the active site we use the protein structure with PDB code 1NVQ, which is CHK1 bound to the ligand UCN at the ATP-binding pocket.

2. **SWISS-MODEL** - Table 4.1 lists the IDs of the residues at the active and allosteric sites of the protein before (Before SM) and after (After SM) it has been run through SWISS-MODEL. Note that the residue IDs are shifted by 1. This shift results from the computation performed by SWISS-MODEL and does not affect our analysis.

Active Site Residues		Allosteric Site Residues	
Before SM	After SM	Before SM	After SM
Cys87	Cys86	Phe93	Phe92
Leu15	Leu14	Pro133	Pro132
Gly16	Gly15	Leu206	Leu205
Glu17	Glu16	Ala200	Ala199
Tyr20	Tyr19	Asp94	Asp93
Val23	Val22	Arg95	Arg94
Ala36	Ala35	Ile96	Ile95
Lys38	Lys37	Glu97	Glu96
Glu55	Glu54	Pro98	Pro97
Val68	Val67	Tyr173	Tyr172
Leu84	Leu83	Gly204	Glu203
Glu85	Glu84	Glu205	Glu204
Tyr86	Tyr85		
Ser88	Ser87		
Gly90	Gly89		
Glu91	Glu90		
Lys132	Lys131		
Glu134	Glu133		
Asn135	Asn134		
Leu137	Leu136		
Ser147	Ser146		
Asp148	Asp147		

Table 4.1: IDs of residues at the active and allosteric sites in CHK1 (RMSD=0.061)

3. **FIRST** output summary

Energy cutoff	-0.500 kcal/mol
Rigid Clusters	685
Hinges	722
Bars	170

4. **Infinitesimal Rigidity**

Dimensions of rigidity matrix	7485×4110
Dimension of null space	410

5. **Analysis**

5.1 *Identifying relevant bodies* - note that we present only some of the bodies found to contain atoms from the active and allosteric sites. A pair $[i \ j]$ denotes that body i contains j number of atoms from that site.

Active Site	Allosteric Site
[1 171]	[1 124]
[16 10]	[18 10]
[12 10]	[3 6]
[2 6]	[187 4]
[137 4]	[186 4]
[136 4]	[184 4]
[112 4]	[116 4]
[111 4]	[115 4]
[69 4]	[412 3]
[57 4]	[389 3]
[56 4]	[326 3]
[54 4]	[325 3]
[53 4]	[324 3]

5.2 *Identifying relevant null space vectors* - following the methodology described in Section 4.6, we identified a set of relevant null space vectors and the colorings that we discuss are produced from some of them. In Figures 4.8-4.10, the solid square frames show the location of the active site in 3JVR, the circular frames show the location of the allosteric site,

and the dashed rectangular frames surround the regions that are zoomed in at the bottom picture of the figures. Discussion for each coloring is provided in the caption of the figures. Note that parts of the bottom picture in Figure 4.10 are purposefully faded to illustrate a potential “path” between the active and allosteric sites.

4.7.2 PDK1

Protein kinases are involved in the control and regulation of cellular processes [30]. Their overexpression or loss of regulatory mechanisms are observed in many diseases such as cancer, Alzheimer’s disease and type 2 diabetes. The phosphoinositide-dependent protein kinase 1 (PDK1) is in the center of growth factor and insulin signaling and is a master kinase that phosphorylates the activation loop of several protein kinases from the AGC group [30].

1. **Literature search** - The protein structure that we use to identify the residues forming the allosteric site appears as 3HRF in the Protein Data Bank. It is the PDK1 protein bound to the ligand P47 at the PIF-pocket. For the active site, we use the protein structure with PDB code 3IOP, which is PDK1 bound to the ligand 8I1 at the ATP-binding pocket.
2. **SWISS-MODEL** - Table 2 lists the numbers of the residues at the active and allosteric sites of the protein before (Before SM) and after (After SM) it has been run through SWISS-MODEL. Note that the residue IDs are shifted by 48. This shift results from the computation performed by SWISS-MODEL and does not affect our analysis.

Active Site Residues		Allosteric Site Residues	
Before SM	After SM	Before SM	After SM
Lys86	Lys38	Lys76	Lys28
Leu88	Leu40	Lys115	Lys67
Gly89	Glu41	Ile118	Ile70
Glu90	Glu42	Ile119	Ile71
Gly91	Gly43	Val124	Val76
Ser94	Ser46	Val127	Val79
Val96	Val48	Thr128	Thr80
Ala109	Ala61	Arg131	Arg83
Lys111	Lys63	Thr148	Thr100
Glu130	Glu82	Phe149	Phe101
Val143	Val95	Gln150	Gln102
Leu159	Leu111	Leu155	Leu107
Ser160	Ser112	Tyr156	Tyr108
Tyr161	Tyr113	Phe157	Phe109
Ala162	Ala114		
Lys163	Lys115		
Asn164	Asn116		
Gly165	Gly117		
Glu166	Glu118		
Lys169	Lys121		
Lys207	Lys159		
Glu209	Glu161		
Asn210	Asn162		
Leu212	Leu214		
Thr222	Thr224		
Asp223	Asp225		

Table 4.2: IDs of residues at the active and allosteric sites in PDK1 (RMSD=0.071)

3. **FIRST** output summary

Energy cutoff -1 kcal/mol

Rigid Clusters 642

Hinges 758

Bars 120

4. Infinitesimal Rigidity

Dimensions of rigidity matrix 3850×3852

Dimension of null space 417

5. Analysis

5.1 *Identifying relevant bodies* - note that we present only some of the bodies found to contain atoms from the active and allosteric sites. A pair $[i\ j]$ again denotes that body i contains j number of atoms from that site.

Active Site	Allosteric Site
[1 261]	[1 142]
[13 10]	[75 4]
[82 4]	[74 4]
[77 4]	[72 4]
[76 4]	[64 4]
[50 4]	[63 4]
[41 4]	[62 4]
[40 4]	[61 4]
[38 4]	[60 4]
[37 4]	[58 4]
[34 4]	[57 4]
[324 3]	[56 4]
[323 3]	[55 4]

5.2 *Identifying relevant null space vectors* - following the methodology described in Section 4.6, we identified a set of relevant null space vectors and the colorings that we discuss are produced from some of them. In Figures 4.11-4.13, the solid square frames show the location of the active site in 3HRF, the circular frames show the location of the allosteric site, and the dashed rectangular frames surround the regions that are zoomed in at the bottom picture of the figures. Discussion for each coloring is provided in the caption of the figures. Note that parts of the bottom

picture in Figure 4.13 are purposefully faded to illustrate a potential “path” between the active and allosteric sites.

Considerations and future work

1. In both MATLAB and Java, comparisons with zero require special consideration. Due to numerical approximations, results that would normally be zero are instead very close, but not equal, to zero. In order to perform comparisons, we use “precision bubbles” with a certain radius r . Thus, we consider a number to be 0 if that number is a distance less than or equal to r from 0. Following are values for the radius of each precision bubble used in the MATLAB and Java computation.

computation	radius
comparing Plücker coordinates to 0 (MATLAB)	10^{-13}
comparing singular values to 0 (MATLAB)	10^{-10}
comparing the difference between velocity vectors to 0 (Java)	10^{-2}

2. The infinitesimal rigidity analysis that we conduct is based on the model produced by FIRST. As we mentioned in Section 4.1, FIRST uses chemical interactions between atoms in order to model the protein as a body-bar-hinge structure. It represents covalent bonds by hinges and hydrogen bonds by 2 to 5 bars. There are, however, the so called hydrophobic interactions reflecting the “preference” of certain parts of the molecule for water. These interactions introduce additional constraints to the model and in order to account for them, FIRST creates *hydrophobic tethers* and represents them as single bars. This imposes certain assumptions which have to be taken into account in the analysis.

Furthermore, the bars representing each hydrogen bond are identical from our perspective since FIRST considers only the combinatorial characteristics of the structure without specifying its geometry explicitly. Thus, we have yet to determine an appropriate representation of the hydrogen bonds when using the body-bar-hinge model for a protein.

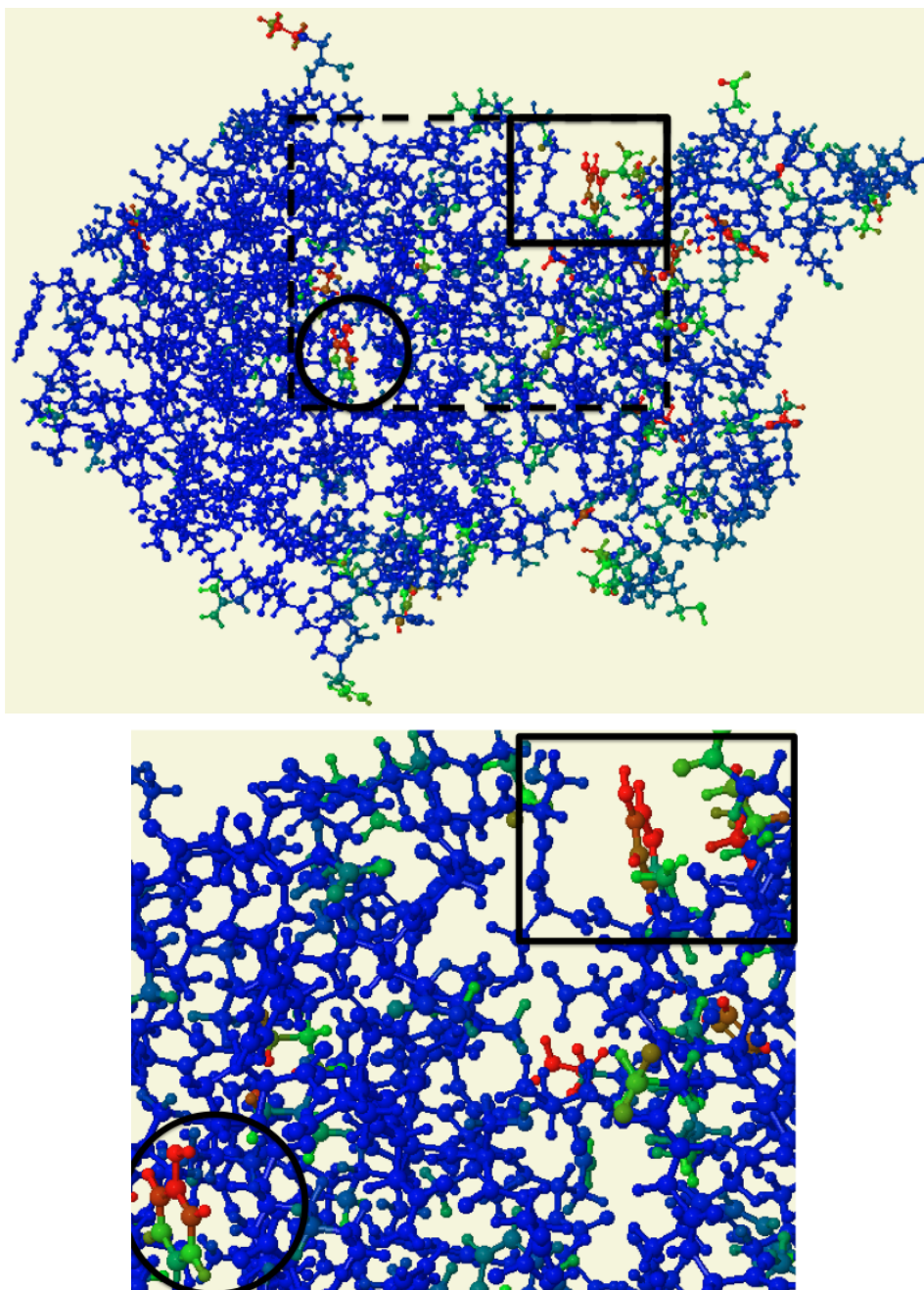


Figure 4.8: (**protein 3JVR; null space vector applied: 302**) In this coloring, both the active and allosteric sites are flexible while the rest of the protein seems mostly rigid. Thus, we cannot make any conclusions about the relationship between the two sites.

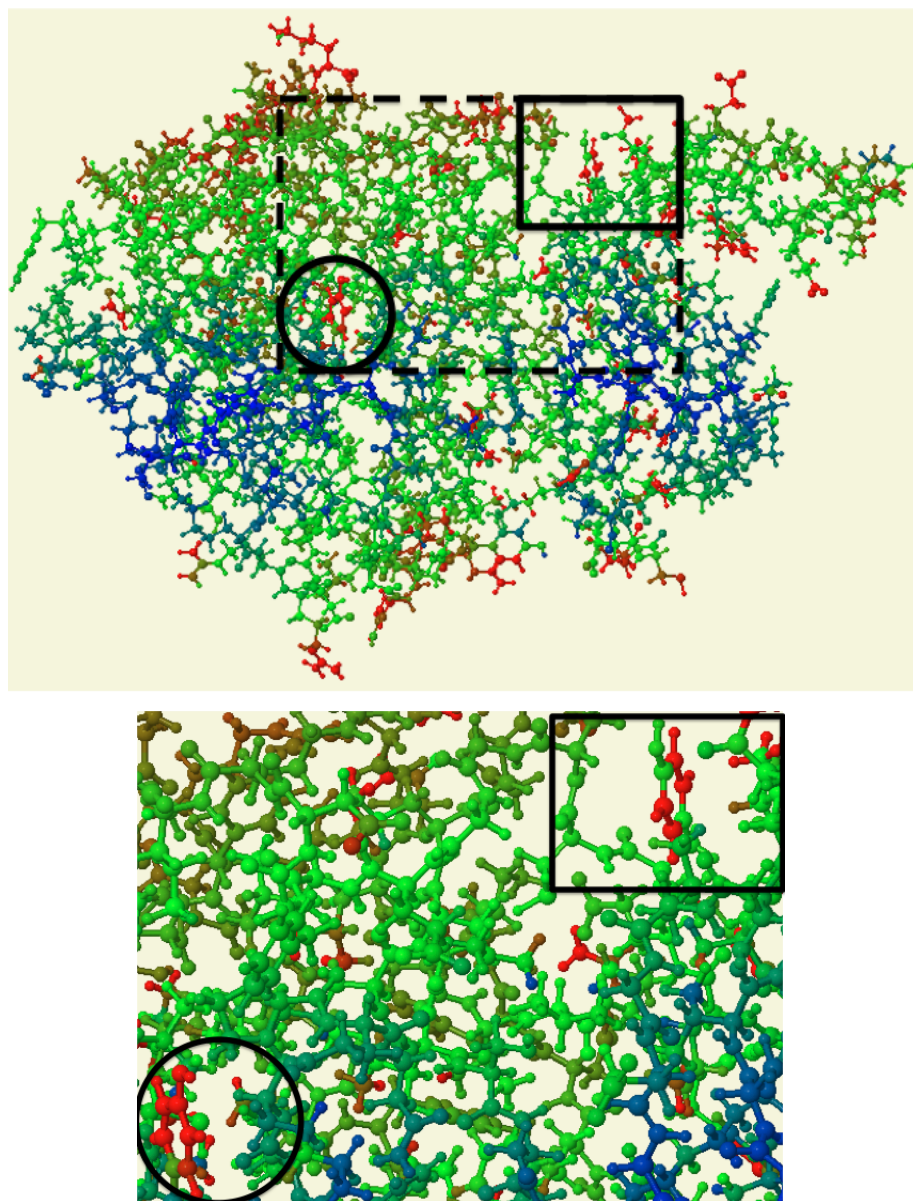


Figure 4.9: (protein 3JVR; null space vector applied: 396) Here we can see again that the active and allosteric sites are flexible. However, most of the atoms in the rest of the protein are also flexible and thus, we cannot justify the existence of a relationship precisely between the two sites.

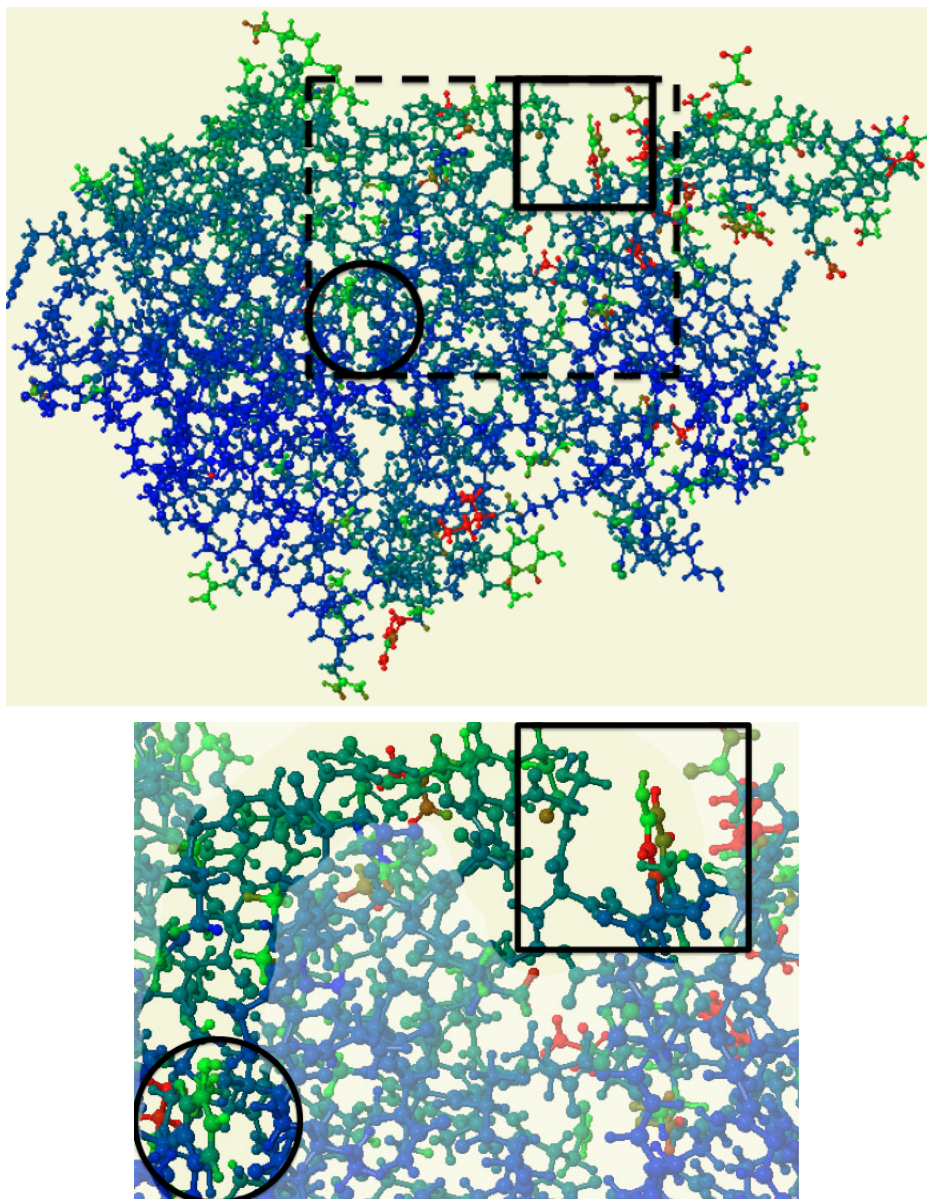


Figure 4.10: (**protein 3JVR; null space vector applied: 6**) In this coloring most of the protein is either rigid or slightly flexible. However, the magnitudes of the velocities of the atoms at the active and allosteric sites are larger, and, if we look carefully, we can identify a potential “path” that leads from one of the sites to the other, as illustrated in the bottom picture

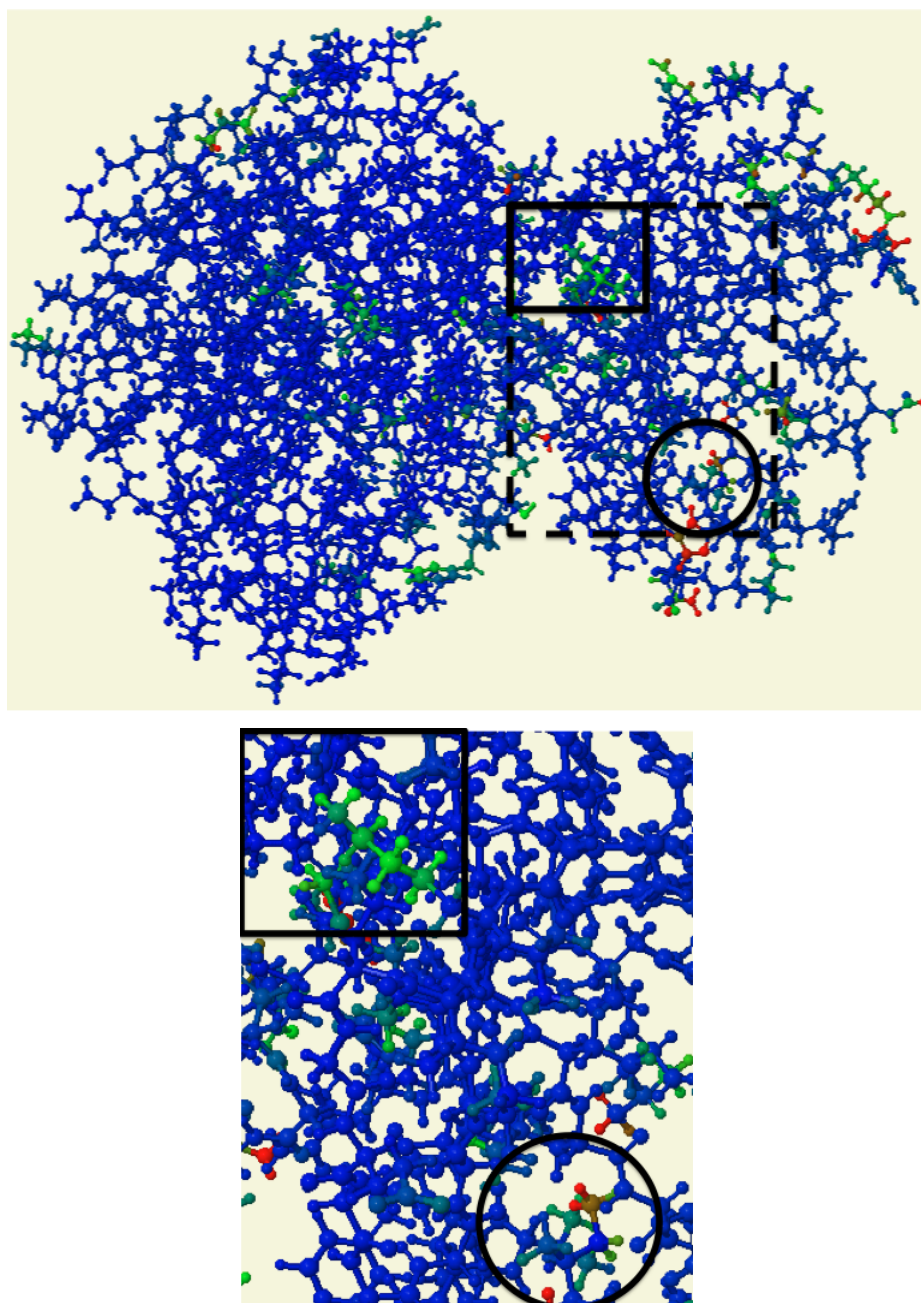


Figure 4.11: (**protein 3HRF; null space vector applied: 3**) In this coloring most of the protein is rigid while some of the atoms at the active an allosteric sites have velocities different than 0. Thus, we cannot make any conclusions about the relationship between the two sites.

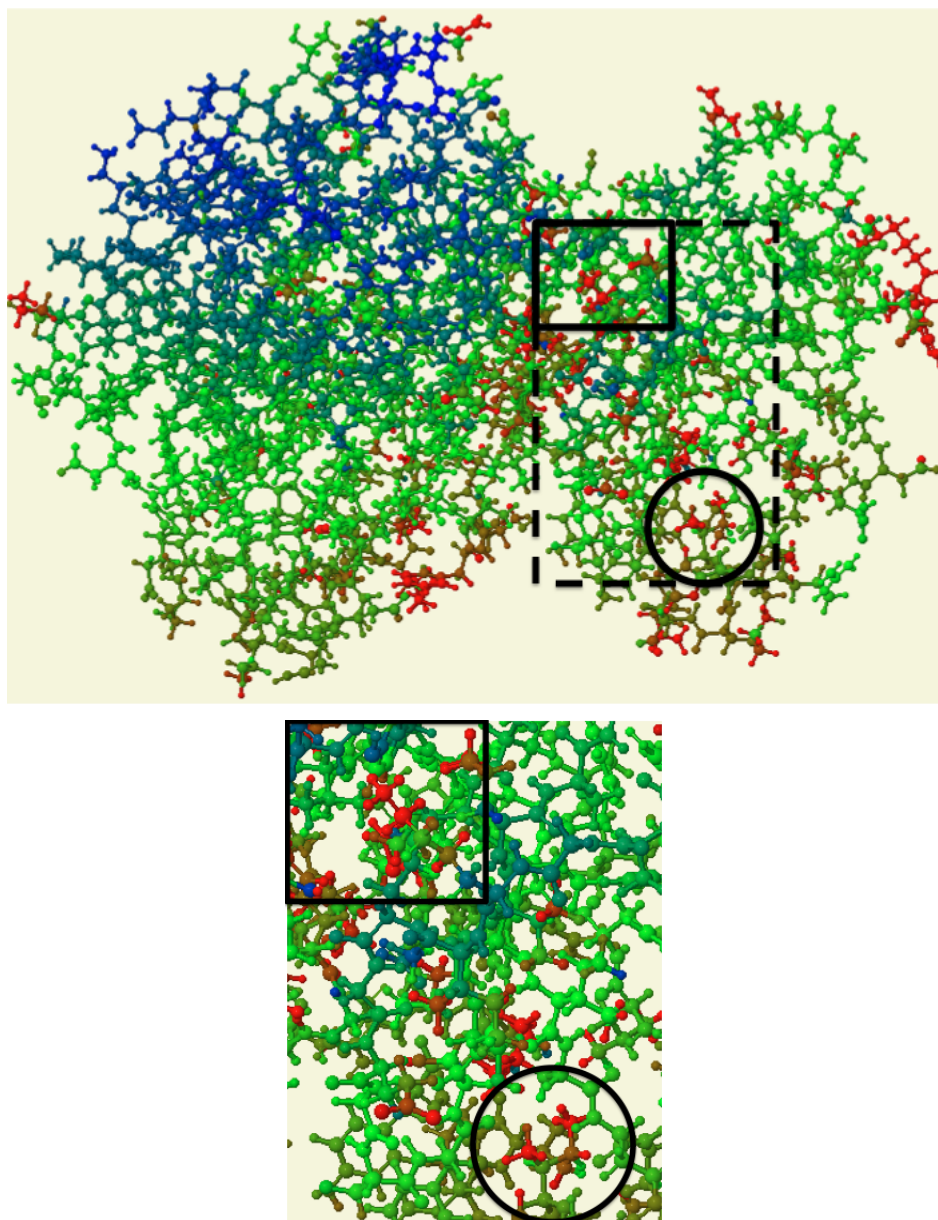


Figure 4.12: (**protein 3HRF; null space vector applied: 337**) Here we can see again that the active and allosteric sites are flexible. However, most of the atoms in the rest of the protein are also flexible and thus, we cannot justify the existence of a relationship precisely between the two sites.

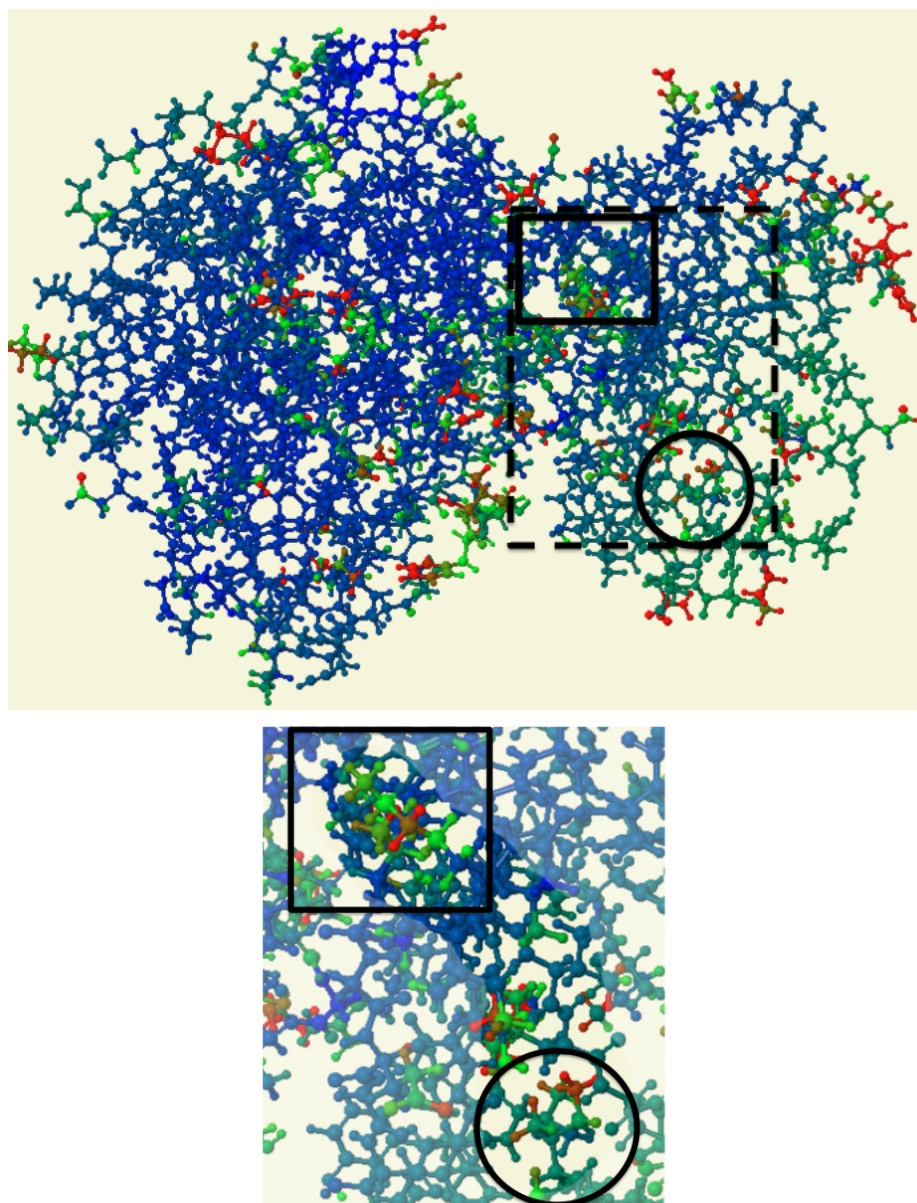


Figure 4.13: (**protein 3HRF; null space vector applied: 375**) In this coloring most of the protein is either rigid or slightly flexible. However, the magnitudes of the velocities of the atoms at the active and allosteric sites are clearly larger, and we can identify a potential “path” that leads from one of the sites to the other. Note that the structure is 3-dimensional and the part of the path that seems to go through “blue” atoms actually goes behind them.

Chapter 5

Infrastructure Design for Motion Simulation Software

Conformational changes are essential to macromolecular function. As we saw in Chapter 4, performing static analysis may give us intuition as to why some of these changes occur. However, it cannot tell us how exactly they happen, and therefore we study the dynamics, or motion, of macromolecules. Due to the limitations of actual experimental observations, macromolecular motion is simulated via standard computational approaches such as MD, as explained in Section 1.2.2.

Unfortunately, macromolecules are typically composed of tens of thousands of atoms, making these standard computations very expensive to perform on the timescale in which “interesting” motions occur. As a result, novel techniques are being developed to improve simulation efficiency. The implementation of most of these techniques, however, is not standardized. Different pieces of code are written not necessarily in agreement with certain software engineering rules, which makes code reuse or modification very difficult. Thus,

there is no effective method for comparing the performance of various approaches. Having the ability to perform such comparisons could improve the development of new techniques and increase the efficiency of already existing ones.

In this chapter, we present a design for an infrastructure that would assist scientists in the application and comparison of motion simulation techniques. By following standard software engineering principles, the infrastructure allows for flexibility in programming and developing various combinations of methods. Its design is motivated by simulation approaches, such as FRODA and ROCK, which simulate motion by first, “moving” the system, potentially violating constraints, then iterating to solve the constraints and resolve collisions [35, 23].

We should note, however, that this is preliminary work. The design has not been implemented at this point, and its goal is mainly to establish the foundation for future research in this direction.

The infrastructure (or, *library*, as we refer to it later) consists of two parts: **data representation** and **motion simulation**. The subsequent sections describe the structure of each of these parts and present a discussion on the benefits and limitations of the overall design. The organization of the library is visualized via standard UML diagrams in which arrows with triangular arrowheads, “ \rightarrow ”, represent “is-a” relationships, and arrows with regular arrowheads, “ \longrightarrow ”, represent “has-a” relationships between classes. We also specify the cardinality restrictions on the relationship lines by denoting 0 or more instances by a star (*), and 1 or more instances by a plus sign (+).

Motivating example

To help the reader follow the design decisions, consider the context of protein motion as an example to which we will refer throughout the chapter.

A protein has *primary* and *secondary* structure elements as shown in Figure 5.1. The primary structure is composed of a sequence, or “backbone,” of amino acids (residues), while the secondary structure elements can be *alpha helices* or *beta sheets*. When the protein folds into its 3-dimensional structure, alpha helices and beta sheets are formed to maintain structural integrity and could be approximated as rigid sub-structures. Therefore, coarse-grained and fine-grained approaches can be used to simulate protein behavior. A

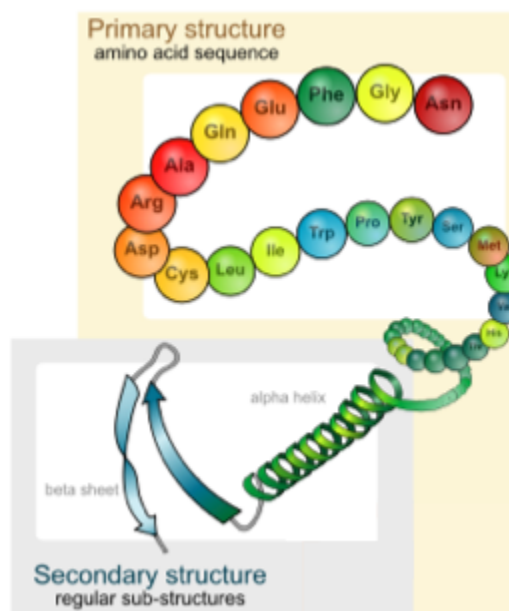


Figure 5.1: Primary and secondary structure of a protein. Adapted from <http://commons.wikimedia.org/>

course-grained method simulates motion on the level of alpha helices and beta sheets, while fine-grained techniques “operate” on the level of residues or atoms. Each amino acid has a side chain with distinct chemical characteristics as explained in Chapter 1. Some motion simulation methods, for instance, focus first on the position of the backbone (ignoring the amino acid sidechains), then zoom in to place the sidechains once a general structure has been achieved.

To accommodate a simulation of the protein that would allow for an interchange of coarse-grained and fine-grained approaches, we present a design that offers the ability to rigidify substructures (e.g., sidechains) and later “undo” the process.

5.1 Data representation

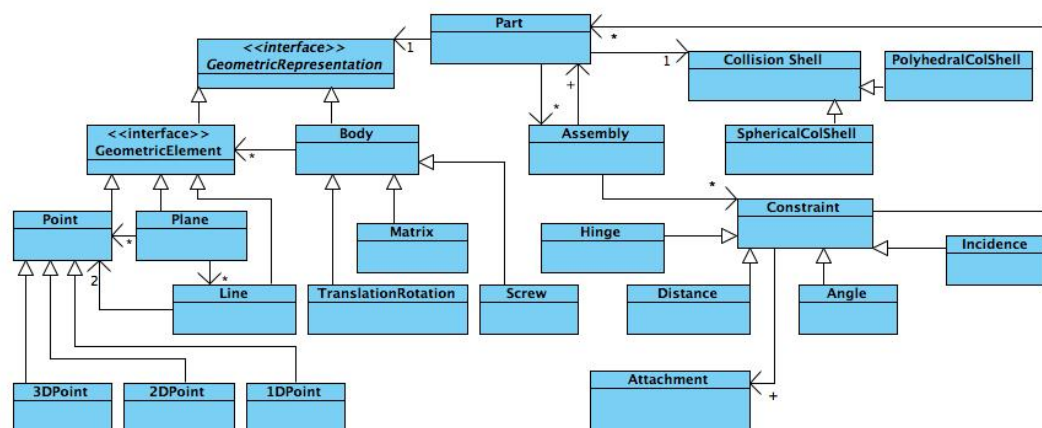


Figure 5.2: Data representation for the infrastructure

In order to apply motion simulation techniques to macromolecules, we first need to model the structures. In fact, we choose a general representation that would accommodate other applications, such as those coming from Computer Aided Design (CAD).

The data representation design that we present is based on two main constructs called **part** and **assembly**. A **part** represents a rigid component whose motion space consists only of the trivial deformations. An **assembly**, on the other hand, is a flexible structure composed of **parts** with constraints between them. The motion of an **assembly** in space is restricted by the constraints between its **parts**.

5.1.1 Geometric representation

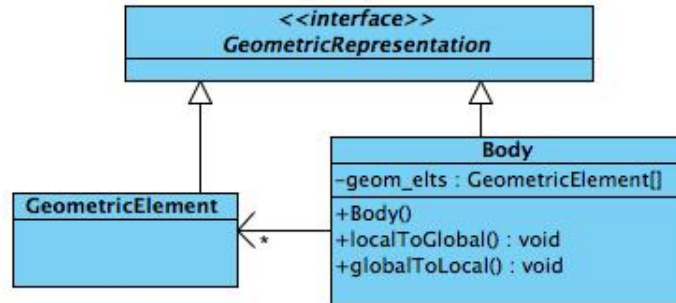


Figure 5.3: GeometricRepresentation class and its children

Each **part** can be defined geometrically as either a geometric element (i.e., a point, line, or plane), or a body. We distinguish between them because a body may contain geometric elements. Therefore, there are 2 separate classes, **GeometricElement** and **Body**, that inherit **GeometricRepresentation** as shown in Figure 5.3.

GeometricRepresentation: Geometric Element

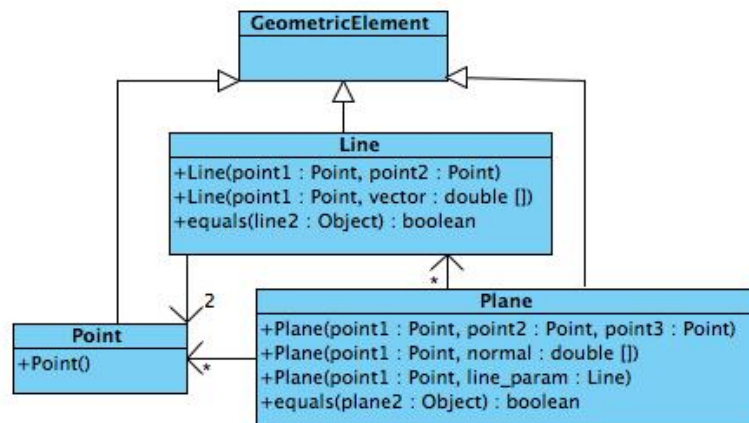


Figure 5.4: GeometricElement class and its children

The class `GeometricElement` has three children, `Point`, `Line` and `Plane` as shown in Figure 5.4. Each represents a geometric element with a specified position in space. A `line` can be defined by `points`, and a `plane` can be defined by `lines` and `points`. Therefore, there exist “has-a” relationships between `Line` and `Point`, `Plane` and `Line`, and `Plane` and `Point`. Note that both `Line` and `Plane` have a method called *equals* that determines whether two `lines` or two `planes` are the same, due to the infinite number of their identical geometric representations.

GeometricElement: Point

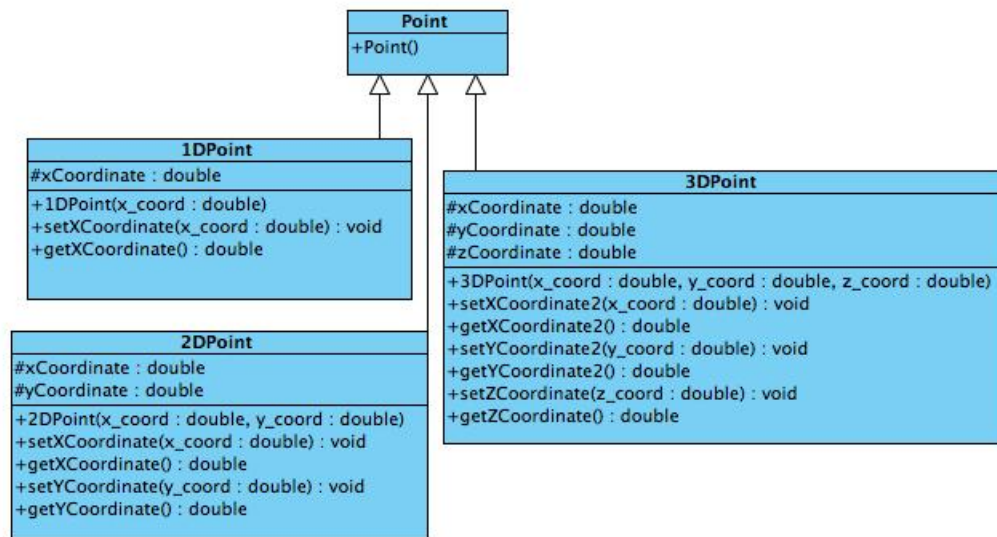


Figure 5.5: Point class and its children

Figure 5.5 illustrates the structure of the `Point` class and its children. There are different designs for this class. Our choice is based on the fact that a d -dimensional point is a `point` and thus, `1DPoint`, `2DPoint` and `3DPoint` inherit directly the `Point` class.

We considered an alternative design, where `1DPoint` inherits `Point`,

`2DPoint` inherits `1DPoint`, and `3DPoint` inherits `2DPoint`. This representation reflects the idea that a 2-dimensional point is a 1-dimensional point with an additional coordinate, and a 3-dimensional point is a 2-dimensional point with an additional coordinate. Although not entirely sound from a software engineering perspective, this hierarchical design avoids the specification of extra data and methods and should be considered for potential efficiency benefits.

Note that in the actual implementation it may be more practical if the type of each dimensional variable (x , y , z) is declared as generic, instead of double. This representation would allow for an easy switch between a double and a pointer to a double, for instance.

GeometricRepresentation: Body

A **body** can be defined by either a transformation matrix, Euler translation and rotation vectors, or a screw axis with a translation vector. The structure of the class is shown in Figure 5.6.

Since a body does not have an explicitly specified shape, we can intuitively think of it as a frame in space. As such, it has local and global coordinates. The local coordinates specify the position of the body's elements in relation to the frame, while the global coordinates relate them to the larger system. A **body** can “switch” between its local and global coordinates via the methods *convertToGlobalCoord* and *convertToLocalCoord*.

5.1.2 Part and Assembly

As we mentioned at the beginning of the chapter, a **part** in the infrastructure represents a rigid component, while an **assembly** is flexible and consists of **parts** with constraints between them.

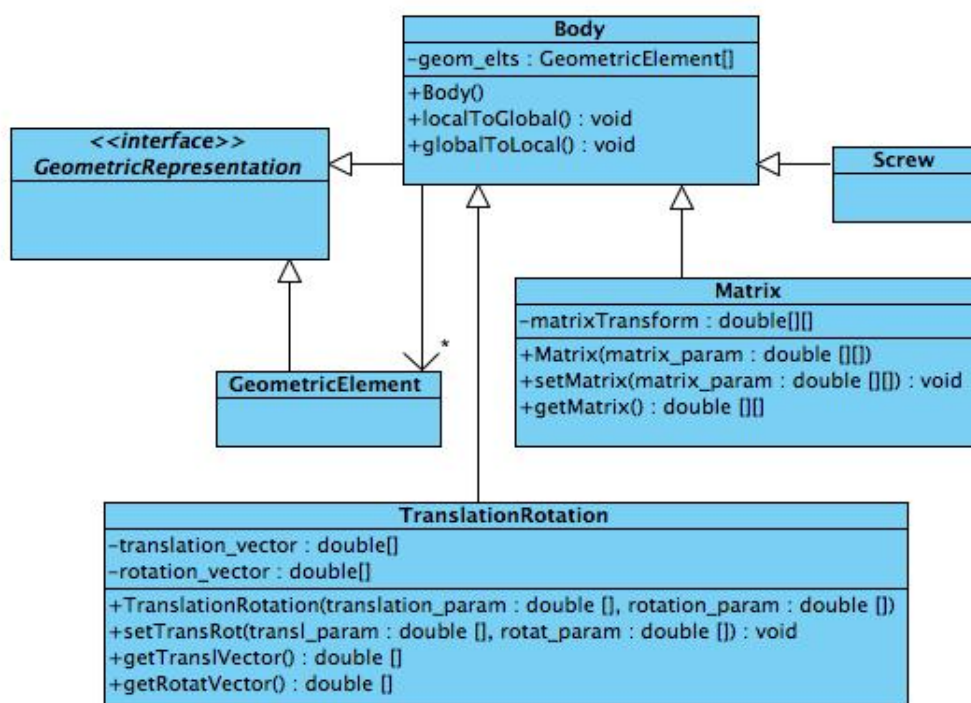


Figure 5.6: Body class and its children

Each **part** has a **geometric representation** and a **collision shell**, and may contain a set of **assemblies**. If the **geometric representation** of a **part** is a single **point**, for instance, the **part** does not have any **assemblies**, while if it is a **body**, the **part** may consist of multiple **assemblies**.

Intuitively, we can think of the **collision shell** of a **part** as the “space” that is occupied by the **part**, in which the presence of another object would cause a collision. In our protein example, the **collision shell** for each atom would be defined by its van der Waal radius (i.e., the radius of a sphere that models the atom).

The **CollisionShell** class is a general class that could should be subclassed, for instance as depicted to represent a spherical or polyhedral shape. It has a **tolerance** attribute that defines the level of flexibility of the collision

shell in handling collisions with other objects.

The “has-a” relationship between **Part** and **Assembly** allows a flexible subset of the **assembly** to be rigidified at any time during the motion simulation by converting it to a rigid **part**. As illustrated by the motivating example, this design choice is necessary to accommodate for different motion simulation techniques.

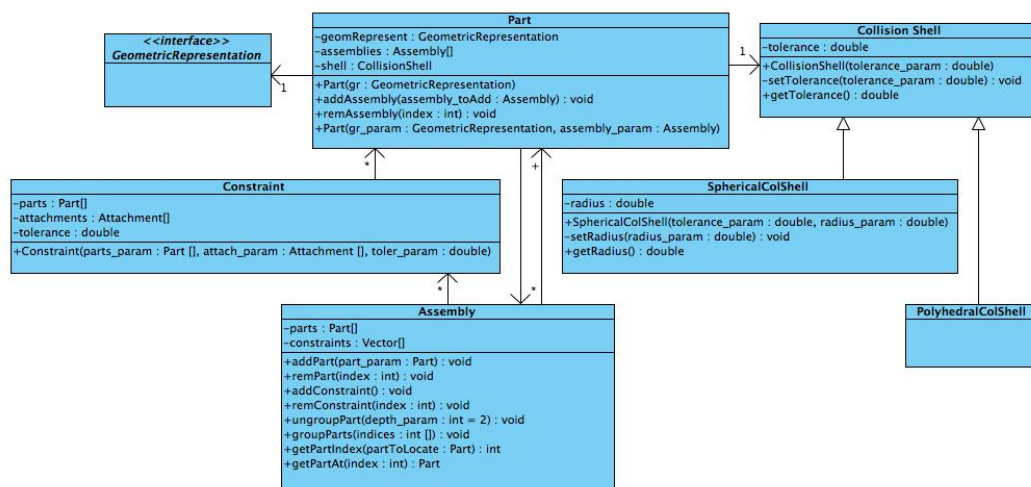


Figure 5.7: Structure of the Part class

An **assembly** has a set of **parts** and a set of **constraints** between these **parts**. A **constraint** and a **part** can be added or removed from the **assembly** via the corresponding methods *addConstraint*, *remConstraint*, *addPart* and *remPart*.

An **assembly** also has the operations *ungroup* and *group*. Ungrouping a **part** in the **assembly** is equivalent to modifying the **assembly** by dividing the **part** into smaller **parts** and adding them to the **assembly**. Via ungrouping, for instance, an alpha helix or a beta sheet can be “broken up” into its residues, then into its atoms. Thus, a course-grained approach in simulating protein motion could be refined to a more fine-grained one.

Due to the possibility of multiple **parts** nested in each other, a **geometric element** may be contained in more than one **part**. This requires the addition of a depth parameter specifying how many “levels down” we want to go when ungrouping a **part** (the default value being 2). This whole process is accompanied by a change in the constraints list. The user may choose to keep or remove some of the already existing **constraints** (this will be explained in more detail in the next section).

Grouping a set of **parts** in an **assembly** is equivalent to rigidifying a subset of the **assembly**. In order to do this, the user has to perform the following steps:

1. Create a new **assembly** from the **parts** and the **constraints** between them,
2. Make a new **part** from the new **assembly** and set its geometric representation to be a **body** with transformation matrix the identity matrix, and
3. Add this new **part** to the already existing **assembly**.

It is essential that when a non-level 0 **part** (i.e., a **part** containing **assemblies**) is created, the **body** transformation matrix is set to be the identity. This ensures that the **geometric representation** of a **part** is always local with respect to the parent **part**. Thus, in the case of a nested hierarchy, the global coordinates of a **geometric element** can always be calculated by multiplying the transformation matrices of all nested **bodies**.

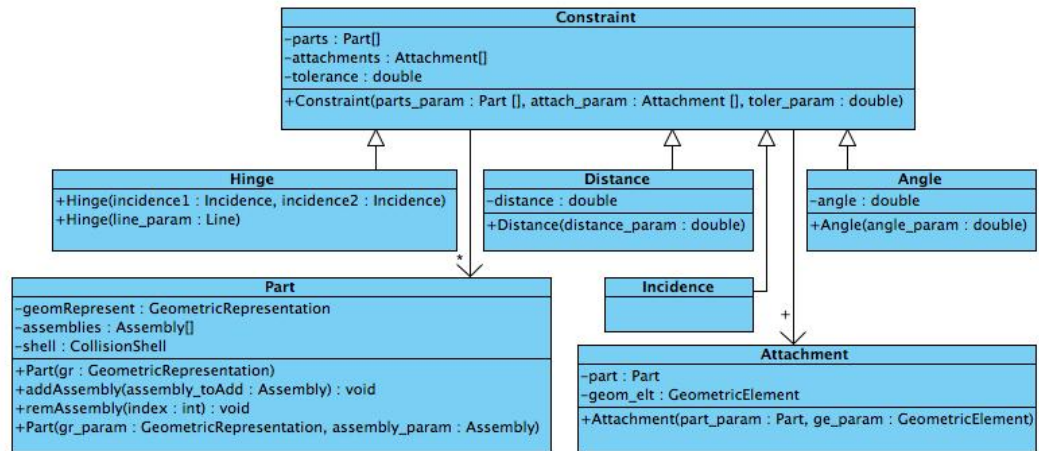


Figure 5.8: Constraint class and its children

Part and Assembly: Constraints and Attachments

The parts in an assembly have **constraints** between them that attach to the **parts** at special **attachment** objects. Consider, for instance, body-bar-hinge structures. Rigid bodies in these structures are constrained by bars or hinges. Each bar imposes a distance constraint between two bodies and attaches to each of them at a single point. A hinge allows only one rotational degree of freedom and can be viewed as a unique line “connecting” the bodies. Thus, we say that a bar **constraint** has two **point attachments**, while a **hinge** has one **line attachment**.

The **Constraint** class has an array of **attachments** as shown in Figure 5.8. It also has a **tolerance** attribute that defines how “strict” the constraint is. There are four different types of constraints - **distance**, **angle**, **incidence**, and **hinge**. The **distance** and **angle** constraints specify the distance and angle between **parts**; an **incidence** constraint is a **geometric element** contained by each of the **parts**, and a **hinge** can be represented by a **line** or two **incidence** points.

Each `constraint` attaches to a `part` via an `attachment` object. An `attachment` object has a `part` and a `geometric element`. The `geometric element` is the actual attachment point, line or plane. The `Part` attribute specifies which `part` (if any) this `attachment` should be associated with if the current `part` was to be ungrouped. It can be viewed as a way of declaring whether a `constraint` should be maintained or removed after the `part` with which it is associated is ungrouped.

5.1.3 Discussion

Our data representation design allows for the creation of structures containing rigid and flexible components, as well as for an easy “transformation” between them (i.e., grouping and ungrouping parts, thus rigidifying flexible components or making rigid components flexible). This feature of the design plays an important role in the motion simulation of macromolecules such as proteins.

Our design can accommodate various choices for geometric elements and body representations as well and can be extended to include different types of constraints and collision shells. However, it presents certain limitations as discussed below.

Representing Constraints in the `Assembly` class

Each `assembly` consisting of n `parts` has an array with n elements of type `Part`. The `constraints` between these `parts` are represented by an array of n vectors in which the i -th element contains all `constraints` associated with the i -th `part`. Since ungrouping or grouping `parts` may involve the removal of certain `constraints`, this representation may turn out to be computationally expensive.

Another possibility would be for each **part** to keep track of the **constraints** associated with it. However, if two assemblies share a **part** that has **constraints** with **parts** from both **assemblies**, this representation would not be more efficient than the one proposed.

Creating new Parts

A new **part** can be created explicitly only if the **part** is at level 0, i.e., it does not contain any **assemblies**. At the same time, a **part** can be created implicitly with the method *groupParts* in the **Assembly** class. This representation ensures that a **part** is always in a nested hierarchy, and thus, cannot be in more than one non-nested **bodies**. In order to allow for a part to be in more than one non-nested bodies, the design needs to incorporate one of the following:

- A **geometric element** keeps track of all of its local coordinates with respect to its parents.
- A parent keeps track of all its **geometric elements'** local coordinates.
- A convenience method for auto-creating an explicit **constraint** from an implicit **incidence**, i.e., forcing duplications and adding **incidence constraints**.

5.2 Motion simulation

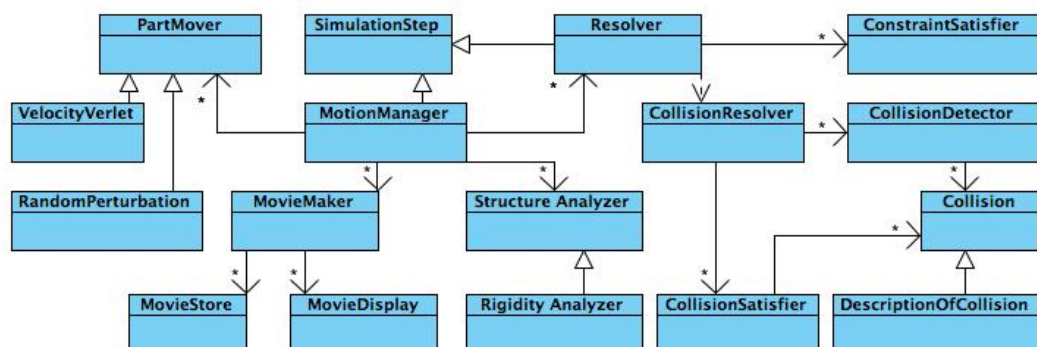


Figure 5.9: Motion Simulation part of the infrastructure

The motion simulation part of the infrastructure assumes the data representation design described in Section 5.1. It presents a library of classes whose implementation would allow for the simulation of macromolecular motion or, more generally, motion of constrained structures.

The basic design is shown in Figure 5.9. There is a **SimulationStep** class that allows for iteration over a number of steps, and a **MotionManager** class that inherits **SimulationStep** and controls the overall simulation. The motion that can be modeled with this library follows a number of steps including *structure analysis*, *motion generation*, *collision detection and resolution*, and *constraint satisfaction*. The subsequent sections describe the classes associated with these steps.

5.2.1 SimulationStep and Motion Manager

SimulationStep

The class **SimulationStep** (as illustrated in Figure 5.10) contains a sequence of operations that define a single step of the motion simulation process. Each

operation corresponds to a class in the infrastructure, and hence **sequence** is defined as a vector of objects. We could also represent **sequence** as a vector of numbers where each object is identified by a number. However, this representation would be more inefficient because the user would have to assign new numbers to classes added at a later time; this would result in many checks to find the object associated with a particular number.

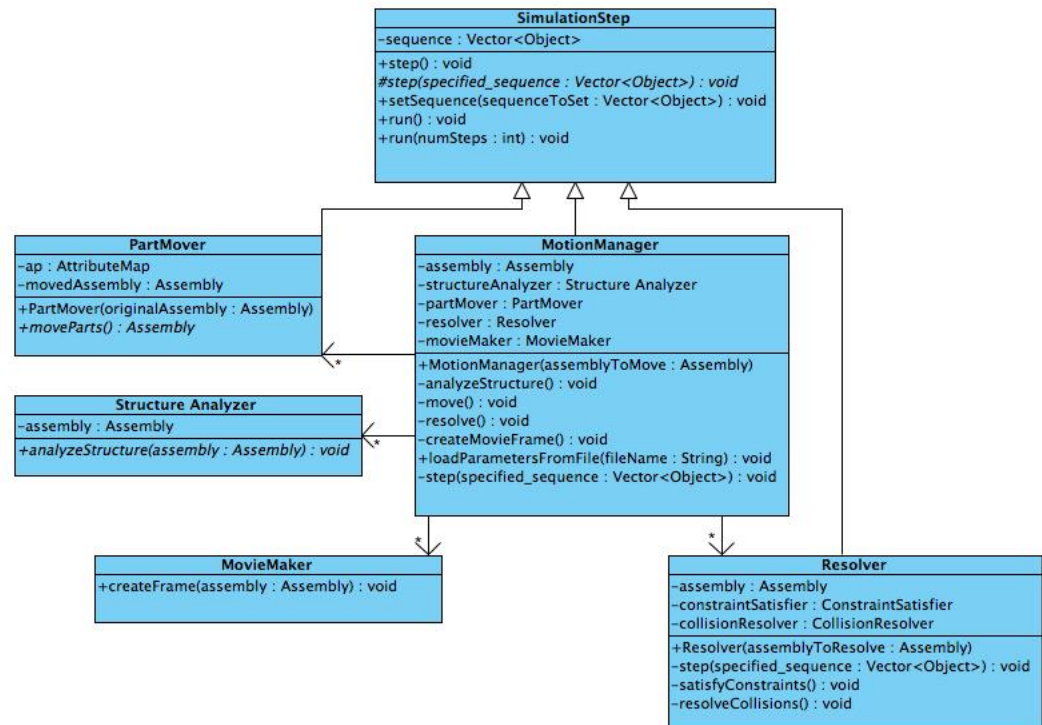


Figure 5.10: SimulationStep, Motion Manager and their children

Besides the sequence of operations, **SimulationStep** has methods facilitating the implementation of the sequence. The method `step` (no parameters) executes a single step of the simulation. The method `step(specified_sequence)` is abstract (to require the children of **SimulationStep** to provide their own implementations of the method), and protected (to prevent unauthorized classes from calling it directly). Two additional methods, `run` and `run(numSteps)`,

execute *step* an infinite or specified number of times, respectively.

The class `SimulationStep` is inherited by `MotionManager`, `Resolver` and `PartMover`.

MotionManager

`MotionManager` controls the overall motion simulation. It has a `PartMover`, `StructureAnalyzer`, `Resolver` and `MovieMaker`, and can “execute” them in any order since it inherits `SimulationStep`. One step may consist of analyzing the structure, moving it, resolving collisions, satisfying constraints and producing a movie frame. The class also has a method that loads parameters from a given file.

5.2.2 StructureAnalyzer

`StructureAnalyzer` has an abstract method, *analyzeStructure*, that is implemented by each of its children. The purpose of the class is to give the option of analyzing and modifying the structure of an **assembly**. For instance, the user can modify an **assembly** by ungrouping or grouping some of its **parts**. An example would be an algorithm like FIRST, which may identify and group rigid components.

5.2.3 PartMover

`PartMover` (also shown in Figure 5.10) should be subclassed to define different algorithms for motion simulation, such as velocity Verlet and random perturbation. It moves the structure using one of these algorithms, which may result in an embedding that violates constraints or has collisions.

5.2.4 Resolver

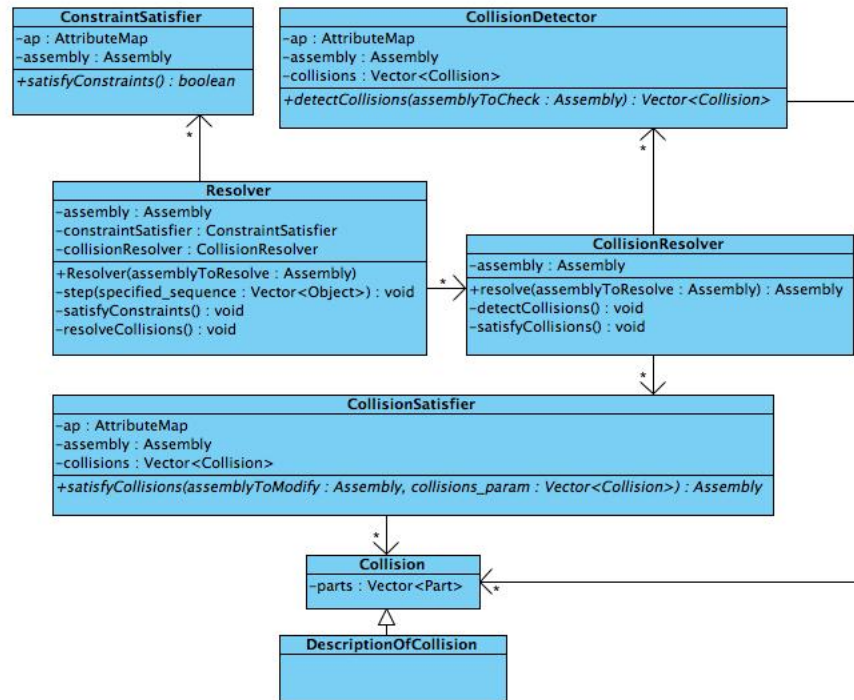


Figure 5.11: Resolver

After the structure has moved, **Resolver** checks for collisions and makes sure the constraints are satisfied. It has a **ConstraintSatisfier** and a **CollisionResolver** as shown in Figure 5.11, and controls the order and number of their executions since it inherits **SimulationStep**.

5.2.5 MovieMaker

MovieMaker creates movie frames. Since the user may want to save and/or display these frames, **MovieMaker** has a **MovieStore** and a **MovieDisplay** classes, each of which has children that implement different methods for storing and displaying frames.

5.2.6 Discussion

The most significant benefit of our motion simulation design is its flexibility. The library allows for the execution of different sequences of simulation steps and supports step repetition; it can be easily extended to include more algorithms for motion generation, as well as for collision detection and constraint satisfaction, and we can even change the structure of the system whose motion we are simulating by implementing the **StructureAnalyzer** class. This flexibility provides the user with the opportunity to combine different algorithms and compare their performance.

The limitations of the design come from the “division” of the simulation steps. We assume that motion generation, constraint satisfaction, collision detection and resolver, and structure analyzer are all separate steps that can be executed in any order. However, it is possible to have an algorithm that combines some of these actions in a single step (for instance, satisfies constraints as it generates motion) and our design does not explicitly support this.

Chapter 6

Conclusions and Future Work

This thesis has addressed two problems: 1) **understanding the allosteric effect** in proteins, and 2) improving the development of **motion simulation software**.

The first problem has important applications in drug design. We approached it by applying concepts from computer science and mathematics to examine protein structure and its *potential* for motion. We modeled proteins as discrete structures whose movements are restricted by specific geometric constraints. This allowed us to conduct rigidity analysis on 2 protein structures, CHK1 and PDK1, and apply various tools to explore their infinitesimal motion space. Our “toolbox” was composed of matrix Lie groups, instantaneous screws and Grassmann-Cayley algebra on the theory side, and SWISS-MODEL, FIRST, MATLAB and Java on the applications side. We performed the analysis via an interactive web application and visualized the results through special colorings of the protein atoms corresponding to the atoms’ infinitesimal velocities. Our analysis shows a potential relationship between the active and allosteric sites of the protein and provides a preliminary

foundation for further research on the mechanisms of the allosteric effect.

Future work on this project would include:

- Developing a method for visualization of not only the magnitudes of the atoms' infinitesimal velocities, but also their direction (the Java program already computes the full velocity vectors).
- Automating the identification of relevant null space vectors, as well as linear combinations of them, that produce colorings of the protein revealing a relationship between the active and allosteric sites.
- Identifying and analyzing more protein candidates and collaborating with biologists to verify the produced results.
- Developing a sound computational method for building the rigidity matrix that would account for the double representation of bars in the rigidity model of the protein produced by FIRST.
- Extending the web application to 1) allow users to specify a scale for coloring the protein atoms, and 2) save different colorings and compare them.

The second problem addressed in this thesis is related to the development of motion simulation software. Different computational techniques exist for motion simulation, each with advantages and limitations, but currently none efficiently produce desired results. This creates a barrier for scientists trying to understand the behavior of macromolecules. We approached the problem by applying standard software engineering principles to design an infrastructure that would allow for the application and comparison of various motion

simulation techniques. Our design consisted of data representation and motion simulation parts. While it has some limitations, its main advantages are its flexibility and easy extensibility. The future implementation of this library would provide an effective infrastructure for comparing the performance of various approaches. This would foster the development of new motion simulation techniques and could improve the efficiency of already existing ones.

Future work on this project would include:

- Determining the remaining details on the types of certain functions and classes.
- Implementing the infrastructure and testing the optimality of the hierarchical structure of the design.
- Extending the design to support algorithms that perform a number of motion simulation steps simultaneously.

Bibliography

- [1] Classical Normal Mode Analysis: Harmonic Approximation.
<http://www.colby.edu/chemistry/pchem/notes/>.
- [2] FIRST: Floppy Inclusions and Rigid Substructure Topography.
<http://flexweb.asu.edu/software/first/>.
- [3] HotPatch: Statistical analysis of unusual patches on protein surfaces.
<http://hotpatch.mbi.ucla.edu/>.
- [4] Jmol: an open-source Java viewer for chemical structures in 3D.
<http://www.jmol.org/>.
- [5] PDB: The Protein Data Bank. <http://www.pdb.org/>.
- [6] K. Arnold, L. Bordoli, J. Kopp, and T. Schwede. The SWISS-MODEL Workspace: A web-based environment for protein structure homology modelling. *Bioinformatics*, 22:195–201, 2006.
- [7] Leonard Asimow and Ben Roth. The rigidity of graphs II. *Journal of Mathematical Analysis and Applications*, 68:171–190, March 1979.
- [8] Ivet Bahar and Burak Erman. Direct evaluation of thermal fluctuations in proteins using a single-parameter harmonic potential. *Folding Design*, 2:173–181, May 1997.
- [9] Robert S. Ball. *A treatise on the theory of screws*. Cambridge University Press, 1900.
- [10] Sam Flores. The database of macromolecular motions: new features added at the decade mark. *Nucleic Acids Res*, 34, 2006.
- [11] Jack Graver, Brigitte Servatius, and Herman Servatius. Combinatorial rigidity. *Graduate Studies in Mathematics*, 2, 1993.
- [12] Jenny Gu and Philip E. Bourne, editors. *Structural Bioinformatics*. John Wiley Sons, Inc, Hoboken, New Jersey, second edition, 2009.

- [13] Turkan Haliloglu, Ivet Bahar, and Burak Erman. Gaussian dynamics of folded proteins. *Physical Review Letters*, 79(16):3090–3093, October 1997.
- [14] Jeanne Hardy, Joni Lam, Jack Nguyen, Tom O’Brien, and James Wells. Discovery of an allosteric site in the caspases. *PNAS*, 101(34):12461–12466, August 2004.
- [15] Jeanne Hardy and James Wells. Searching for new allosteric sites in enzymes. *Structural Biology*, 14(6):706–715, December 2004.
- [16] Donald Jacobs and Bruce Hendrickson. An Algorithm for Two-Dimensional Rigidity Percolation: The Pebble Game. *Journal of Computational Physics*, 137(CP975809):346–365, 1997.
- [17] Donald Jacobs, A. Rader, M. Thorpe, and Leslie Kuhn. Protein flexibility predictions using graph theory. *Proteins*, 44:150–165, 2001.
- [18] Donald Jacobs and M. Thorpe. Generic Rigidity Percolation: The Pebble Game. *Phys. Rev. Letts.*, 75:4051–4054, 1995.
- [19] F. Kiefer, K. Arnold, M. Künzli, L. Bordoli, and T. Schwede. The SWISS-MODEL repository and associated resources. *Nucleic Acids Research*, 37:D387–D392, 2009.
- [20] Vijay Kumar. Rigid body motion and the Euclidean group. <http://roboticscourseware.org>, 2008.
- [21] Audrey Lee. *Geometric constraint systems with applications in CAD and biology*. PhD thesis, University of Massachusetts Amherst, May 2008.
- [22] Audrey Lee and Ileana Streinu. Pebble game algorithms and sparse graphs. *Discrete Mathematics*, 2007.
- [23] Ming Lei, Maria Zavodszky, Leslie Kuhn, and M. Thorpe. Sampling protein conformations and pathways. *Journal of Computational Chemistry*, 25(9):1133–1148, February 2004.
- [24] J. Monod, J. Wyman, and J. P. Changeux. On the nature of allosteric transitions: A plausible model. *Molecular Biology*, pages 88–118, May 1965.
- [25] David L. Nelson and Michael M. Cox. *Principles of Biochemistry*. W. H. Freeman and Company, fifth edition, 2008.
- [26] M. C. Peitsch. Protein modeling by e-mail. *Bio/Technology*, 13:658–660, 1995.

- [27] Harriet Pollatsek. *Lie Groups: A problem-oriented introduction via matrix groups*. The Mathematical Association of America, 2009.
- [28] Adam Schuyler and Gregory Chirikjian. Normal mode analysis of proteins: a comparison of rigid cluster modes with C-alpha coarse graining. *Journal of Molecular Graphics and Modelling*, 22:183–193, July 2004.
- [29] J. M. Selig. *Geometrical Methods in Robotics*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.
- [30] Adriana Stroba and Francis Schaeffer. 3,5-Diphenylpent-2-enoic acids as allosteric activators of the protein kinase PDK1: Structure-activity relationships and thermodynamic characterization of binding as paradigms for PIF-binding pocket-targeting compounds. *Journal of Medical Chemistry*, 52:46834693, 2009.
- [31] Godehard Sutmann. Classical Molecular Dynamics. *NIC Series, Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms*(10):211–254, 2002.
- [32] Tiong-Seng Tay. Rigidity of multi-graphs. I. Linking rigid bodies in n-space. *Combinatorial Theory Series*, B(26):95–112, 1984.
- [33] Tiong-Seng Tay. Linking (n-2)-dimensional panels in n-space II: (n-2, 2)-frameworks and body and hinge structures. *Graphs and Combinatorics*, 5:245–273, 1989.
- [34] Darin Vanderpool and Ted Johnson. Characterization of the CHK1 allosteric inhibitor binding site. *Biochemistry*, 48:98239830, 2009.
- [35] S. Wells, S. Menor, B. M. Hespenheide, and M. Thorpe. Constrained geometric simulation of the diffusive motions in proteins. *Physical Biology*, 2, 2005.
- [36] Neil White. Grassmann-Cayley algebra and robotics, 1994.
- [37] Neil White. Geometric applications of the Grassmann-Cayley algebra. *Handbook of Discrete and Computational Geometry*, 1997.
- [38] Neil White and Walter Whiteley. The algebraic geometry of motions of bar- and-body frameworks. *SIAM Journal of Algebraic Discrete Methods*, 8:1–32, 1987.
- [39] Walter Whiteley. The union of matroids and the rigidity of frameworks. *SIAM Journal Discrete Mathematics*, 1(2):237–255, May 1988.

- [40] Hugh D. Young and Roger A. Freedman. *Sears and Zemansky's university physics: with modern physics*, volume 1. Pearson Education Inc., 11th edition, 2004.