# L'extension de la géométrie des distances

Simon B Hengeveld

COLLEGE DOCTORAL BRETAGNE
MATHS, TELECOMS
INFORMATIQUE, SIGNAL
SYSTEMES, ELECTRONIQUE

Université de Rennes

# THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES

ÉCOLE DOCTORALE N° 601
*Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Électronique*
Spécialité : *Informatique*

Par
**Simon HENGEVELD**

## Extending Distance Geometry methods : not only distances

A study of three applications, with a focus on structural biology

**Thèse présentée et soutenue à Rennes, le 8 février 2024**
**Unité de recherche : IRISA**

### Rapporteurs avant soutenance :

Adrien GOËFFON Professeur, LERIA/Université d'Angers
Juan CORTÉS Directeur de recherches, LAAS-CNRS à Toulouse

### Composition du Jury :

| | | |
|---|---|---|
| Président : | Thérèse MALLIAVIN | Directeur de Recherches (HDR), LORIA-CNRS à Nancy |
| Examinateurs : | Adrien GOËFFON | Professeur des universités (HDR), LERIA/Université d'Angers |
| | Juan CORTÉS | Directeur de Recherches (HDR), LAAS-CNRS à Toulouse |
| | Douglas GONÇALVES | Professeur, Universidade Federal de Santa Catarina |
| | Silvia BOTTINI | Chaire de professeur junior (CPJ), INRAE à Nice |
| | Thérèse MALLIAVIN | Directeur de Recherches (HDR), LORIA-CNRS à Nancy |
| Dir. de thèse : | Antonio MUCHERINO | Maître de conférence (HDR), IRISA/Université de Rennes |

# Acknowledgement

First of all, I want to give my thanks to my supervisor, Antonio Mucherino, for his guidance and support throughout the PhD thesis. I would also like to thank the reviewers (Adrien Goëffon and Juan Cortés) and the committee members (Douglas Gonçalves, Silvia Bottini and Thérèse Malliavin) for their time, involvement and fruitful feedback. Furthermore, I want to acknowledge the different collaborators that I have had the pleasure of working with on the various projects carried out as part of my PhD: Thérèse Malliavin, Jung-Hsin Lin, Leo Liberti, Wagner da Rocha, Frank Plastria, David Pelta, Nara Rubiano da Silva, Douglas Gonçalves and Paulo H. Souto Ribeiro.

I am extremely grateful to my girlfriend, Sofia, who always supported me through both good and difficult moments these last three years. Her encouragement and love was immensely important and helped me finish this PhD. Next, I want to express my deepest appreciation to my parents, who helped me through the tough times and gave me very good advice. Apart from offering me work-related guidance, they were always available when I called on them and were of great emotional comfort. I would also like to mention my grandmother and my siblings, Paul, Hans and Naomi-Joan who also provided support, each in their own way.

My friends and (some ex) housemates, Chris, Imogen and Sjoerd are also owed my gratitude. The many discussions with Chris helped me solve various problems that I encountered throughout the PhD. The evening gaming sessions with Sjoerd (and sometimes my brother Paul) helped me get rid of any frustrations that I might have built up during the day.

# Resumé en Français

## Introduction

Distance Geometry (DG) est l'étude des concepts géométriques basés sur la distance. Un des premiers résultats historiques dans le contexte de DG est la formule de Heron, dont l'origine remonte au 1er siècle de notre ère. Cette formule peut être utilisée pour calculer l'aire d'un triangle uniquement en fonction de ses côtés (les distances entre ses sommets). De nombreux outils et techniques pratiques permettent de mesurer les distances entre objets, comme les capteurs des réseaux de capteurs [1] ou les expériences Nuclear Magnetic Resonance pour les distances inter-atomiques dans les protéines [2]. DG est la branche des mathématiques qui s'occupe de la façon dont nous pouvons obtenir plus d'informations sur l'ensemble des objets en utilisant uniquement des distances par paires. Dans cette thèse, nous étendons la discussion afin de considérer non seulement les distances euclidiennes, mais également d'autres concepts géométriques tels que les angles et les orientations locales ou différents types de distances. Le problème central dans le domaine de DG est le Distance Geometry Problem (DGP) [3], dans lequel nous sommes chargés de trouver les positions (relatives) des objets, de telle sorte que les distances mesurées entre eux soient satisfaites. Dans cette thèse, nous étudions différents aspects du problème de la géométrie des distances. Le DGP est un problème avec de nombreuses applications. Notre objectif principal réside dans l'application de la biologie structurale, où l'objectif est de déterminer la structure d'une protéine sur la base des distances interatomiques. Outre la détermination de la structure des protéines, nous discuterons également du problème du reciblage de mouvement, qui est lié à la dynamique DG [4, 5]. La dernière application que nous étudions est celle des cartes adaptatives. Notre travail pour cette application comprend le développement et la mise en œuvre d'un modèle linéaire qui montre des résultats prometteurs.

Il existe diverses méthodes et algorithmes qui peuvent trouver des solutions au DGP. Elles vont des méthodes basées sur la décomposition matricielle, l'optimisation globale, les heuristiques et les méta-heuristiques. Dans cette thèse, nous nous concentrons sur un algorithme qui exploite la discrétisabilité de certaines instances. Cette classe d'instance

est connue sous le nom de Discretizable Distance Geometry Problem (DDGP) [6]. Lorsque nous discrétisons l'espace de recherche, nous permettons d'énumérer toutes les solutions possibles au problème posé. L'algorithme capable d'effectuer cette tâche est branch-and-prune (BP), qui fonctionne en phase de branchement et d'élagage. Notez que cet algorithme ne peut énumérer complètement toutes les solutions possibles que lorsque nous utilisons principalement des données de distance précises. Dans de nombreuses applications, nous sommes confrontés à un degré élevé de bruit, ce qui conduit à des intervalles de distance. Il existe diverses implémentations de l'algorithme BP conçues pour gérer cette incertitude. Dans cette thèse, nous nous concentrons sur un en particulier: MDjeep [7]. Cette implémentation, initialement écrite en C, a été adaptée à Java pour cette thèse. De plus, dans le cadre de cette thèse, cet algorithme a été étendu pour utiliser non seulement les distances, mais également les angles de torsion. En particulier, elle a été étendue pour exploiter le signe des angles dièdres dans la phase de branchement. Cela signifie que la taille de l'espace de recherche est considérablement réduite, ce qui augmente les chances de l'algorithme d'identifier des solutions de bonne qualité.

## Biologie structurale

L'algorithme BP est particulièrement utile dans le contexte de la biologie structurale. Certaines instances qui surviennent dans le contexte d'expériences Nuclear Magnetic Resonance (NMR) peuvent être discrétisées [8]. De plus, dans ces cas, l'inclusion des informations sur l'angle de torsion devrait améliorer l'efficacité de la méthode BP. Pour étudier cela, nous avons présenté plusieurs expériences avec l'algorithme étendu BP, implémenté en Java. La première série d'expériences concerne des instances issues de données réelles NMR. Dans la littérature, il est courant d'étudier BP pour la biologie structurale en utilisant des données simulées au lieu de données réelles [7–14]. En effet, le bruit dans les données réelles est si élevé que les méthodes BP ont tendance à échouer. Les expériences présentées dans cette thèse montrent que, au moins pour les petites protéines, les structures générées à l'aide de la méthode BP étendue sont de bonne qualité. Cependant, à mesure que la taille des protéines augmente, l'incertitude capturée dans les instances augmente également. Cette incertitude a un impact important sur la qualité des résultats. Cela nous amène à la deuxième série d'expériences. Une source d'inexactitude dans les cas provient des variations dans la géométrie covalente des protéines. Nous présentons une analyse qui montre que les angles de liaison et l'angle de torsion $\omega$ peuvent présenter de

grands écarts d'un acide aminé à l'autre. Cependant, dans les cas que nous utilisons, ces variations ne sont pas capturées et ces angles (et les distances associées) sont souvent considérés comme constants. Dans les expériences que nous présentons, nous montrons l'effet de ces variations sur la méthode BP et les solutions générées. Dans les résultats, nous constatons une corrélation claire entre la taille des protéines et la qualité des solutions, montrant que pour des protéines plus grosses, l'algorithme produit des structures moins bonnes. Les travaux futurs sur ce sujet visent principalement à résoudre le problème de ces variations, ainsi qu'à améliorer la gestion de l'incertitude dans les instances.

## Géométrie de distance dynamique

Nous examinons ensuite la géométrie dynamique des distances en nous concentrant principalement sur les mouvements humains. Les mouvements humains sont des animations généralement enregistrées par capture de mouvement. Ces mouvements décrivent un personnage squelettique effectuant un mouvement à travers différentes images. La raison pour laquelle nous nous intéressons particulièrement aux mouvements humains est que, comme chez les humains, les protéines présentent également une structure squelettique. Par conséquent, avec cette étude, nous posons les bases de futurs projets qui pourraient se concentrer sur la modélisation de la dynamique des protéines. Dans cette section de la thèse, nous nous concentrons sur le problème du retargeting de mouvement [15–17]. Ici, l'enjeu est de copier le mouvement d'un acteur vers un deuxième personnage, de morphologie différente. Ce problème peut être décrit en utilisant Distance Geometry dynamique et des travaux antérieurs montrent que cette approche basée sur la distance peut être fructueuse [4, 5]. Nous étendons davantage l'approche basée sur la distance, en travaillant directement dans l'espace des angles d'Euler. De plus, nous proposons une nouvelle représentation des mouvements humains, inspirée de la biologie structurale, qui pourrait être particulièrement utile dans le contexte du reciblage des mouvements humains [18, 19]. En utilisant cette nouvelle représentation, nous présentons une analyse statistique, mettant en valeur la pertinence de la représentation. Les prochaines étapes sur ce sujet consistent notamment à améliorer encore la stratégie basée sur la distance pour le reciblage de mouvement. Deuxièmement, les travaux devraient être étendus à la modélisation de la dynamique des protéines.

## Cartes adaptatives

La dernière application sur laquelle nous nous concentrons est celle des cartes adaptatives [20, 21]. Ici, nous générons des cartes où les distances entre les Point of Interest (POIs) clés sur la carte ne sont pas euclidiennes, mais plutôt basées sur des mesures subjectives de proximité, qui dépendent de l'utilisateur. L'un des défis lors de la génération de cartes adaptatives est de déplacer les POI de manière à ce que ces distances subjectives soient satisfaites, tout en s'assurant que la nouvelle carte ressemble toujours à l'ancienne carte à usage général. Ce problème de relocalisation des POI peut être formulé comme un type de DGP, où nous incluons des contraintes basées sur les orientations locales entre les POI. Ces mêmes contraintes permettent de linéariser le problème. Pour cette thèse, un nouveau modèle linéaire a été proposé et implémenté. De plus, nous présentons diverses expériences avec le nouveau modèle, montrant des résultats prometteurs. Enfin, nous examinons une possibilité d'amélioration de la méthode, qui consiste à essayer différentes rotations des axes des orientations locales. Les travaux futurs se concentreront sur l'amélioration de cette méthode de rotation.

## Représentations binaires et DGP en dimension 1

Enfin, l'auteur a apporté des contributions mineures à deux autres sujets. Le premier travail se concentre sur les représentations binaires de problèmes combinatoires et sur la manière dont nous pouvons utiliser les méta-heuristiques pour trouver des solutions. Cela concerne DG car les solutions aux instances de la sous-classe DDGP peuvent être représentées à l'aide d'une chaîne binaire. Le deuxième travail considère le DGP en dimension 1. En particulier, nous nous sommes concentrés sur un type spécifique d'instances «paradoxales», qui semblent triviales mais sont en fait difficiles à résoudre par les méthodes branch-and-prune. Nous proposons des méthodes alternatives, basées sur une reformulation matrice par vecteur, qui permettent de résoudre plus efficacement ces instances paradoxales.

# Table of Contents

# List of Figures

# List of Tables

# List of Acronyms and Abbreviations

| | |
|---|---|
| APA | Alternating Projections Algorithm. |
| BH | Basin Hopping. |
| BP | branch-and-prune. |
| CNNs | convolutional neural networks. |
| DDGP | Discretizable Distance Geometry Problem. |
| DG | Distance Geometry. |
| DGP | Distance Geometry Problem. |
| DVOP | Discretization Vertex Order Problem. |
| dynDGP | Dynamical Distance Geometry Problem. |
| EDM | Euclidean Distance Matrix. |
| GO | Global Optimization. |
| LP | Linear Programming.<br>linear program. |
| MBH | Monotonic Basin Hopping. |
| MDGP | Molecular Distance Geometry Problem. |
| MDS | Multi-dimensional scaling. |
| MIP | Mixed-integer programming. |
| MSAs | Multiple Sequence Alignments. |

| | |
|---|---|
| NeRF | Natural Extension Reference Frame. |
| NLP | Nonlinear Programming. |
| NMR | Nuclear Magnetic Resonance. |
| NOE | Nuclear Overhauser effect. |
| PBH | Population Basin Hopping. |
| PCA | Principal Component Analysis. |
| PDB | Protein Data Bank. |
| POI | Point of Interest. |
| POIs | Points of Interest. |
| Revorder | Referenced Vertex Ordering. |
| RMSD | Root mean square deviation. |
| SA | Simulated Annealing. |
| SBB | Spatial Branch-and-Bound. |
| SDP | Semidefinite Programming. |
| SoME | Summer of Math Exposition. |
| SPE | Stochastic Proximity Embedding. |
| SPG | Spectral Project Gradient. |
| SPM | Successive Projection Methodology. |
| VNS | Variable Neighbourhood Search. |

Chapter 1

# Introduction

Distance Geometry (DG) is the study of geometric concepts based on distance. An early historical result in the context of DG is Heron's formula, which was proven in the 1st century CE. This formula can be used to compute the area of a triangle solely based on its sides (the distances between its vertices). Many practical tools and techniques allow for the measurement of distances between objects, such as the sensors in sensor networks [1] or Nuclear Magnetic Resonance experiments for inter-atomic distances in proteins [2]. DG is the branch of mathematics occupied with how we can obtain more information about the set of objects by using only pairwise distances. In this thesis, we extend the discussion so that we do not only consider Euclidean distances, but also other geometric concepts such as angles and local orientations or different distance types. The core problem within the field of DG is the Distance Geometry Problem (DGP), in which we are tasked with finding the (relative) positions of objects, such that the measured distances between them are satisfied. This thesis is focused on different variants of the DGP, its discretization, and some of its major practical applications, with an emphasis on structural biology.

In this introductory chapter, we start by giving formal definitions for the DGP (Section 1.1). Next, we will touch on the applications of the DGP (Section 1.2). Afterwards, in Section 1.3 we show that sometimes it is desirable to include other geometric information in the instances besides distances. We will discuss different methods of solving the general DGP found in the literature in Section 1.4. Finally, in Section 1.5, we will describe the Java package that was created and used for the various experiments conducted as part of this thesis. The second chapter of the thesis will focus on one of the most interesting variants of the DGP: the Discretizable Distance Geometry Problem (DDGP), whose instances arise in the context of molecular biology. The rest of the thesis is structured according to three applications of the DGP, to each of which a chapter is dedicated (Chapters 3 to 5). Chapter 3 focuses on structural biology, Chapter 4 discusses motion retargeting and human motions and Chapter 5 explores the application of adaptive maps. Finally, Chapter 6 closes off with a short summary and a brief discussion of some ongoing work to which the author has made minor contributions as part of the thesis.

## 1.1 Defining the Distance Geometry Problem

The Distance Geometry Problem (DGP) [3] is a problem that involves determining the positions of a set $V$ of $n$ points in a $K$ dimensional space, based on a (possibly incomplete) set $\mathcal{D}$ of pairwise proximity measures between these points. In general, $\mathcal{D}$ is a set of distances $\delta(u, v)$ for different pairs of points $(u, v) \in V$. Historically, this problem has had different names [22], such as the *position-location problem* [23] or the *K-embeddability problem* [24]. In the context of sensor network localization, the problem is often referred to as the *Graph Realization problem* [1]. The DGP was proven to be NP-complete for $K = 1$, while for $K > 1$, it was shown to be NP-hard [24]. The most studied variant of the DGP is the *assigned* version, where the assignment of every distance $\delta \in \mathcal{D}$ to a corresponding pair of points $\{u, v\} \in V \times V$ is known. There also exists the less researched *unassigned* DGP [25]. In this thesis, we will focus on the assigned variant.

An instance of the assigned DGP may be represented in different ways. One of the most widely used representations in the literature [3, 18, 19, 22, 26–30], and the representation that we will use throughout this thesis, utilizes a graph structure to describe the problem. For this representation, we use a simple, undirected graph $G = (V, E, d)$. The vertices of $G$ correspond to the set of points that we want to find a realization for. The existence of an edge $\{u, v\} \in E$ in $G$ indicates that we have access to the distance from $u$ to $v$. Finally, $d$ is the weight function in $G$ which pairs every edge $\{u, v\} \in E$ to its corresponding distance $\delta \in \mathcal{D}$:

$$d : \{u, v\} \in E \longrightarrow \delta(u, v) \in \mathcal{D}. \tag{1.1}$$

Using the graph representation, the DGP can be formalized as follows [3, 22]:

**Definition 1** *Given a simple weighted undirected graph $G = (V, E, d)$ and an integer $K \in \mathbb{Z}_+$ the (assigned) DGP asks whether a realization*

$$x : v \in V \longrightarrow x_v \in \mathbb{R}^K \tag{1.2}$$

*exists, such that*

$$\forall \{u, v\} \in E, \quad ||x_u - x_v|| \in \delta(u, v) \tag{1.3}$$

There are two important details to note here. Firstly, it is natural to think of the distances $\delta$ and the norm $||\cdot||$ in the constraints (1.3) to be Euclidean. However, there are

different methods in which the proximity between objects is estimated via non-Euclidean measures. Despite this, we will still attempt to find a realization in the Euclidean space. For example, in Section 1.4.6 and Chapter 5, we will see that in order to linearize the DGP, we need to use a linear distance measure, such as the $L_1$ and $L_\infty$ norms [31, 32]. Secondly, various applications will have distances that may include noise (for example, distances obtained through physical experiments). For this reason, the constraints (1.3) leave space for the distances $\delta(u,v)$ to be non-exact (an interval) $[\underline{\delta}(u,v), \overline{\delta}(u,v)]$:

$$\underline{\delta}(u,v) \leq ||x_u - x_v|| \leq \overline{\delta}(u,v),$$

as well as the exact case (degenerate interval):

$$||x_u - x_v|| = \delta(u,v).$$

In some cases, where we have a lot of noise on the input distances, it may not be possible to find a realization such that all distances are satisfied. In these cases, it can instead be useful to consider the DGP as a Global Optimization (GO) problem, where we aim to minimize the error on the distances [26]:

**Definition 2** *Given a simple weighted undirected graph $G = (V, E, d)$ and an integer $K \in \mathbb{Z}_+$ the unconstrained optimization-based DGP asks to determine a realization*

$$x : v \in V \longrightarrow x_v \in \mathbb{R}^K$$

*such that a penalty function $\sigma$ is minimized.*

The penalty function $\sigma$ should measure the violation of the distance constraints. For example, we may use [4, 33]:

$$\sigma(x) = \frac{1}{2} \sum_{\{u,v\} \in E} (\|x_u - x_v\| - \delta(u,v))^2 \tag{1.4}$$

Note that when all distances are exact, the problems given in Definitions 1 and 2 are equivalent. An important property of $\sigma(x)$ is that it is differentiable at $x$ when no two points are in the same position. Thus we can compute the gradient $\nabla\sigma(x)$ and use methods such as gradient descent to find local minima. In Section 1.4, we will see several optimization techniques that find solutions for the DGP by minimizing such penalty functions.

When the distance graph $G$ is complete, or at least dense, it may make sense to represent the DGP using an Euclidean Distance Matrix (EDM) [34–36], which is a hollow square matrix containing the pairwise squared distances between the points in $V$. Consider some order on the points in $V = \{v_1, \ldots, v_n\}$, where $n = |V|$. The entry of the EDM at indices $(i, j)$ contains the squared distance $\delta(v_i, v_j)^2$. This is a matrix of real values, which means an EDM cannot be used to represent interval distances, but only exact ones. This may be too limiting for some of the applications of the DGP. Figure 1.1 showcases the graph representation of the DGP and compares it to the EDM representation.

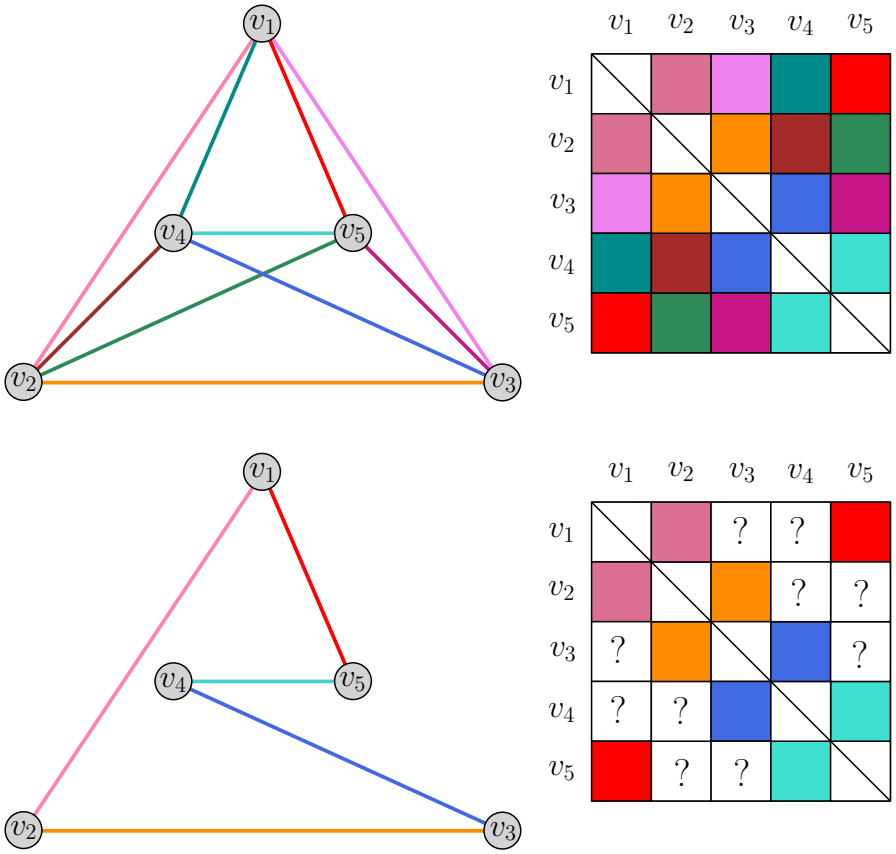



Figure 1.1 – On the left we see the graph representation of the DGP, where the presence of an edge represents the fact that we know a distance. The colours of the edges represent the value of the distance. On the right, we see the EDM representation of the same two instances. On the top, we have a fully complete instance where we have access to all pairwise distances. On the bottom, we see a cycle graph, where we miss half of the distances, leading to a sparser matrix. Inspiration was taken from Fig. 1 in [36].

Note that in the graph representation of the DGP, we do not have a specific vertex order on $V$, while the EDM does include a implicit vertex ordering. The advantage of the EDM representation is that we can use several techniques to find an embedding for $V$ that

operates directly on the EDM (see Section 1.4.1). However, there are several complications with these techniques. In many real-world applications, we will have missing distances, and thus the EDM will be incomplete. One can only use the matrix-based techniques to find a realization of the distances when the EDM is complete. This leads to a problem known as *EDM completion*, where efforts are made to complete the missing distance information in the matrix, for example through the use of Semidefinite Programming. More information on these techniques is given in Section 1.4.1.

## 1.2 Different DGPs and their applications

Distance Geometry Problems can come in many forms. Instances of these different types of problems can be found in different practical applications. Common themes are the notions of discretization, graph rigidity, non-Euclidean distance norms, and the fact that in practice we often have a large degree of noise on the input distances. As part of this thesis, practical work was done on four of these applications (molecular structure determination, motion retargeting, adaptive maps, and clock synchronization).

A very important application, and the main focus of this thesis, is *molecular structure determination* [27, 33, 37–45]. In this application, the vertices $V$ of the graph $G$ correspond to atoms of a molecule, and the distance information $d$ associated with the edges $E$ of the graph are the inter-atomic distances. The goal is to determine the three-dimensional structure of the molecule such that the inter-atomic distances are satisfied. The instances of the DGP in the context of molecular structures often belong to a special subclass of the problem. The distance graphs that we obtain satisfy special conditions that let us discretize the search space [6, 8, 26]. This means that we may be able to enumerate all three-dimensional structures that a protein can fold into. These types of problems are referred to as instances of the Discretizable Distance Geometry Problem (DDGP). More detail on this instance type can be found in Chapter 2. The problem of molecular structure determination is further examined , where several experiments will be presented. In the literature, most experiments with DDGP methods for protein structure determination are done with artificially generated instances from known structures. In Chapter 3, experiments will be presented with instances created from real protein data. An example of some of the results of these experiments can be seen in Figure 1.2, which shows the structures of two proteins with complex secondary structures, that were computed by exploiting several available distances between the atoms of the molecules.

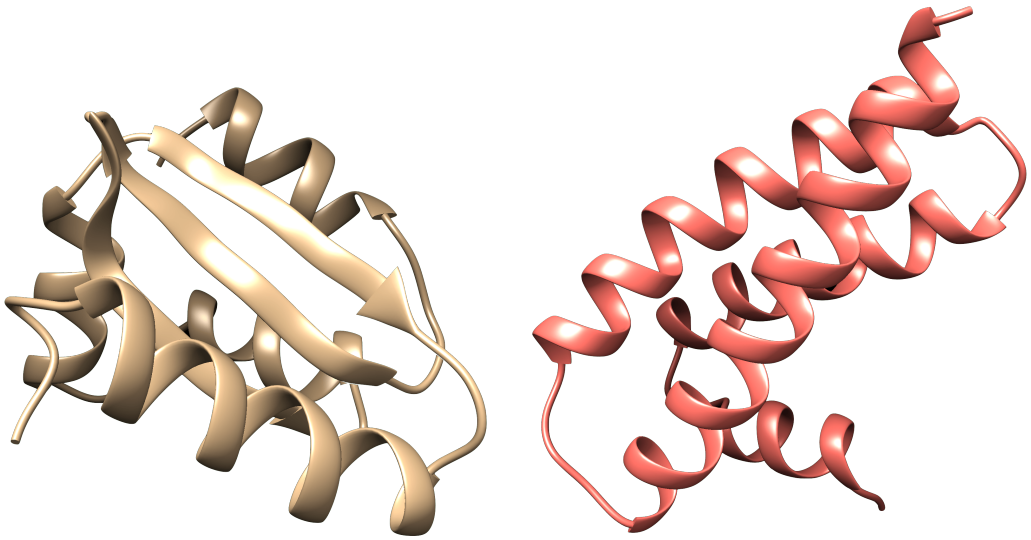


Figure 1.2 – Examples of conformations of two proteins (PDB codes from left to right: 1YD0A, 4WZXA). These structures were computed using the inter-atomic distances and other geometric information, using software created as part of this thesis. The image was generated using molecule viewing software Chimera [46].

It is known that, because of the inter-atomic forces at play, the atoms of a protein move over time, forcing the protein to take on different conformations. Instances of the DGP where we have such a dynamic aspect, where the points that we want to embed may move over time, are referred to as instances of the Dynamical Distance Geometry Problem (dynDGP). An example of an application of this dynamical variant is *motion retargeting*, where we want to copy the animation of one character to another while avoiding unwanted contact between the joints of the character. To do this, we may exploit the distances between the joints in the original character, and attempt to map these to the second animation. To solve such a dynDGP, essentially we have to solve a series of DGP instances. In the case of motion retargeting, we are solving one DGP instance per frame. Aside from motion retargeting, the dynDGP has other applications such as simulating pedestrians in a crowd [47, 48] or aircrafts in the air [49]. More information about the dynDGP will be given in Chapter 4, in which several experimental results will be presented on the problem of motion retargeting.

The concept of discretizability, which was mentioned in relation to the application of molecular structure determination, is closely linked to the notion of *graph rigidity* [23, 50–54]. In fact, if a graph is rigid, we know the number of possible incongruent realizations of the graph is finite [22, 55]. A graph is said to be rigid if its edges will not *bend* or

*flex* under an applied outside force. This leads us to a historical application of Distance Geometry (DG): statics [56, 57], which is the study of forces that act on non-moving bodies in physical systems that are in static equilibrium. More details of how this application relates to DG can be found in [22].

Another interesting application of the DGP is *wireless sensor network localization* [1, 58–63], which is a problem where rigid graph theory is very relevant [64]. Here, we have a set of wireless sensors in a network. To properly route communication signals through the network, the router must determine the positions of the sensors. Every sensor, given a certain radius, is aware of its neighbours. This awareness is expressed in the form of a distance, which means that we have an instance of the DGP. In our graph representation for this problem, the vertices $V$ are the sensors and edges $E$ reflect the distances between them. These sensors are prone to making errors, so similarly to when we deal with interatomic distances for molecular structure determination, the measured distances between the sensors may contain a lot of uncertainty. Rigid graphs have the property of sustaining various kinds of deformations due to translation, rotation, and reflection. Hence, it is fruitful to use the concepts of rigid graphs, to find the locations of the sensors, even when we have noise on the distances between them.

A classical application of the Distance Geometry Problem is *dimensionality reduction* [65–70]. Dimensionality reduction is the transformation of data points in a high-dimensional space to a lower-dimensional space. In the context of Distance Geometry, the main idea is to start by computing the distances between the data points in the input. This is an example of a case in which we use distance norms other than the Euclidean one. This is because distances computed at higher dimensions are likely to be non-realizable in lower dimensional spaces. Therefore, other distance norms may be used, such as the $L_1$ norm (see Section 1.4.6 for a definition of this norm). Once we have the distances between the data points, we can solve an instance of the DGP with a different target dimension $K$, lower than the original dimension of the data points. The DG-based techniques for solving dimensionality reduction problems are known as Multi-dimensional scaling (MDS) [34, 36, 65], and work directly on EDMs (see Section 1.4.1).

Another application of Distance Geometry in which non-Euclidean distance norms are important is the more recently studied problem of *Adaptive maps*. More often than not, geographical maps are meant for general use. Because of this, the distances between Points of Interest (POIs) on modern maps are based on the Euclidean distance between them. Adaptive maps on the other hand are maps that are modified based on characteristics

specific to the end-user, such as the slope of the route or the accessibility of the user. To create such an adaptive map, the original geographical map needs to be modified to match these new user-specific distances. This corresponds to solving a DGP in dimension 2. To make sure the adaptive map still bears a resemblance to the original, some extra linear constraints are added to the problem. This makes Linear Programming one of the best tools for finding solutions to the problem. More information on Linear Programming as a tool to solve DGPs can be found in Section 1.4.6 while more details on adaptive maps and several related computational experiments can be found in Chapter 4.

In dimension 1, the $L_1, L_2$, and $L_\infty$ norms are the same. In this dimension, we have the *clock synchronization* problem, where we are given a set of clocks, a subset of offset measurements between pairs of clocks as well as some offsets to a central clock, for which we know the time. The goal is to compute the precise time on each of the clocks [61, 71, 72]. This is the main practical application for the DGP in dimension $K = 1$. The clocks represent the vertices $V$ of our graph $G$ and the edges $E$ and their weight $d$ corresponds to the temporal offsets between the clocks. Section 6.2.2 focuses on this problem and the DGP in dimension 1. Specifically, we focus on one special difficult subclass of the DGP in dimension 1, in which our distance graph $G$ is a cycle graph.

One of the reasons that the DGP is an interesting and multifaceted problem is that applications seemingly appear everywhere. For example, a niche application, closely related to sensor networks, is that of controlling and locating autonomous underwater vehicles [73, 74]. An application of the DGP can even be found in music [75], where Distance Geometry is used to provide a quantitative analysis of rhythm and music. Aside from these, DG is also relevant for visualization [76, 77] and graph coloring [78].

## 1.3 Not only distances

In some of the applications mentioned in the previous section, we may have more geometric data available that is not captured by the input distances. Therefore, it is desirable to include this extra information in the instances so that we may exploit it.

### Vector and torsion angles

In the context of molecular structure determination, we have access to two types of angles: *vector* and *torsion* angles. Vector angles $\zeta(v_1, v_2, v_3) \in [0, \pi/2]$ are induced by three vertices $v_1, v_2$ and $v_3$, where we know $\delta(v_1, v_2)$ and $\delta(v_2, v_3)$. These vector angles can

be easily converted to a third distance $\delta(v_1, v_3)$ by using the cosine rule, without losing any information. In the case of proteins, these angles come in the form of *bond angles*, which are angles created by every pair of atomic bonds that include a common atom.

In contrast to these types of angles, torsion angles cannot be so easily translated to distance information. Given four vertices $v_1, v_2, v_3$ and $v_4$. The torsion angle $\tau \in [-\pi, \pi]$ is the *dihedral* angle created by the two planes induced by $v_1, v_2, v_3$ and $v_2, v_3, v_4$. From this point onward, we will confuse the two terms dihedral and torsion angle.

Examples of one torsion angle and two vector angles induced by four vertices are shown in Figure 1.3. In the case we know the three distances $\delta(v_1, v_2)$, $\delta(v_2, v_3)$ and $\delta(v_3, v_4)$, as well as the vector angles $\zeta(v_1, v_2, v_3)$ and $\zeta(v_2, v_3, v_4)$ we may convert the torsion angle to a distance $\delta(v_1, v_4)$.

One way to do this is by computing the coordinates of the points $v_1$ and $v_4$ with the origin set at the midpoint of the vector $\overrightarrow{v_2v_3}$. We imagine the particular case where the torsion angle $\tau = 0$. Then, from this specific case, we generalize for all other cases by rotating the vector $\overrightarrow{v_1v_2}$ around the $x$ axis, with an angle of $\tau/2$. We rotate $\overrightarrow{v_3v_4}$ in the opposite direction, with angle $-\tau/2$. This gives us the below formulas for $v1$ and $v4$:

$$v_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\tau/2 & -\sin\tau/2 \\ 0 & \sin\tau/2 & \cos\tau/2 \end{pmatrix} \begin{pmatrix} -\delta(v_1, v_2)\cos\zeta_1 \\ +\delta(v_1, v_2)\sin\zeta_1 \\ 0 \end{pmatrix} - \begin{pmatrix} \delta(v_2, v_3)/2 \\ 0 \\ 0 \end{pmatrix},$$

$$v_4 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\tau/2 & \sin\tau/2 \\ 0 & -\sin\tau/2 & \cos\tau/2 \end{pmatrix} \begin{pmatrix} -\delta(v_3, v_4)\cos\zeta_2 \\ +\delta(v_3, v_4)\sin\zeta_2 \\ 0 \end{pmatrix} + \begin{pmatrix} \delta(v_2, v_3)/2 \\ 0 \\ 0 \end{pmatrix}.$$

All that remains is to compute the distance from $v_1$ to $v_4$ using the Euclidean distance formula.

However, converting such a torsion angle to a distance is not a lossless process. In fact, $+\tau$ and $-\tau$ lead to the same distance $\delta(v_1, v_4)$. It is therefore desirable to include the torsion angles directly in the instances and exploit them in other ways, such that the information of the sign is not lost.

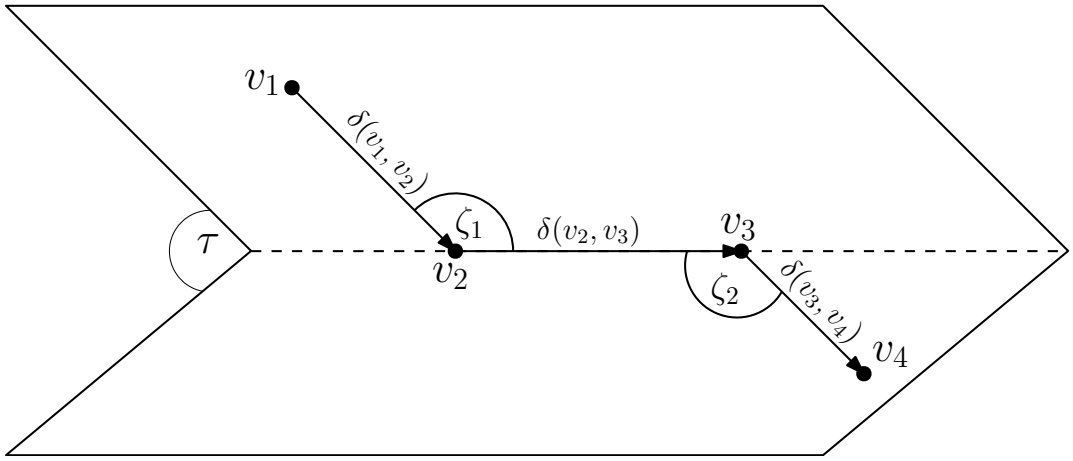



Figure 1.3 – An illustration of two vector angles $\zeta_1 = \zeta(v_1, v_2, v_3)$ and $\zeta_2 = \zeta(v_2, v_3, v_4)$ as well as the dihedral angle $\tau$

To achieve this, we represent the problem using a directed weighted *hypergraph* $G(V, E, d)$. In such a graph, an edge (called *hyperedge*) can join any number of vertices. Such a hyperedge $e \in E$ consists of a pair of subsets of $V$. In our hypergraph we will only have two types of edges, those that relate to distances and those that relate to torsion angles. We will refer to the edges that correspond to distances as $E_\delta \subset E$. An edge that relates to the distance between the vertices $u$ and $v$ is simply represented using a pair of singletons: $(\{u\}, \{v\}) \in E_\delta$. For every hyperedge $(\{u\}, \{v\}) \in E_\delta$, we should have the corresponding hyperedge $(\{v\}, \{u\}) \in E_\delta$. The edges that relate to torsion angles concern quadruplets of vertices which together (in a specific order) form a torsion angle $\tau$. These edges $E_\tau \subset E$ will consist of a pair of a triplet of vertices with a singleton. For example, if a hyperedge relates to a torsion angle formed by the quadruplet $v_1, v_2, v_3$ and $v_4$, we will have an edge $(\{v_1, v_2, v_3\}, \{v_4\}) \in E_\tau$. Note that these (directed) edges imply a specific vertex order. Vertex orders and their importance are discussed in detail in Chapter 2. In order to complete our description, the hypergraph is paired with the weighting function $d$. The function $d$ maps each hyperedge $(\{u\}, \{v\}) \in E_\delta$ to an interval distance $\delta(u, v)$, where $\delta(u, v) = \delta(v, u)$. Every edge $(\{v_1, v_2, v_3\}, \{v_4\}) \in E_\tau$ is mapped to an interval $[\underline{\tau}(v_1, v_2, v_3, v_4), \overline{\tau}(v_1, v_2, v_3, v_4)]$ described by $\tau(v_1, v_2, v_3, v_4)$. Note that just like before, the intervals $\delta(u, v)$ and $\tau(v_1, v_2, v_3, v_4)$ may be degenerate, i.e., they may correspond to an exact value.

This hypergraph lets us define a new variant of the DGP, in which torsion angles given in the input should also be observed:

**Definition 3** *Given a weighted directed hypergraph graph $G = (V, E, d)$, where $d$ maps edges $E$ to distances $\delta$ and torsion angles $\tau$, the (torsion angle enriched) DGP asks whether a realization*

$$x : v \in V \longrightarrow x_v \in \mathbb{R}^3 \tag{1.5}$$

*exists, such that*

$$\forall(\{u\}, \{v\}) \in E_\delta, \quad ||x_u - x_v|| \in \delta(u, v)$$

*and*

$$\forall(\{v_1, v_2, v_3\}, \{v_4\}) \in E_\tau, \quad dihed(x_{v_1}, x_{v_2}, x_{v_3}, x_{v_4}) \in \tau(v_1, v_2, v_3, v_4, )$$

*where $dihed(x_{v_1}, x_{v_2}, x_{v_3}, x_{v_4})$ corresponds to the dihedral angle that is formed by the positions of $v_1, v_2, v_3$ and $v_4$.*

In Section 2.5, we will see that including this extra information into our instances will let us decrease the size of the search space in the context of the branch-and-prune (BP) method. Furthermore, experiments in Section 3.5.2 will illustrate the usefulness of torsion angles in the context of molecular structure determination. Molecular structure determination is not the only application for which we are interested in these angles. In Chapter 4, we will see that they are also of interest for motion retargeting, where they allow us to define a new representation for human motions.

### Local relative orientations

Instances of the DGP relating to adaptive maps showcase another interesting case where we have additional geometric information. Because we have access to the original geometric map, we have access to the relative orientations between each pair of points that we want to embed. Specifically, we have access to the relative orientations for every pair of vertices $(u, v) \in E$. In the case of adaptive maps where the dimension $K = 2$, these orientations are included in the instances in terms of linear constraints by using four quadrants (NW, NE, SW, and SE). The edge set $E$ is partitioned into four subsets that correspond to these quadrants. For example, when we have $(u, v) \in$ NW we know that the positions for $v$ should be to the *North-West* of the positions for $u$. Including these constraints let us define a new variant of the DGP, where the local orientations

are imposed on the desired realization $x$. This variant of the problem can be solved by constrained optimization methods. The linear nature of these constraints makes Linear Programming an important contender for solving these instances. More information can be found in Section 1.4.6 and Chapter 5.

## 1.4 Methods for solving the DGP

Many algorithms have been proposed for solving the Distance Geometry Problem. There are methods using matrix operations directly on the EDM. We also have the option to use fast-running non-linear optimization methods that work mostly well on small instances. For applications where it makes sense to linearize the problem, we can opt for Linear Programming to find good-quality solutions effectively. Finally, if our instances meet specific conditions we can use geometric build-up methods that have the capability of providing us with all possible realizations to the problems at hand. In this section, we will describe the most important methods and frameworks. We also touch on the strengths and weaknesses of each of these methods.

### 1.4.1 Operating on Euclidean Distance Matrices

We are given a collection of $n$ points in dimension $K$, described by a coordinate matrix $X$ with $K$ rows and $n$ columns, where every column represents the set of coordinates of one of the points. We can now exploit the Euclidean distance formula to devise a function that computes the Euclidean Distance Matrix given such a (centered) coordinate matrix $X$ [36]:

$$\mathrm{edm}(X) = \mathbf{1}\mathrm{diag}(X^\top X)^\top - 2X^\top X + \mathrm{diag}(X^\top X)\mathbf{1}^\top \tag{1.6}$$

where $\mathbf{1}$ is a column vector of ones and $\mathrm{diag}(A)$ is a column vector with the diagonal entries of the matrix $A$. Solving the DGP, given an EDM, corresponds to inversing the equation shown in (1.6), as long as all the distances are exact and we are considering the Euclidean norm. This is precisely the approach that Multi-dimensional scaling (MDS) [34, 36, 65] methods take to find solutions for the DGP. Recall from Section 1.1 that when we have instances with interval distances, we cannot use an EDM to represent the problem. Therefore we cannot use classical MDS techniques for these instances. However, as we will see towards the end of this section, EDM completion methods can be used to deal with

instances with interval distances. Classical MDS was implemented as part of the Java package for this thesis and included as a benchmark for other methods (see Section 1.5).

Note that in (1.6), we see that the EDM can actually be defined as a function of the *Gram matrix* $G$:

$$G = X^\top X \tag{1.7}$$

Given a complete Euclidean Distance Matrix $D$, the first step of classical MDS is to compute $G$, starting from $D$. We can do this by exploiting the Geometric centering matrix $J$:

$$J = I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top \tag{1.8}$$

$$G = -\frac{1}{2}JDJ \tag{1.9}$$

where $I$ is the identity matrix. Schoenberg's theorem [79] shows that there is a one-to-one relation between a matrix being an Euclidean Distance Matrix and its Gram matrix $G$ being *positive semi-definite* (all of its eigenvalues are non-negative) [34]. This means that if $D$ is a valid EDM we can use the eigenvalue or singular value decomposition of $G$ to compute $X$, which is the last step of classical MDS. This method was already proposed in 1901 by the name Principal Component Analysis (PCA) [80], and has been widely used for finding realizations for EDMs [42, 81]. This method works reasonably well, even when the distances include noise. This is because it uses only the top-$K$ eigenvalues to compute the realization matrix $X$, where $K$ is the dimension we are trying to embed in. However, remember classical MDS only works if we have access to a complete EDM, with all pairwise squared distances. The application that is most relevant to these methods is that of of dimensionality reduction (see Section 1.2), because in this application we often know all pairwise distances between the input data points and can thus construct a complete EDM.

In the case where we have an incomplete EDM, it may still be possible to use EDM-based methods to solve these incomplete instances. The general approach is by first using an algorithm to complete the missing distance information in EDM. Then, as a final step, we can use classical MDS to compute the coordinate matrix $X$ corresponding to a realization of the EDM. One can directly exploit the *rank* of the EDM. Because the coordinate matrix $X$ has $K$ rows, the ranks of $X$ and $X^\top X$ are at most $K$. The other two terms in (1.6) have rank 1, which means that any valid EDM has a maximal rank of $K + 2$ [36]. Because we know the embedding dimension $K$ and thus the rank of the

matrix, one approach is to try to iteratively complete the EDM, while enforcing the (noisy) distances that we know, until the desired rank is reached. Alternatively, because we know that the EDM will have a low rank, we may use several general algorithms for low-rank matrix completion that have been proposed in the literature. One example of this is OptSpace [82], which is based on spectral methods followed by local manifold optimization. Apart from gradient descent methods, we can also use an alternating least squares minimization approach, such as AltMinComplete proposed in [83].

Aside from exploiting the rank of the EDM, we can try to complete it by propagating triangle and tetrangular inequalities from the known distances. The well-known EMBED algorithm works using this approach [37]. These inequalities give us lower and upper bounds on the missing distances. To be able to use classical MDS, we need an EDM with only real, exact values, and no interval distances. The EMBED algorithm overcomes this issue by iteratively selecting a "trial" matrix by randomly choosing values within the distance intervals. These random values can be chosen uniformly or one can use metrization, to compute a random metric space whose distances all lie within their respective limits [37]. The obtained matrix will most likely not be a valid EDM, which is why the second step includes running local optimization to attempt to convert the invalid EDM to a valid one. A similar approach is the Alternating Projections Algorithm (APA) [84]. This is based on Successive Projection Methodology (SPM) [85], which projects a starting alternatively on two convex sets. Instead of first exploiting triangle and tetrangular inequalities, here we simply generate a random starting point EDM $D$, such that all entries of $D$ either satisfy the known distance constraints or are equal to 0 otherwise. Then, we alternatively project $D$ on two convex sets: the set of matrices with a zero-diagonal and the set of matrices that correspond to a positive semi-definite Gram matrix $G$ (1.9). At the intersection of these two sets lie the completed, valid EDMs.

Semidefinite Programming (SDP) [1, 35, 38, 45, 58, 59] has been widely used for EDM completion as well. Semi-definite programming can be used to solve convex relaxations of a global optimization problem, with the requirement that the variables in the optimization problem correspond to entries in a positive semi-definite matrix. Because we know that the gram matrix $G$ is positive semi-definite, we can cast EDM completion as a semidefinite program, and then we solve this (convex) relaxation. In fact, the EDM completion problem corresponds to a special case of the general Semidefinite Programming feasibility problem [86]. These semidefinite programs may be solved using primal-dual interior point-solvers. However, this is an expensive process and may lead to inexact solutions.

Instead, it is possible to avoid SDP solvers altogether when a facial reduction technique is used [87, 88]. Interestingly, this method finds solutions by regarding the problem in three equivalent forms: as a graph realization problem, as the EDM completions problem as well as a rank restricted SDP. The face of the SDP cone corresponding to a given clique in the graph is characterized. Next, the intersection of two faces corresponding to overlapping cliques is characterized. Such an intersection may be obtained by computing the singular value decomposition. This intersection step is equivalent to completing blocks of the EDM. This process is repeated until the EDM is fully completed, which can be done in a finite number of intersection steps. Note that this method cannot be used for all instances of the problem. In some cases, the entire problem may not be reduced to a single clique. In this case, it is still necessary to use a SDP solver to complete the EDM.

In general, EDM-based methods can be effective and produce good-quality results. When the EDM is complete, classical MDS is a fast-running method that provides good-quality results even if we have some noise present in the input distances. However, having a complete EDM is a very strict requirement for most important applications. For example, in molecular structure determination, we are missing many distances making the EDM incomplete (see Figure 1.1, bottom-right). When we are missing only a few distances, the EDM completion methods can still lead to good-quality results. Another advantage of some of these EDM completion methods (EMBED and SDP approaches) is that they can handle interval distances in the input as well. However, these algorithms do have some pitfalls. A very important shortcoming is that when the EDM is sparse even the methods based on EDM completion tend to fall short [45]. This means that for many of the harder applications of Distance Geometry, these methods will not be very effective.

Furthermore, when we have instances with a large number of points, these large matrices use a lot of memory space. Furthermore, the MDS based methods do not scale so well: generally, $O(n^2)$ or worse. In the next two sections, we describe (meta) heuristics and non-linear optimization techniques which aim to address this problem of scalability, and will generally run much faster on larger applications.

### 1.4.2 Stochastic Proximity Embedding

One straightforward heuristic has been proposed for the DGP, known as Stochastic Proximity Embedding (SPE) [89]. For a description of the algorithm using pseudocode, see Algorithm 1. We start by initializing with a random initial realization $x$ and choosing a learning rate $\lambda$. From here, we try to improve $x$ in iterations, until a maximum number

of iterations is reached. To do this, we have a double loop: the inner loop runs for several *steps* $S$ while the outer loop runs for an amount of *cycles* $C$. In the inner loop, at every step, we select one random vertex pair $(u, v)$ for which we know the true distance $\delta(u, v)$. Then, we adjust the coordinates $x_u$ and $x_v$ such that the distance $||x_u - x_v||$ better approximates $\delta(u, v)$. The amount by which we "correct" the coordinates is calculated by using the learning rate $\lambda$. In the outer loop, after each set of steps, we decrease our $\lambda$ by a predetermined amount. This means that in the earlier cycles, we will update the coordinates more drastically. When we start to converge towards a solution, only small modifications are made.

**Algorithm 1** Stochastic Proximity Embedding (SPE)

**Input:** Cycles $C$, steps $S$, learning rate $\lambda$ and decrease factor $\delta_\lambda$, small tolerance value $\epsilon$
**Output:** Realization $x$

```
x ← initial random realization
while c ≤ C do
    while s ≤ S do
        Select a random distance δ(u, v)
        d_uv ← ||x_u − x_v||
        if d_uv ∈ δ(u, v) then
            r_uv ← δ(u, v) if it is exact, otherwise the nearest bound to d_uv
            x_u ← x_u + (λ/2) ((r_uv − d_uv)/(d_uv + ε)) (x_u − x_v)
            x_v ← x_v + (λ/2) ((r_uv − d_uv)/(d_uv + ε)) (x_v − x_u)
        end if
        s ← s + 1
    end while
    c ← c + 1
    λ ← λ − δ_λ
end while
return x
```

SPE was implemented as part of this thesis to serve as a benchmark for the other solvers (see Section 1.5). One of the main advantages of SPE is that it is very straightforward to implement. SPE is a great alternative to EDM based methods, as the computation time scales in order of $O(n)$ and because it can trivially handle interval distances. Furthermore, the inner loop of the SPE procedure lends itself to parallelization, which will make it run even faster. Another advantage of SPE over the MDS techniques is that it does not require the distance graph to be dense at all. However, these advantages come at a price. As it is such a simple heuristic, we cannot always expect high-quality solutions.

Because of its stochastic nature, we have no guarantees on the accuracy of the solution. In the next section, we will see some (global) optimization techniques that are a little more robust than SPE.

### 1.4.3 Nonlinear Programming

In a heuristic like SPE, at every iteration, we only modify up to two positions of the realization $x$. However, we can also update the positions $x_v$ for all $v \in V$, at the same time, by regarding the problem as a Global Optimization (GO) problem (see Definition 2). We may use different Nonlinear Programming (NLP) methods to attempt to solve this global optimization problem. These methods exploit the fact that penalty functions such as (1.4) are differentiable on realizations $x$ when there are no pairs of vertices that have the same position [26].

When we use the graph representation of the DGP (see Definition 1) we may use some mixed combinatorial methods based on graph decomposition. The idea is to divide our graph $G$ into a set of smaller graphs. Then, in the first phase, we use local optimization to solve these subgraphs. In the second phase, we attempt to combine the solutions to the subgraphs to find a solution for the global optimization problem. The first algorithm to use this graph decomposition approach was ABBIE [90], where local optimization techniques were used for both phases. Alternatively, Semidefinite Programming techniques may be used for both phases as well [91, 92]. This second, global optimization phase of this approach is the most challenging step and is what can complicate matters. Often, heuristic approaches must be taken to do this effectively. An example of this is the algorithm 3D-ASAP [93]. A drawback of these graph decomposition methods is that they only work on exact distances, and extending them to interval data is not trivial [22].

Another option is Spatial Branch-and-Bound (SBB), which is a divide-and-conquer technique used to find the solution to global optimization problems. SBB uses a deterministic search using three steps: branching, bounding and, pruning. We start with the branching phase, in which we divide the global problem into regions. Each of these regions lets us compute lower and upper bounds on the objective function by utilizing local optimization. This is called the bounding step. Finally, in the pruning step, we discard any regions that have lower bounds larger than the best local optimum found so far. We repeat these three steps, on increasingly smaller regions, until the computed bounds converge, which means we have found a local optimum. This general purpose approach was tried for the DGP in [94]. The advantage is that it is deterministic, while most of the other

approaches discussed in this section are stochastic. However, this method is extremely slow, even for small-sized instances.

There are also several algorithms based on smoothing, in which local optimization is used to find minimizers for increasingly finer smoothings of the original global optimization problem. At each iteration, we end up with fewer local minima, until we finally end up with a convex optimization problem that we can solve with local optimization. This is known as a *global continuation* method. A well-known example of this approach is the DGSOL algorithm [43, 44]. A similar algorithm (DCA) was introduced in [95, 96], which differs mostly in their choice of local optimizer. DGSOL is a fast-running algorithm that provides good-quality solutions for small instances. To extend to (slightly) larger instances, DGSOL was combined with the meta-heuristic VNS [97] (discussed as a stand-alone method in the next section). This lead to better quality solutions for larger instances, but increased the computation time of the approach.

An example of a global constrained optimization method that was introduced specifically to solve instances related to molecular structure determination is GNOMAD [98]. This is a multi-level iterative algorithm that groups the vertices at the highest level. Next, it attempts to identify an order within each of the groups based on the contribution of each vertex to the total error on the distance constraints. Then, in this order, the vertices are moved using a series of local optimization steps, while making sure all constraints remain satisfied. This algorithm can handle exact distance constraints as well as intervals.

These non-linear optimization methods can be a good compromise between the accuracy of MDS methods and the fast computation times of SPE. Not all methods work well with interval data, such as the graph decomposition methods. On the other hand, methods based on smoothing can be adapted such that they use *hyperbolic smoothing*, which works for interval data [99]. In general, we cannot guarantee the level of quality of the solutions. However, SBB is an example that does provide us with a bound on the expected error. Despite this, the SBB method is not the best option, as it is extremely slow on larger instances [94].

### 1.4.4 Meta-heuristics

A common way to tackle the optimization problem given in Definition 2 is by means of *meta-heuristic searches*. The general idea of these methods is to repeatedly find local minima for the penalty function $\sigma$, until something close to the global minimum has been found. Different meta-heuristics have been tried in the context of Distance Geometry. Two

that were tested in [94] were variations of Multi-start [100] and Variable Neighbourhood Search (VNS) [101]. The most simple of these is Multi-start, which works by repeated local searches (gradient descent) from randomly selected starting points. Multi-start was tested on the DGP in [102]. VNS is a more sophisticated meta-heuristic. Instead of running gradient descent from random starting points, it performs the searches in neighbourhoods of already identified local optima, in an attempt to find the global optimum. These two methods were effective for finding solutions for the DGP, but only for instances on the small side, with $n < 50$ [94]. For larger instances, the method is too slow.

Two meta-heuristic global optimization methods for the DGP were presented in [102]. These methods are variations of the Basin Hopping (BH) algorithm, which was inspired by Monte Carlo minimization and used for finding minimum energy structure for molecules [103]. For solving instances of the DGP, two versions of this algorithm were tried: Monotonic Basin Hopping (MBH) and Population Basin Hopping (PBH). MBH looks like VNS in the way that it repeatedly finds local minima and performs neighbourhood changes. However, in contrast to VNS, a change of neighbourhoods is done by performing a *local move*, which is a procedure that allows us to move from one local minimum to another. The nature of these local moves is very important to the efficacy of the algorithm. Because these algorithms are stochastic, a better choice for the local move procedure will lead to a higher chance of finding a global optimum. Two options for local moves are discussed in [102], showing that a problem-specific local move is the best option. PBH is an extension to MBH that includes performing several MBH runs in parallel, in order to increase the chance of success. Aside from these algorithms, there are a group of methods that are based on the Simulated Annealing (SA) meta-heuristic. These are specialized for protein structure determination and are discussed in Section 3.4.1.

The stochastic meta-heuristics find good-quality solutions for small instance sizes [22, 94, 102]. However, the larger the instances get, the more difficult it becomes for some of these methods to perform well. Just like the optimization techniques mentioned in the previous section, most of these methods do not give any guarantee about the level of the quality of the solutions. They are perhaps best used as a refinement tool, in combination with another DG method. In general, the meta-heuristic methods are quite easily adapted such that they work with interval distances as well as exact distances. These methods differ also in another facet: some of them are general purpose while others need are more specialized. The advantage of the general purpose methods is that generally there is a lot of information about them in the literature, and often their implementation is

straightforward. The specialized methods may be harder to implement, but in the end, they may provide us with better results, especially if we further fine-tune them to the specific application that corresponds to the instance of the DGP that we are attempting to solve.

### 1.4.5 Spectral Projected Gradient

We focus our attention on a specific local optimization method: Spectral Project Gradient (SPG). SPG is often used in the context of Distance Geometry [4, 39]. We highlight this method because it was used in various experiments in this thesis, across different applications. In particular, the algorithm can be used to refine a solution obtained via another approach. For example, it has shown to be an effective refining tool for instances relating to molecular structure determination, where it was used in combination with SDP and classical MDS [104]. SPG can be used for constrained as well as unconstrained optimization. In the Java package (see Section 1.5) for this thesis, we used the constrained variant in combination with branch-and-prune (see Section 1.4.7 and Chapter 2), which has been shown to be a fruitful approach in [9]. Aside from this, we will show experiments in the context of Dynamical Distance Geometry Problem, where the unconstrained variant of SPG was used as the main method for solving the motion retargeting problem. SPG was used as a tool for the motion retargeting problem also in [4, 19, 105, 106].

Given an initial realization $x_0$, the spectral projected gradient is a method that will try to improve the quality of the solution iteratively by minimizing the penalty function $\sigma$. This initial realization $x_0$ can be chosen arbitrarily, but when we use SPG as a refinement method, $x_0$ will generally already be of a decent quality. The main component of SPG is gradient descent, which is a well-known local optimization procedure and one of the principal ingredients of deep learning methods such as neural networks [107–110]. For gradient descent, the step size is an important parameter. If the step size is too large, we might step over a minimum. When it is too small we could take forever to converge. We attempt to address this issue by using a spectral projected gradient method, which means that the step size used throughout the gradient descent is *spectral* [111], based on the underlying local Hessian, rather than using a standard decrease in the objective function. The spectral projected method is combined with a non-monotone line search [112] to speed up the convergence. For this line search method, global convergence was shown for smooth and non-convex functions, and R-linear convergence was proven for strongly convex functions [113]. It is important to note that these convergence properties were shown

for unconstrained optimization problems. In practice, the method is still very effective for constrained optimization.

The pseudocode for the full procedure is given in Algorithm 2 and will be referenced throughout the explanation. This pseudocode is based on [106, 111] and the implementations of SPG in the context of Distance Geometry. SPG operates through a main loop, which runs for a maximal number of iterations $I$, which is given as an input parameter.

**Algorithm 2** Spectral Project Gradient (SPG)

**Input:** Initial solution $x_0$, penalty function $\sigma$, maximal iterations $I$, thresholds $\epsilon$ and $\alpha_\epsilon$, stepsize bounds $\mu_{\min}, \mu_{\max}$, decrease parameter $\gamma$, nonmonotonicity degree $\eta$
**Output:** Final solution $x$

$i \leftarrow 0$, $q_0 \leftarrow 0$
$m_0 \leftarrow \sigma(x_0)$ ▷ Using penalty function $\sigma$
**while** $i < I$ and $\sigma(x_i) > \epsilon$ **do** ▷ Stopping criteria 1 and 2
    **if** i $= 0$ **then**
        $\mu_0 \leftarrow 1$
    **else**
        $y_{i-1} \leftarrow \sigma(x_i) - \nabla\sigma(x_{i-1})$
        $s_{i-1} \leftarrow x_i - x_{i-1}$
        $\mu_i \leftarrow (s_{i-1}^\top s_{i-1})/(s_{i-1}^\top y_{i-1})$
        $\mu_i \leftarrow \min(\mu_{\max}, \max(\mu_{\min}, \mu_i))$ ▷ Using safeguards
    **end if**
    $d_i \leftarrow -\mu_i \nabla\sigma(x_0)$ ▷ Using $\mu_i$ to compute $d_i$
    $d_i = P_\Omega(x_i - d_i) - x_i$ ▷ Projection constraints (only for constrained version)
    $\alpha_i \leftarrow 1$
    **while** $\sigma(x_i + \alpha_i d_i) > m_i + \gamma\alpha_i \nabla\sigma(x_i)^\top d_i$ **do** ▷ Nonmonotone line search
        $\alpha_i \leftarrow \alpha_i/2$
    **end while**
    **if** $\alpha_i < \epsilon_\alpha$ **then**
        **return** $x_i$ ▷ Terminate early (stopping criteria 3)
    **end if**
    $x_{i+1} \leftarrow x_i + \alpha_i d_i$ ▷ Computing next solution and preparing next iteration
    $q_{i+1} \leftarrow \eta q_i + 1$ ▷ $q_i$ is a helper variable for $m_i$
    $m_{i+1} \leftarrow (\eta q_i m_i + \sigma(x_{i+1}))/q_{i+1}$
    $i \leftarrow i + 1$
**end while**
**return** $x_i$ ▷ Output the final solution

Recall the penalty function $\sigma(x)$ which can be used to measure the violation of the

distance constraints for a realization $x$:

$$\sigma(x) = \frac{1}{2} \sum_{\{u,v\} \in E} (\|x_u - x_v\| - \delta(u,v))^2$$

Because this function is differentiable (when no two points have the same position), we can use its gradient to improve a solution $x$. At every iteration $i \in 0, \ldots, I$, a new, improved solution $x_{i+1}$ is computed by performing a step in the opposite direction $d_i$ of the gradient $\nabla\sigma(x_i)$. The direction $d_i$ can be computed from the gradient using the spectral stepsize $\mu_i$ (line 12) and the projection $P_\Omega$ [114, 115] (line 13):

$$d_i = P_\Omega(x_i - \mu_i \nabla\sigma(x_i)) - x_i,$$

$\Omega$ is the feasible convex set onto which we know how to project solutions $x$. Incorporating the projections $P_\Omega$ is what allows us to perform constrained optimization by using SPG. The spectral stepsize $\mu_i$ is computed based on the previous step from iteration $i-1$ to $i$ (line 9):

$$\mu_i = \frac{s_{i-1}^\top s_{i-1}}{s_{i-1}^\top y_{i-1}}.$$

where $s_{i-1} = x_i - x_{i-1}$ and $y_{i-1} = \nabla\sigma(x_i) - \nabla\sigma(x_{i-1})$. This stepsize should be bounded by the parameters $\mu_{\min}$ and $\mu_{\max}$ as a safeguard (line 10). The last ingredient we need is the multiplier $\alpha$, obtained from the non-monotone line search (lines 14-17). Initially, $\alpha_i$ is set to 1.0. Next, $\alpha_i$ is reduced by half until the following stopping criteria are met (line 15):

$$\sigma(x_i + \alpha_i d_i) \leq m_i + \gamma\alpha_i \nabla\sigma(x_i)^\top d_i,$$

where $\gamma \in (0,1)$ is a sufficient decrease parameter and $m_i$ is a variable based on the penalty values $\sigma$ of the previous iterations. Often, a parameter $\eta$ is included in the line search as well, which controls the degree of nonmonotonicity and is used to update $m_i$ throughout the iterations. More information about these parameters and variables as well as an analysis of this step can be found in [113]. Once we have obtained the three ingredients $\mu_i$, $d_i$ and $\alpha_i$, we proceed to compute the next solution $x_{i+1}$ by taking a step in the right direction (line 21):

$$x_{i+1} = x_i + \alpha_i d_i.$$

The above three steps (computation of spectral stepsize, non-monotone line search, and stepping in the opposite direction of the gradient) are repeated until one of several

stopping criteria is reached:

1. The maximal number of iterations $I$ is reached (line 3).
2. The objective value $\sigma(x_i)$ reaches below a certain threshold $\epsilon$, where $\epsilon$ is given as a parameters (line 3).
3. The stepsize multiplier $\alpha$ reaches below a threshold $\alpha_\epsilon$, where $\alpha_\epsilon$ is given as a parameter (line 18).

### 1.4.6 Distance norms and Linear Programming

The optimization variant of the DGP in Definition 2 is non-linear. Firstly, the penalty function $\sigma$ has quadratic terms. Next, the Euclidean distance norm ($L_2$), which we generally have in the applications of the DGP, is non-linear:

$$L_2(x_u, x_v) = \sqrt{\sum_i^K (x_u^i - x_v^i)^2} \tag{1.10}$$

where $x_u^i$ is the $i$-th coordinate of vertex $u$ in the realization $x$.

However, it is possible to linearize this problem when we replace the Euclidean norm ($L_2$) with a linearizable distance measure. In [31], it was proposed to replace the Euclidean distances by the $L_\infty$ norm:

$$L_\infty(x_u, x_v) = \max_i |x_u^i - x_v^i| \tag{1.11}$$

This norm has some useful properties when the DGP instance contains interval distances. When we use the $L_\infty$ norm, these bounds are now linear. In [32] this idea was expanded upon. Here, not only the $L_\infty$ norm was considered, but also the $L_1$ norm, which is also known as taxi-cab or Manhattan distance:

$$L_1(x_u, x_v) = \sum_i^K |x_u^i - x_v^i| \tag{1.12}$$

These alternative distance norms cannot be linearized without the introduction of binary variables. Using these binary variables, it is possible to replace nonlinear functions with piecewise linear forms. This allows us to transform the non-linear optimization problem described in Definition 2 to a linear one. We can specify Mixed-integer programming (MIP) formulations of the now linear problem, which is a special type of linear program

(LP) which allows for a mix of binary and integral variables. The MIP formulations for the $L_1$ and $L_\infty$ norms can be found in [32].

In practice, using Linear Programming for general instances of the DGP is not so effective as it becomes very slow with larger input sizes, due to the combinatorial explosion of constraints [32]. This approach is most promising in the context of the additional local orientation constraints discussed in Section 1.3. For the application of adaptive maps, this approach provides us with good-quality solutions with a low computation time. Experiments with this approach are presented in Chapter 5.

### 1.4.7 Build-Up and Branch-and-prune

In this section, we describe two unique approaches. The methods discussed here do not rely on solving the DGP as an optimization problem or matrix decomposition problem. Instead, these methods iteratively embed the vertices by exploiting geometric properties of the combinations of distances that we have access to.

The Build-Up algorithm was proposed in the context of the DGP for molecular structure determination [116–118]. As the name of the algorithm suggests, we attempt to build up an embedding for the vertices $V$ in a sequential manner. Because the focus was on determining protein structures, the dimension $K$ is set to 3. We start by fixing the coordinates for the first four vertices. Next, we find a vertex $v$ for which we know a distance to at least four vertices that we have already embedded. These four vertices are called *reference* vertices for $v$. When we have a vertex $v$ and four such reference vertices, we can use *trilateration* to compute the coordinates for $v$. We repeat this process until there are no vertices left to embed or we cannot embed any more vertices (in which case the algorithm fails). The original version of the algorithm had an issue with an accumulation of round-up errors, which meant that there were large inaccuracies in the coordinates of the vertices embedded towards the end. An update to the algorithm was made [118] such that these numerical errors were put under control. A second modification to the algorithm made it so only three reference vertices are needed when embedding a vertex $v$ [119]. It is important to note that this approach only works if we have the right number of distances available at the right time. Essentially, we require a specific order on the vertices. Because we do not know beforehand which vertex order will lead to a successful run of the algorithm, this approach is heuristic in nature.

This lead to the development of the branch-and-prune (BP) framework [10]. Similar to the Build-Up algorithm, we iteratively embed the vertices. This framework works for

all dimensions $K$, as long as when embedding a vertex $v$ we have at least $K$ reference distances for $v$. An important difference with the Build-Up algorithm is that in the BP framework, much research has been done on how to identify vertex orders which guarantee that every vertex $v$ will have a sufficient number of reference distances. This means that, given the correct vertex order, the BP algorithms are not heuristics, but exact methods (as long as we are dealing with exact distances). Furthermore, instead of identifying one position at every step, we may identify two or more possible positions for a vertex $v$. Effectively, this allows us to discretize the search space, making it possible for us to enumerate every possible solution to an instance of the DGP that allows for a vertex order that satisfies our requirements. An in-depth description of the branch-and-prune algorithm and its implementations is given in Chapter 2.

The several papers that researched the Build-up algorithm showed that can be very effective for determining protein structures given somewhat sparse distance graphs [117, 118]. The resulting solutions had low Root mean square deviation (RMSD) scores when compared to the structures obtained from x-ray crystallography. The biggest drawback of the Build-up algorithm is that we are not guaranteed to find a solution. When we embed the vertices using a greedy method, we do not know if we will always have the right reference distances available. A second important drawback, when comparing Build-up to BP, is that the requirements for a vertex order for the Build-Up algorithm are stricter than for BP.

Another advantage of the BP framework is that is has several extensions that allow us to solve instances that contain interval distance data [11, 12]. The BP framework is one of the most promising avenues of research for solving the DGP, especially in the context of structural biology. This framework was implemented in the Java package and intensively used for experiments in this thesis.

## 1.5 The Java Package

Several different methods for the DGP were implemented as part of this thesis. Most of these methods were grouped into one Java package. While this Java package is not published in its entirety (as it is still in development), a subset of the classes used in the context of protein structure determination was made public on GitHub [1]. More information about this repository is found in Section 3.5.1. The class diagram of the full

1. `https://github.com/simonheng/BP_ProteinFileReader`

Java package is shown in Figure 1.4. This diagram shows the relations between the several classes in the package based on dependency and inheritance. To avoid the diagram from getting cluttered, duplicate dependencies are not visualized. For example, the utility classes MATHS and GEOMETRY are used by almost all other classes, but these arrows are not included if the dependency is already passed through an different class.

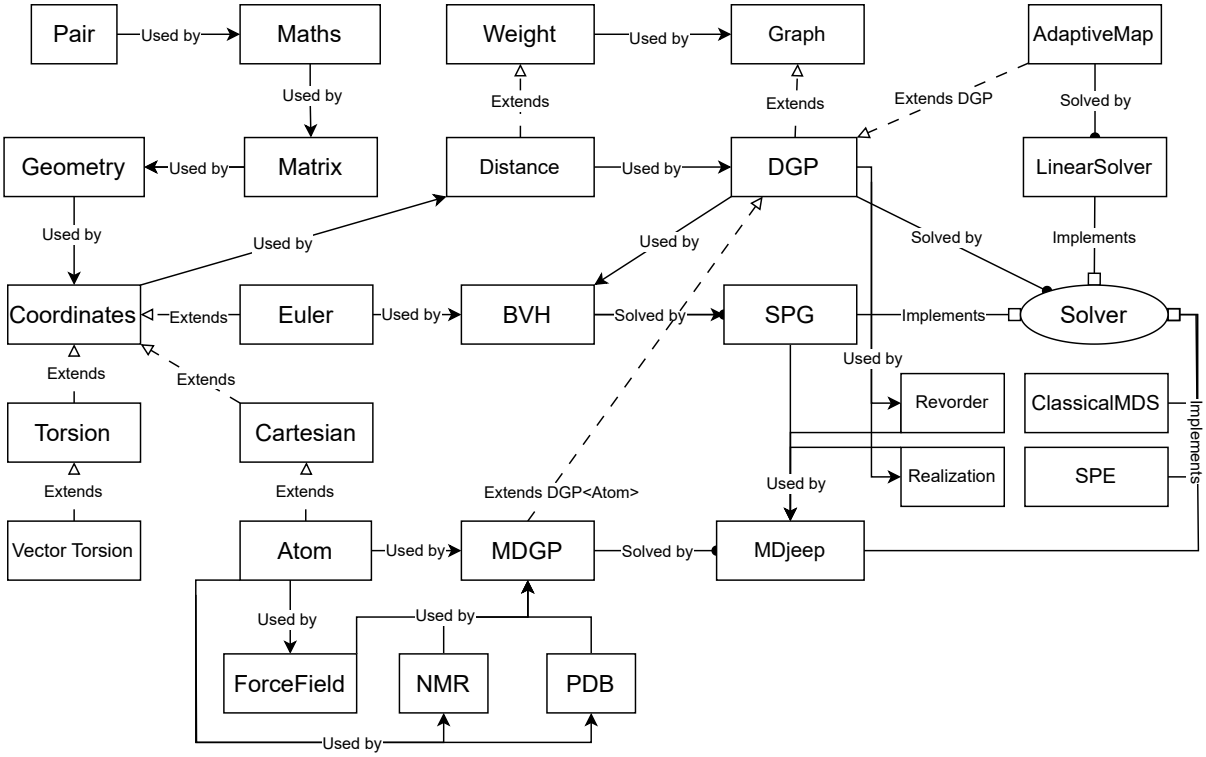



Figure 1.4 – A class diagram representing the java package that was created for this thesis.

In the top left of the diagram we have four classes with subroutines that are used by almost all other classes in the project. PAIR is an object class which represents an immutable tuple, while the MATHS and GEOMETRY are utility classes containing static methods corresponding to subroutines such as complex number division and sphere intersections (Section 2.3). MATRIX is an object class for matrices, based on the JAMA package [2].

We use the graph representation of the DGP, which is why we include the object classes GRAPH and WEIGHT to describe weighted directed and undirected graphs. The DGP object class is an extension of the GRAPH class where the weights are DISTANCE

2. `https://math.nist.gov/javanumerics/jama/`

objects. The vertices of the graph described by the DGP class are extensions of the abstract Coordinates class. For each of the different applications of the DGP, the package contains an object class that extends the DGP class which represents the instances for the application in question. For protein structure determination (Chapter 3) we have the class MDGP, which stands for Molecular DGP. The vertices of the graph that MDGP describes are of the Atom type, which is a special implementation of the Cartesian coordinate subclass. MDGP also relies on three other classes which deal with molecular input data: ForceField, NMR and PDB. More information about this is given in (Chapter 3). For motion retargeting (Chapter 4), we have the BVH class, which uses the Euler coordinate extension class for its vertices. The Vector Torsion object class represents the new representation that we propose for human motions, described in Section 4.4. Finally, for adaptive maps (Chapter 5) we have the AdaptiveMap class which uses Cartesian vertices, where the dimension $K$ is 2.

The different methods of solving the DGP are included in the Java package as implementations of the Solver interface. The solutions to the problems may be represented using the Realization class. Some of these solvers work for any extension of the DGP class while other work for only specific ones. The MDjeep class contains an implementation of the branch-and-prune framework (Chapter 2) which only works if the DGP is discretizable. As we will see in Chapter 3, this is the case for the instances that arise for protein structure determination, which is why in the Java package MDjeep is used specifically to solve instances represented by the MDGP class. The class Revorder is used to make sure that we have an order of vertices which allows us to discretize the search space (see Section 2.1 and Definition 5). As we will see in Chapter 5, we use Linear Programming in the context of Adaptive Maps, which is why the LinearSolver class works specifically with the Adaptive Map class. The LinearSolver class relies on the Java interface provided by CPLEX [120] using the Concert Technology. The Classical MDS class works on DGP objects as long as the coordinates are Cartesian and the graph is complete. The classical MDS (Section 1.4.1) method was implemented as a benchmark for the linear programming technique used for adaptive maps. As a benchmark for the more general methods, SPE (Section 1.4.2) was implemented in the SPE class, which is compatible with any of the DGP (sub)classes that use Cartesian coordinates. The implementation of SPG (Section 1.4.5) works with any extension of the DGP object. When the coordinates used are Cartesian, SPG uses an exact method to compute the gradient. This is the case when SPG is used as a refinement step for MDjeep (more

on this in Section 2.4.2). In the context of motion retargeting (Chapter 4), where the coordinates are EULER (or perhaps in the future VECTOR TORSION), SPG relies on a numerical differentiation method (finite differences) [121] to approximate the gradient.

Chapter 2

# Discretizable Distance Geometry

Given certain assumptions, the search space relating to an instance of the Distance Geometry Problem can be *discretized*. In fact, we can construct a tree in which each node corresponds to a candidate position for a vertex $v$. Every path from the root node to a leaf node corresponds to a possible solution to the DGP instance at hand. Combinatorial methods may be used to explore the tree and enumerate every possible solution for the given DGP instance. In particular, we will look at the branch-and-prune framework and its implementations for the dimension $K = 3$. As part of this thesis, one of the two open-source branch-and-prune implementations was implemented into Java with some extra features.

In this chapter, we begin with the main theory of discretization and vertex orders in Section 2.1. Next, in Section 2.2, we describe the algorithmic framework of branch-and-prune, which is the method that exploits the discretization. Section 2.3 discusses several methods that can be used for generating coordinates during the branching step of BP. Afterwards, in Section 2.4 we investigate how the different implementations handle interval distances in the input. Finally, we discuss how including torsion angles in the DDGP instances lets us decrease the size of the search space (Section 2.5).

## 2.1 Introduction

It has been shown that, when several assumptions are met, a DGP instance $G = (V, E, d)$ with $K > 0$ can be *discretized* [6]. The discretization of the search space of such an instance makes it possible for combinatorial methods to enumerate all feasible solutions for the problem at hand. This is of great interest, especially in the applications of molecular structure determination (see Chapter 3), because it allows us to identify distinct protein structures given inter-atomic distances. In fact, in the context of protein structure determination, the instances often satisfy the required assumptions for discretization. For more information about which inter-atomic distances are required to discretize, see Sections 3.3 and 3.4.3. The subclass of the DGP which captures such instances is referred to as the Discretizable Distance Geometry Problem (DDGP). The DDGP was shown to be NP-hard [6, 8].

As part of this thesis, an explanatory video concerning this subclass of instances was created and entered to the Summer of Math Exposition (SoME) 22 contest. It can be found on Youtube [1]. In order to discuss this subclass and the necessary *discretization assumptions*, we need to partition the set of edges $E$. Let $E'$ be the subset of edges in $G$ that are related to exact distances. As a consequence, $E \setminus E'$ is the subset of edges in $G$ that contain all the interval distances. We can now give a formal definition of the DDGP [6, 26]:

**Definition 4** *A simple weighted undirected graph $G = (V, E, d)$ represents a DDGP instance in dimension $K$ if and only if there exists a vertex ordering on $V$ such that the following two assumptions are satisfied:*

- **(a)** *$G[\{1, 2, \ldots, K\}]$ is a clique whose edges are in $E'$;*
- **(b)** *$\forall v \in \{K+1, \ldots, |V|\}$, there exist $K$ vertices $u_1, u_2, \ldots, u_k \in V$ s.t.*
  - **(b.1)** *$u_1 < v$, $u_2 < v$, $\ldots$, $u_K < v$;*
  - **(b.2)** *$\{\{u_1, v\}, \{u_2, v\}, \ldots, \{u_{K-1}, v\}\} \subset E'$ and $\{u_K, v\} \in E$;*
  - **(b.3)** *$\mathcal{V}_S(u_1, u_2, \ldots, u_K) > 0$ (if $K > 1$),*

*where $G[\cdot]$ is the subgraph induced by the subset of vertices of $V$ given in argument and $\mathcal{V}_S(\cdot)$ is the volume of the simplex $S$ generated by a valid realization of the vertices $u_1, u_2, \ldots, u_K$.*

1. `https://www.youtube.com/watch?v=ZTR8txn2wBU`

The first assumption **(a)** lets us fix the coordinate space of the instance at hand. Because of assumptions **(b.1)** and **(b.2)**, at least $K$ other distinct vertices, preceding the current vertex $v$ in the given vertex ordering, can be used as a reference for positioning the vertex $v$. Because of this notion, these vertices are referred to as *reference vertices* and the distances $\delta(u_1, v), \delta(u_2, v), \ldots, \delta(u_K, v)$ are referred to as *reference distances*. Moreover, assumption **(b.2)** makes sure that at most one of the $K$ available distances has a large enough degree of uncertainty to be considered as an interval distance, while at least $K-1$ other distances can be considered as "exact". These two previous conditions ensure that for every vertex $v \in V$ to be placed, there is only a "small" subset (under some conditions, a discrete subset) of feasible positions for that vertex. This is because the $K$ reference distances can be used to define $K$ Euclidean objects, which we can then intersect to obtain a discrete number of positions for $v$. For each reference vertex $u_i$ for which $\{u_i, v\} \in E'$, this object is a $K$-sphere, which we construct with radius $\delta(u_i, v)$. If one of the reference vertices ($u_K$) relates to an interval distance, we will instead construct a $K$-spherical shell, where the thickness of the shell is linked to the size of the interval. The assumption **(b.2)** is necessary because if more than one of the reference vertices would relate to an interval distance, the intersection between the Euclidean objects would not correspond to a disjoint set of arcs or points which would mean that we cannot use the algorithms that will be discussed in this chapter. The final assumption **(b.3)** makes sure that these $K$ Euclidean objects will have at most two disjoint sets as intersection. In the case that all $K$ reference distances are exact, these $K$-spheres will intersect in at most two positions where we know that all reference distances are satisfied. These are the candidate positions for $v$. When we instead intersect $K-1$ spheres and 1 spherical shell, we obtain at most two disjoint arcs. The vertex $v$ may be anywhere inside these two arcs, which means the problem is still continuous. These arcs may then be sampled so that we can still obtain a discrete number of positions for $v$. More details on this will be given in Section 2.4. This discretization lets us model the domain of the search space as a tree, where we have a layer for every vertex $v$. Each possible position of $v$ will correspond to a node in this layer. An example of such a tree is given in Figure 2.1. Note that the first $K$ vertices (that form a clique in $G$) are fixed, so we have only one candidate position for each of them (modulo isometries).

To be able to satisfy the assumptions in Definition 4 we need a specific order on vertices. Given an unordered instance of the DGP and reordering it such that the assumptions are met is not a trivial task. For instances related to molecular structure determination, different handcrafted orders have been proposed in the literature [11, 122, 123]. Reorder-

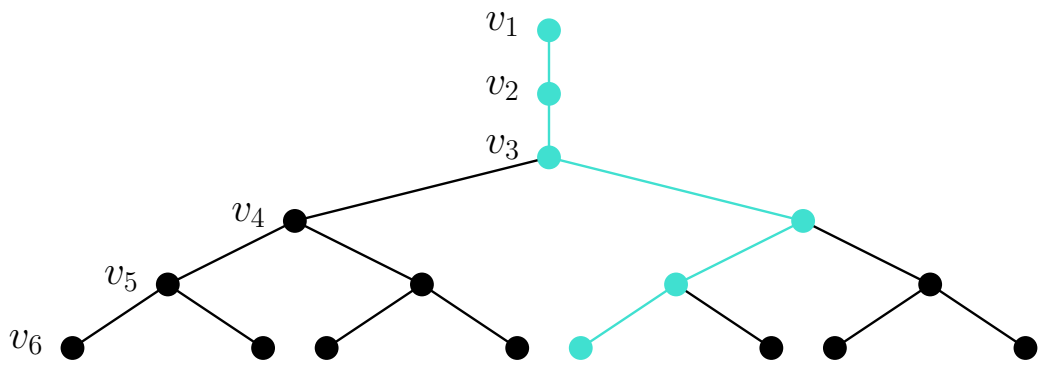



Figure 2.1 – An example of a tree representing the search space of an instance of the DDGP with $|V| = 6$ and $K = 3$. A path in the tree, corresponding to a possible solution to the instance is highlighted.

ing the vertices of $V$ such that the discretization assumptions are met corresponds to a problem in the literature known as the Referenced Vertex Ordering (Revorder) [124] problem. Let $\sigma$ be a vertex order with ranks $1, \ldots, n$, where $n = |V|$ and let $R_\sigma(i)$ be the reference set of the vertex $v$ with rank $i$ in the order $\sigma$. The Revorder problem can be defined as a decision problem as follows [124]:

**Definition 5** *Given a simple directed graph $G = (V, E)$, a positive integer $K$ is there a vertex order $\sigma$ such that:*

$$\begin{cases} \forall i \in \{1, \ldots, K\}, & |R_\sigma(i)| = i - 1 \\ \forall i \in \{K+1, \ldots, n\}, & |R_\sigma(i)| \geq K \end{cases}$$

A vertex order that satisfies the above definition is called a *referenced order*. Note that this definition does not distinguish between reference vertices relating to exact and interval distances. However, the definitions and the algorithms discussed in this section can be trivially extended so that the assumptions from the previous section are precisely met.

In the literature, several works have been published which study a problem that is similiar to the Revorder problem [8, 125]. This problem is known as the Discretization Vertex Order Problem (DVOP) [125] and is in fact a specific case of the Revorder problem where an extra assumption is included. This assumption is known as the *consecutivety assumption*, which requires that the reference set of a vertex $v$, together with $v$ itself, are consecutive in the vertex order. However, this is not strictly required for discretization, and thus in later works this additional assumption was dropped.

For the DVOP, a greedy algorithm was proposed in [8]. This algorithm was greatly improved upon in [124], which extended it to the more general Revorder problem. The greedy search starts from an initial clique (of size $K$) and attempts to build a referenced order from there. The running time of this greedy algorithm is $\mathcal{O}(n \log(n) + m)$, where $n = |V|$ and $m = |E|$. The same paper also proposed other algorithms and branch-and-bound enumeration schemes that pertain to the Revorder problem. The greedy algorithm was implemented into Java during this thesis, and used for various experiments (the Revorder class in Figure 1.4).

Once we have identified a suitable order for the vertices, either by using an algorithm or by following a handcrafted order, we may proceed with exploring the full search space of the tree. This can be done by utilizing the branch-and-prune algorithm, which is discussed in the next section.

## 2.2 Branch-and-prune in dimension 3

The branch-and-prune (BP) [10–12] algorithm is the method that exploits the discretization of the DDGP instances. We will focus on the case where $K = 3$, which matches the application of structural biology. In this section, we will describe the general branch-and-prune framework. In the next sections, we will describe how each phase of the algorithm may be handled or implemented differently.

The algorithm operates in three main phases, the *initialization phase*, the *branching phase*, and the *pruning phase*. In the initialization phase, we embed the first three vertices $u, v$, and $w$ in the order, which is possible because these three vertices are a clique in the graph (see assumption **(a)** in Definition 4). For example, we may fix the first vertex directly at the origin: $x_u = (0, 0, 0)$. We can then use the distance $\delta(u, v)$ to place the second vertex, e.g. $x_v = (-\delta(u, v), 0, 0)$. To place the third vertex $w$, we draw two circles. One around $u$, using as radius $\delta(u, w)$ and one around $v$ using radius $\delta(v, w)$. These circles intersect in two points, which are both candidate positions for $w$. We can simply pick one of the two and use it to place the third vertex $w$ (the other intersection point leads to a symmetric solution). Essentially, in this phase, we are fixing the coordinate space.

After the initialization phase, the next step is to iteratively or recursively embed the rest of the vertices, exploiting the vertex order. We do this by switching between the branching phase and the pruning phase until we have identified all valid candidate positions for every vertex $v \in V$.

In the branching phase, we perform the sphere intersections to compute the possible embeddings of a vertex $v$. This step is illustrated in Figure 2.2. Different methods can be used to compute these candidate positions, and this differs among the implementations. Section 2.3 will discuss these various methods. In the case that one of the reference distances is an interval, several approaches can be taken so that we may still discretize the search space. This will be further investigated in Section 2.4. It is important to note that when all distances are exact, BP is an exact method that guarantees to enumerate all possible solutions to the DGP instances at hand. However, when we have interval distances, the sampling methods that the implementations use are heuristic, and thus the algorithm loses this property of exactness.

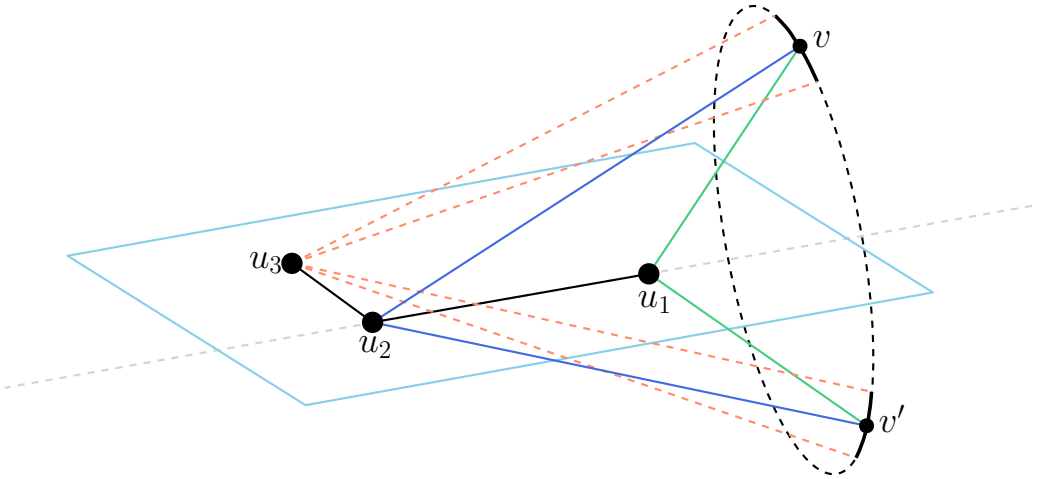



Figure 2.2 – An illustration how a vertex $v$ is embedded in dimension $K = 3$, using three reference distances. The distance $\delta(u_3, v)$ may be an interval distance.

Once we have obtained a candidate position for a vertex $v$ in the branching phase, we validate it during the pruning phase. In this phase, we verify if there exists any other vertex $u < v$ for which we have a known distance $\delta(u, v)$ and that is not one of the reference vertices. For each of such vertices $u$, we check whether the computed position for $v$ satisfies the distance $\delta(u, v)$. If not, we reject this candidate position and prune the corresponding branch. The distances $\delta(u, v)$ are referred to as *pruning distances*. When a pruning distance $\delta(u, v)$ is checked, the implementations generally use a tolerance parameter $\epsilon$, allowing for a small error. The reason for this is to take into account noise that may be included in the input distances. Other information, aside from distances, may be verified as well in this phase. In [126], experiments were conducted in the context of molecular structure determination, where different pruning devices were used. One of the pruning devices

that was tested was based on verifying torsion angles $\tau$, that were part of the instances, as we described in Section 1.3. However, as we will see in Section 2.5, it may be more advantageous to directly use the torsion angle information during the branching phase so that we can avoid creating infeasible branches.

Two open-source BP implementations for $K = 3$, which are both written in C, are publicly available: IBP-NG [2] [127] and MDJEEP [3] [7]. A multi-threaded implementation of IBP-NG is also available, known as TAIBP [4] [128]. MDJEEP was incorporated in the Java package for this thesis (Figure 1.4), so that several new features could be tested. While, compared to C, Java is slower at execution time, it is easier to permute which means different ideas can be implemented and tested at a higher rate. Table 2.1 shows concisely how the various implementations overlap and how they differ. One aim of the Java implementation of MDJEEP was to combine the best parts of both MDJEEP and IBP-NG.

| | IBP-NG (C) | MDJEEP (C) | MDJEEP (Java) |
|---|---|---|---|
| **Error tolerance** | Tolerance parameter | Tolerance parameter $\epsilon$ | Tolerance parameter $\epsilon$ |
| **Tree traversal** | Iterative (depth-/breadth-first) | Recursive (depth-first) | Recursive (depth-first) |
| **Coordinate generation (Section 2.3)** | Clifford algebra [129] | Matrix approach [130] | Matrix approach [130] |
| **Intervals (Section 2.4)** | Sampling the arcs | Refinement by SPG | Refinement by SPG |
| **Torsion angles (Section 2.5)** | Used for branching | Purely distance based (sign is lost) | Used for branching |

Table 2.1 – An overview of the similarities and differences between the three implementations.

To further illustrate the BP algorithm and its framework, the pseudocode for the main recursive step in MDJEEP, based on the Java implementation, is shown in Algorithm 3. In the Java implementation, and in this pseudocode, at every recursive step we build a candidate list $L$ (line 6) for a vertex $v$ (at position $i$ in the order). During the coordinate generation phase (lines 7-13), we populate the candidate list with either directly com-

2. `https://github.com/geekysuavo/ibp-ng`
3. `https://github.com/mucherino/mdjeep`
4. `https://github.com/tmalliavin/TAiBP`

puted candidate positions (three exact distances) or samples from the arcs (one interval distance). More information about this step can be found in Section 2.3. Next, for each of these candidate positions, we verify its feasibility (line 15). If the candidate $x_v$ is feasible, we move on to the next layer $i + 1$. If it is not feasible, MDjeep carries out an extra refinement step using local optimization, in order to attempt to satisfy the pruning distances, before moving on to the next candidate $x_v$. More about this refinement step can be found in Section 2.4.2.

**Algorithm 3** The main recursive step of MDjeep: BP($i$, $x$)

**Input:** Index $i$ of the vertex $v$ in a discretization order $\sigma$, incumbent solution $x$

```
 1: if i > |V| then
 2:     save current solution x                                  ▷ We are done
 3: else
 4:     v ← vertex at position i in σ
 5:     u_1, u_2, u_3 ← reference vertices for v
 6:     L ← empty list of candidate positions
 7:     if u_3 ∈ E \ E' then                  ▷ We have one reference interval distance
 8:         compute at most two candidate arcs                   ▷ See Section 2.3
 9:         add samples from both arcs to L                      ▷ See Section 2.4
10:     else
11:         compute at most two candidate positions              ▷ See Section 2.3
12:         add candidate positions to L
13:     end if
14:     for x_v ∈ L do
15:         if x_v is feasible then                              ▷ Pruning phase
16:             BP(i + 1, x)          ▷ Move on to the next candidate vertex in the order
17:         else                      ▷ Local optimization refinement (see Section 2.4.2)
18:             refine current solution x and verify feasibility again
19:             if still infeasible move on to next candidate position x_v
20:             else BP(i + 1, x), with refined solution x
21:         end if
22:     end for
23: end if
```

## 2.3 Coordinate generation

In the branching phase of BP, when $K = 3$, we compute the intersection of either three spheres or two spheres and one spherical shell in order to identify the candidate positions

of a vertex $v$. When we have three reference vertices $u_1, u_2$ and $u_3$, this corresponds to solving the following system of quadratic equations:

$$\begin{cases} ||x_v - x_{u_3}||^2 = \delta(u_3, v)^2 \\ ||x_v - x_{u_2}||^2 = \delta(u_2, v)^2 \\ ||x_v - x_{u_1}||^2 = \delta(u_1, v)^2 \end{cases} \tag{2.1}$$

In one of the earliest implementations of BP [6], where only exact distances were considered, solutions to the above quadratic system were identified by solving two linear systems [131]. However, this approach lead to a build-up of errors, which was especially noticeable for larger instances. That is why in later works, a different method was used altogether. Instead of solving the quadratic system given in (2.1), we can instead exploit the torsion angle $\tau$, defined by the two planes defined by the triplet of vertices $(u_3, u_2, u_1)$ and $(u_2, u_1, v)$. Figure 2.3 shows the relation between this dihedral angle, and the candidate positions for $v$. The torsion angle $\tau$ can be computed by exploiting the cosine law (see [8]). The necessary distances are available either because of assumption **(b)**, or we may calculate them from the positions $x_{u_1}, x_{u_2}$ and $x_{u_3}$. If $\delta(u_3, v)$ is an interval distance, we can instead compute the corresponding interval $(\underline{\tau}, \overline{\tau})$ and use the methods discussed below to compute the endpoints of the arc. When we compute $\tau$ from the distances we do not have any information about the sign and are forced to consider both candidate positions $v_+$ and $v_-$. Alternatively, $\tau$ can be part of the DGP instance (see Section 1.3). When we have the signed $\tau$ included in the input, we may a priori disregard the branches corresponding to the incorrect sign of the angle. This is further discussed in Section 2.5.

Using $\tau$, the positions $x_{u_1}$, $x_{u_2}$ and $x_{u_3}$ as well as the three distances $\delta(u_1, v), \delta(u_2, v)$ and $\delta(u_3, v)$, we can compute the Cartesian coordinates $x_v$. Different methods have been used for this step. In [8, 11], a matrix multiplication method was proposed which was very efficient in practice. However, it relied on the consecutivity assumption (mentioned in Section 2.1), which means that $u_1, u_2$ and $u_3$ have to be consecutive in the order. Instead, in [130], a new method was put forward that does not require the consecutivity. Furthermore, practical tests showed that the method performs well and that there is little to no build-up of errors. This approach is used in the BP solver MDjeep and was also implemented as part of this thesis and used for the experiments discussed in later chapters. This coordinate generation method prevents error build-up by avoiding the accumulation of matrices when we step from one vertex to the next one. Instead, only one matrix $U$ is

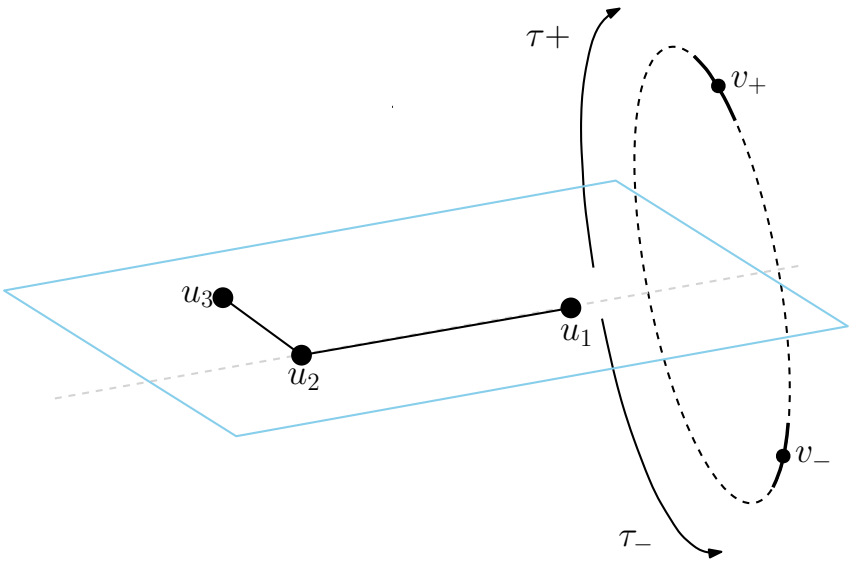



Figure 2.3 – The candidate positions $v_+$ and $_v-$ can be computed by exploiting the dihedral angle $\tau$. In case we know the sign of $\tau$ (given in the input), we can consider only one of the two candidate positions or arcs.

defined, which can be used to convert a vertex position in the coordinate system defined by $u_1$ to the canonical coordinate system. The three columns of $U$ itself are the unitary vectors $\hat{x}, \hat{y}, \hat{z}$ which correspond to the three axes of the coordinate system centered in $u_1$. The first column, $\hat{x}$ is defined by the vector $\overrightarrow{u_2u_1}$. The third column, $\hat{z}$ can be obtained by the vectorial product $\overrightarrow{u_2u_1} \times \overrightarrow{u_2u_3}$. Finally, the second column $\hat{y}$ is the result of the vectorial product of the other two column vectors: $\hat{y} = \hat{x} \times \hat{z}$. Once we have $U$, we can use it to obtain the coordinates of a candidate position $x_v$ as follows:

$$x_v = x_{u_1} + U \begin{bmatrix} -\delta(u_1, v)\cos\tau \\ \delta(u_1, v)\sin\zeta\cos\tau \\ \delta(u_1, v)\sin\zeta\sin\tau \end{bmatrix} \tag{2.2}$$

where $\zeta$ is the vector angle defined by the triplet $u_2, u_3, v$ which can be obtained using the cosine law. A very similar method for the conversion of torsion space to Cartesian space is presented in [132]. Here, a matrix called $\hat{\hat{M}}$ (corresponding to the above $U$) is used. The method is referred to as the Natural Extension Reference Frame (NeRF). NeRF is used in the software Rosetta [133], which can be used for analyzing protein structures, which is an application where torsion angles are often encountered.

Aside from the above methods, there are many different ways to compute the candidate positions $x_v$ from the geometric information that we have available during the

branching step. For example, the BP implementation IBP-NG relies on a specific (usually handcrafted) vertex order and Clifford algebra when computing the vertex positions [127, 129].

## 2.4 Handling interval distances

In the previous section, we discussed the tools that we can use to compute the candidate positions for a vertex $v$. However, in the case that we have a vertex $v$ for which one of the reference distances is an interval, we can only use the coordinate generation methods to obtain the endpoints of the arc as well as the corresponding interval on the dihedral angle $\tau$. In this case, we need to take some extra measures to obtain a discrete number of candidate positions for each vertex $v$. There are two main methods that we can use, which we will discuss below. As mentioned earlier, both these methods are heuristic which means that for interval distances, BP is no longer an exact method, as we cannot guarantee that all possible solutions will be identified.

### 2.4.1 Sampling the arcs

The first method we will look at is straightforward and is how the implementation IBP-NG handles the interval distances (see [127]). The algorithm is parameterized by a parameter, which in IBP-NG is called the *branching parameter* $B$. In case we encounter a vertex $v$ for which we have a reference interval distance, we simply sample $B$ points from the computed arc. In practice, this is done by cutting the interval distance into $B$ pieces. For each sub-interval, we select the middle point as a candidate position for $v$ and use it to create a new branch. For a visual example, see Figure 2.4 (a). This means that in practice, IBP-NG will generally create 2 branches for every $v$ with exact reference distances, and $2B$ for every $v$ with an interval reference distance. In the case that IBP-NG is supplied with torsion angle information, only 1 or $B$ branches are created at every branching step (we can disregard half the branches). The choice of the parameter $B$ is vital for this approach. In case it is picked too small, the chance is high that the correct sample position is missed and the tree is pruned entirely. When it is picked too large, the size of the tree may explode and the algorithm will take very long to terminate. Even when $B$ is a large number, there is a significant chance that this approach will not be able to find a solution for the DDGP instance at hand. This is especially the case in which we

have a large number of interval distances or a high degree of noise. To avoid this problem, MDjeep relies on a different method for handling intervals.

### 2.4.2 A coarse-grained representation and local optimization

In MDjeep, a combination of a coarse-grained representation of the search space [9] and a refinement step is used. For this, a resolution parameter $\rho$ is included for the input. The key point is that we define boxes around each candidate position of a vertex. In a later refinement step, the points will be allowed to move within these boxes. Consider the case in which we have a vertex $v$, and all its reference distances are exact. We are quite certain about the position of $v$, so we define a very small box, where its dimensions are equal to the tolerance parameter $\epsilon$. When we have a vertex $v$ with two exact reference distances and one interval, we first compute the length of the arc induced by the interval distance. If the length of the arc is smaller than $\rho$, we fit the entire arc into one box (see Figure 2.4 (b). When it is larger than $\rho$, we divide the arc into equally sized sub-arcs smaller than $\rho$, and cover each such piece with a box (see Figure 2.4 (c)). Within these boxes, the vertex is initially positioned on the arc or sub-arc, near the center of the box. For each such box, we create a branch in our tree. When we use this method, the tree that represents the search space is different from the classical branch-and-prune tree. Each node no longer only describes a candidate position of a vertex $v$, but also its surrounding box.

These boxes come into play in the pruning phase of the algorithm. When we encounter a pruning distance that is not satisfied we do not immediately prune. First, we run a constrained local optimization method, minimizing the error on the distances and moving all the vertices within their boxes. For this refinement step, we use a Spectral Project Gradient (SPG) method as described in Section 1.4.5. The SPG method is included in the Java package (see Figure 1.4). In case local optimization manages to satisfy the pruning distances, we can continue with the next vertex. Otherwise, the branch is pruned. In [13], it was noted that when we use box sizes that are too small, this can lead to issues for SPG. This is because bounds that are too strict may harm its ability to converge to a local optimum. Furthermore, in each refinement step, the reference vertices of a vertex $v$ may be moved. Therefore, the box corresponding to a candidate position of $v$ should be expanded. In practice, this issue is largely solved by re-centering the box around the new candidate position obtained after each refinement step. These expanding and re-centering routines are also included in the Java version of MDjeep. Apart from its use for the size

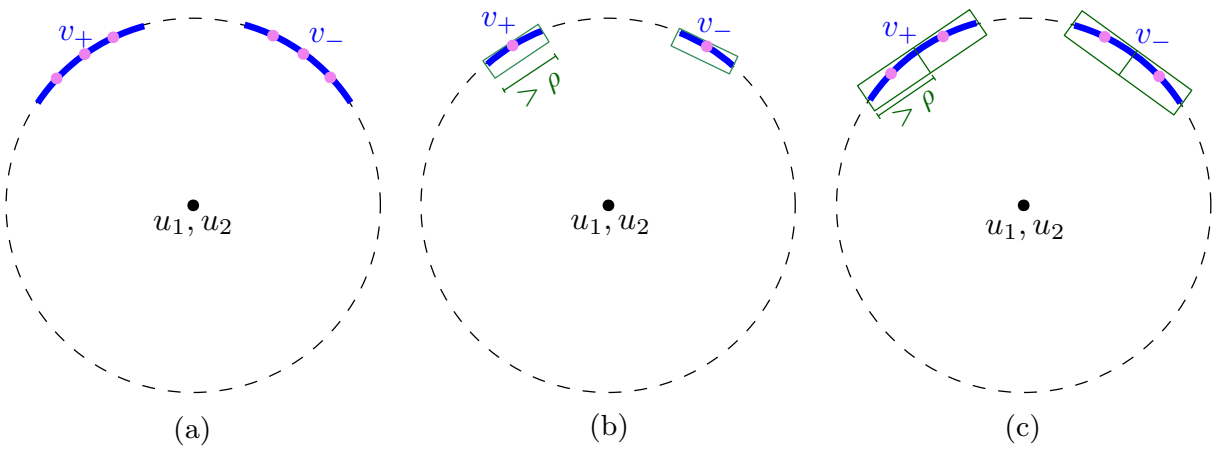



Figure 2.4 – In dashed lines, the circles obtained by intersecting the two spheres centered in $u_1$ and $u_2$.
(a) The strategy of selecting a predefined number $B$ samples. In this example, $B = 3$, so six branches would be created.
(b) The arc is smaller than the resolution parameter $\rho$. We fully cover the arc with a box and initialize the candidate position $v_+$ in the center of the arc. We create 2 branches.
(c) The arc is larger than the resolution parameter $\rho$. We cover the arc with equally sized boxes and initialize the four candidates positions $v_+$ in the center of their sub-arcs. We create four branches.

of the boxes, the resolution parameter $\rho$ has a second application as well [9]. If at least one solution has already been found we can use $\rho$ to verify if any new solution is too close to this previously found solution. For example, when the BP algorithm is currently exploring alternative candidates for a given vertex $v$ to only consider positions $x_v$ that are at a distance larger than $\rho$ away from the position of $v$ in the previously found solution. Alternatively, after generating a new solution, one can measure the Root mean square deviation (RMSD) between the new solution and the previous one. If the RMSD is lower than some threshold (one could use $\rho$), two solutions should be considered too close, and the current branch should be discarded.

Compared to the sampling method, this refinement approach improves the chances of identifying solutions. This was confirmed in [134], where an intermediate version of Java MDjeep and ibp-ng were compared experimentally. However, since we are using local optimization, we cannot guarantee that every possible solution will be found.

## 2.5 Torsion angles and the search space

Recall that in Section 1.3, we discussed certain DGP instances where not only distance information is captured, but also torsion angles. These instances are very relevant for the branch-and-prune methods. When we compute candidates for a vertex $v$, given reference

vertices $u_1, u_2$ and $u_3$, the problem of intersecting the three spheres (or two spheres and one spherical shell) can be reduced to the problem of computing and exploiting a torsion angle $\tau$. However, in the case that we have a torsion angle $\tau$ in the instance, we can forego computing it. Furthermore, we will have access to the sign of the dihedral angle, which means we may disregard the branch corresponding to the other sign (see Figure 2.3). Including the torsion angles in the input not only discontinues our use for the distance $\delta(u_3, v)$ but also lets us greatly reduce the size of the search space. Such instances allow for the definition of the DDGP$_3$, specific to dimension 3, based on Definition 3 in Section 1.3. We will use the same weighted directed hypergraph $G(V, E, d)$ to represent the instances. Recall that the hyperedge set $E$ is partitioned into $E_\delta$ (a pair of two singletons) and $E_\tau$ (spanning a quadruplet of vertices), where $E_\delta$ relates to the distances and $E_\tau$ corresponds to a torsion angle in the input. We will further partition the edgeset $E_\delta$ such that the edges $E'_\delta \subset E_\delta$ are the edges that relate to exact distances $\delta(u, v)$. The weighting function $d$ maps the edges $E_\delta$ to the distance $\delta$ and the edges $E_\tau$ to a torsion angle interval $\tau$.

**Definition 6** *A weighted directed hypergraph $G = (V, E, d)$ represents a DDGP$_3$ instance in dimension 3 if and only if there exists a vertex ordering on $V$ such that the following two assumptions are satisfied:*

- **(a)** *$G[\{1, 2, 3\}]$ is a clique whose edges are in $E'$;*
- **(b)** *$\forall v \in \{4, \ldots, |V|\}$, there exist 3 vertices $u_1, u_2, u_3 \in V$ s.t.*
  - **(b.1)** *$u_1 < v$, $u_2 < v$, $\ldots$, $u_3 < v$;*
  - **(b.2)** *$\{(\{u_1\}, \{v\}), (\{u_2\}, \{v\})\} \subset E'_\delta$ and either $(\{u_3\}, \{v\}) \in E_\delta$ **or** $(\{u_3, u_2, u_1\}, \{v\}) \in E_\tau$;*
  - **(b.3)** *$\mathcal{V}_S(u_1, u_2, u_3) > 0$*

The main difference can be found in assumption (**b.2**), where we now require either the edge $(\{u_3\}, \{v\}) \in E_\delta$ or the torsion angle formed by the reference vertices $\{u_3, u_2, u_1\}$ and $v$ to be included in the instance ($(\{u_3, u_2, u_1\}, \{v\}) \in E_\tau$). Consider the case where, given the vertex order, for every vertex $v$, we have such a torsion angle $\tau(u_3, u_2, u_1, v)$ available. If all reference distances are exact and the torsion angles are precise values, we will end up with a very narrow tree, with only one solution (given a feasible instance). In case we have an instance including reference interval distances, the search space will only grow as a result of these interval distances, and the lengths of the induced arcs (or the branching parameter $B$, if we branch by sampling). This case is of interest because when dealing

with certain instances related to protein determination, we have access to the torsion angle $\tau(u_3, u_2, u_1, v)$ for (almost) every vertex $v$ the vertex order (see Section 3.3). For these instances, we can greatly reduce the search space when we directly use the torsion angle values in the branching phase. This is done by IBP-NG, but not by the original C-version of MDJEEP, which is purely distance-based. The Java implementation of MDJEEP does use the torsion angles during branching. Of course, the solutions identified by the Java-version of MDJEEP will be a subset of the solutions found by the C-version. However, in practice we often stop execution after a certain number of solutions are identified. These first solutions have a higher chance of being accurate when we use the sign of the torsion angles in the branching phase.

Chapter 3

# Molecular Structure Determination

One of the most important applications of the DGP, and the main topic of this thesis, is *molecular structure determination* [27, 33, 37–45]. The work done in the context of this thesis focuses only on *proteins* which are a specific type of molecules. However, a lot of the work may be extended to other types of molecules as well. Proteins are molecules that perform vital functions in our bodies. They make up about 15-20% of the human body [135]. Additionally, proteins are important components of medicines, such as vaccines. How the human body reacts to a medicine relates strongly to the three-dimensional structure of the proteins that make up the treatment. This makes protein structure determination a topic of great practical importance and by extension one of the most researched applications of the Distance Geometry Problem. In this application, the vertices $V$ of the graph $G$ represent atoms of a given molecule, and the distance information $d$ associated with the edges $E$ of the graph reflects inter-atomic distances. These distances may either be derived from the chemical geometry of the molecules or experimental techniques, such as Nuclear Magnetic Resonance (NMR) [2]. The DGP instances that we have in the context of this application satisfy the special conditions discussed in Chapter 2 that let us discretize the search space [6, 8, 26], and use an algorithm that can enumerate all the possible conformations of the protein at hand.

We will start this chapter with an introduction to the problem of protein structure determination (Section 3.1). Next, in Section 3.2, we review the experimental and machine learning methods that we can use. In Section 3.4 we look at which distances are available in the protein instances and we review the use of different DGP and DDGP methods in the literature. After this, Section 3.5 will cover different experimental works done as part of this thesis, including experiments with real protein data. Finally, in Section 3.6 we will summarize the chapter and draw some conclusions.

## 3.1 The problem of Protein Structure Determination

Proteins are molecules that are part of many vital functions of the human body, from immunity to brain activity. They consist of a chain of smaller molecules, called *amino acids* or *residues*. There are 20 different amino acids that can exist within a protein. Each amino acid has the same core structure, which consists of a central carbon atom (CA) bonded to an amino group (NH2) a carboxyl group (COOH), and a hydrogen atom (HA). This core structure is known as the *backbone* of an amino acid. Aside from the backbone, the residues have a group of atoms that make each type of amino acid unique. This group of atoms is attached to the CA atom and is known as the *side chain*. Amino acids chained within a protein are linked together by *peptide bonds* which are formed when the carboxyl group of one amino acid combines with the amino group of a second. The backbones of the amino acids in combination with the peptide bonds form the backbone of the protein.

Proteins are described by different levels of structure. The most simple level is the primary structure, which refers to the linear sequence of amino acids that make up the protein. The secondary structure of a protein refers to the structural patterns that combinations of amino acids often conform to. For example, groups of amino acids often form secondary structure patterns such as alpha helices and beta sheets. Finally, the tertiary structure of a protein is the three-dimensional conformation of the atoms that make up the molecule. Essentially, the tertiary structure of a protein describes how its different amino acids *fold* into one, complex three-dimensional structure. Figure 3.1 visualises the different structural levels of a protein.

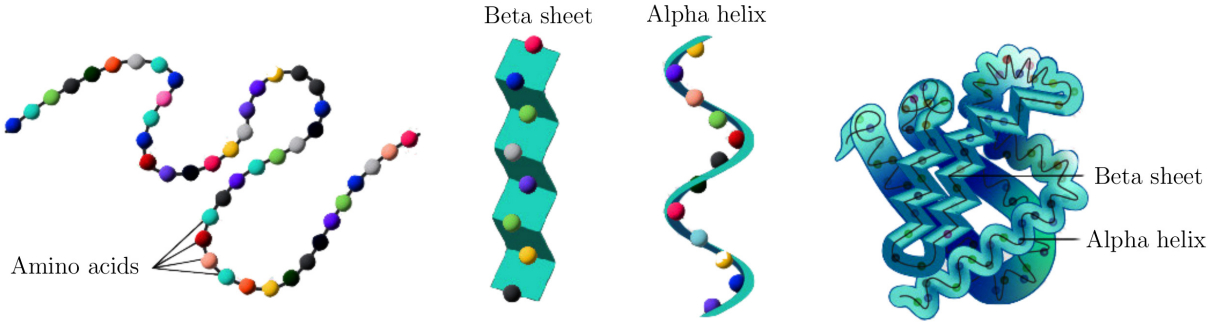



Figure 3.1 – Three different levels of protein organization. Left: primary structure (linear sequence of amino acids). Middle: secondary structure (structural patterns, alpha helix, and beta sheets). Right: tertiary structure (three-dimensional confirmation of proteins). Image modified of original provided by National Human Genome Research Institute [1].

1. `https://www.genome.gov/genetics-glossary/Protein`.

If we have an unidentified protein, different techniques may be used to determine its primary structure, such as Edman degradation [136–138] and mass spectrometry [139–141]. Once we know its primary structure, we have identified the protein. However, to know its function we need to predict, compute, or measure its tertiary structure. This is known as *protein structure determination*. A protein *in vivo* will tend to fold in such a way that its total energy is minimized. The potential energy of tertiary protein structure $x$ can for example be estimated using a *force field* $f$, which are computational models that estimate forces between atoms of a protein:

$$f : \mathbb{R}^{Kn} \longrightarrow \mathbb{R}$$

An example force-field function is AMBER [142], which relies on the chemical composition of the protein structure $x$ and force field parameters (such as $r_{\text{eq}}$, $\theta_{\text{eq}}$, $K_r$ and $K_\theta$) to compute the global energy $E_{\text{total}}$:

$$E_{\text{total}} = \sum_{\text{bonds}} K_r(r - r_{\text{eq}})^2 + \sum_{\text{angles}} K_\theta(\theta - \theta_{\text{eq}})^2 + \sum_{\text{dihedrals}} \frac{V_n}{2}[1 + \cos(n\phi - \gamma)] + \sum_{i<j} \frac{A_{ij}}{R_{ij}^{12}} - \frac{B_{ij}}{R_{ij}^{6}} + \frac{q_i q_j}{\epsilon R_{ij}}$$

More information about this function is found [142], while these types of angles and other force-field parameters used in this function are discussed in Section 3.3. Therefore, protein structure determination (generally in the context of computer simulation) is regarded as a minimization problem [143, 144]:

**Definition 7** *Given the primary structure $P$ of a protein, determine a tertiary structure $x$ that respects the chemical composition implied by $P$ such that a force field function $f(x)$ is minimized.*

There are different approaches that one can take to try to compute or predict protein structures, these include experimental techniques, machine learning methods, and several distance-based approaches. Some of these methods, in particular the methods that we will use in the experiments in this chapter do not rely on energy terms but will instead define the protein folding problem as a version of the DGP or DDGP. In the next sections we will explore the various methods that one can use to determine protein structures.

## 3.2 Experimental techniques and machine learning

There are various (non-distance-based) approaches to protein structure determination. In this section, we will look at two popular and effective groups: techniques relying on physically analysing the protein and methods that exploit large datasets of known structures using machine learning, mostly by employing neural networks. These approaches are connected because the known structures that make up the datasets that are used for training the neural networks have often been measured using one of the experimental techniques.

### 3.2.1 X-ray crystallography

Most known protein structures have been measured using X-ray crystallography [145–148]. The first step in this process is to purify and the protein and then crystallize it. Obtaining this crystal is not an easy step, as it requires conditions that promote the formation of a regular, repeating crystal lattice. Different techniques are used to grow crystals, such as vapor diffusion [149] and (micro) batch crystallization [150]. Once the crystal has been grown the next step is to mount it on a goniometer inside an X-ray diffractometer. Then, the crystallized protein is exposed to a series of X-ray beams. The crystal causes the incident rays to diffract in many directions onto a screen. These directions, angles, and the intensities of the diffracted beams are then analyzed which allows for the determination of the distribution of the electrons in the protein. Techniques such as Fourier analysis and Patterson synthesis are employed in this step. The resulting map of the electron density can in turn be used to deduce the position of each atom in the crystallized protein, giving us the three-dimensional structure. Once the three-dimensional structure is obtained, it undergoes validation to ensure accuracy and reliability, generally this involves checking its geometric feasibility.

The structures measured with X-ray crystallography are generally of very high quality, but can lack precision, because using the X-ray beams causes a small portion of radiation damage to the proteins. Aside from this, there are several other limitations. Firstly, this method works well only on proteins that have rigid structures that form nice, ordered crystals. It fails to measure the structure of flexible proteins because X-ray crystallography relies on having many molecules aligned in the same orientation. The portions of a protein that do not follow this pattern will be invisible in the electron maps which in turn leads to missing coordinates [151]. Another shortcoming is that the structure that the protein

forms inside the crystal is not necessarily the natural confirmation that it has in other environments.

### 3.2.2 Cryo-Electron Microscopy

A more recent technique avoids the need for crystallization and X-ray beams. This eliminates the issue of the X-ray beams causing damage to the protein sample. CryoEM [152–154], short for Cryo-Electron Microscopy, relies on cooling the molecules down to cryogenic temperatures, preserving them in a thin layer of non-crystalline ice. To do this, the proteins are applied to a grid covered with a thin layer of support material. An example of a support material that is often used is carbon [155]. The excess solution is blotted away, leaving a thin film of sample across the holes of the grid. The grid is then rapidly plunged into a cryogen (such as liquid ethane or propane) to freeze the sample [156]. Once the sample is frozen, the grid is transferred to a cryo-electron microscope. The microscope operates under vacuum conditions to minimize electron scattering. A beam of electrons is transmitted through the frozen protein, and the interactions of electrons with the sample produce an image. Several such images are taken from different orientations of the sample, resulting in a collection of 2D projection images. The collected 2D images are processed using computational methods to reconstruct a 3D model of the specimen. This process, known as single-particle reconstruction [157], involves aligning and averaging the 2D images to improve signal-to-noise ratio and resolve the 3D structure. Advanced image processing techniques, such as classification and refinement algorithms [158], are employed to sort and refine the images to obtain a high-resolution 3D reconstruction.

The CryoEM technique is more recent and is able to rival [159] and in some cases even surpass [160] the quality of the structures found by X-ray crystallography. CryoEM is still being extensively researched and improved. Despite these advantages, there are several drawbacks [161]. It suffers the same limitation as X-ray crystallography when it comes to flexible proteins: we can only measure to protein structure found in the molecule's frozen state. Furthermore, X-ray crystallography is a much faster technique than CryoEM and thus has a higher throughput. Work is also being done to combine the results of CryoEM and X-ray crystallography in one model. This process is known as *integrative modeling* [162]. Both methods give us access to high-resolution protein structures, albeit structures of the protein in a specific, fixed state. We will see that when we use specific distance-based approaches we can overcome this issue. Applying these experimental techniques is an expensive process requiring large machines and a

lot of power. Therefore, it is desirable to have different methods, which rely only on computation. One promising avenue of approach is based on machine learning.

### 3.2.3 Nuclear Magnetic Resonance Spectroscopy

The above two methods are effective for measuring the structure of a protein. However, one main shortcoming is that before the measurements are made, the protein is fixed in space in a way where, either by freezing or by capturing it in a crystal. This means that the structures that we measure using X-ray crystallography or CryoEM do not necessarily correspond to the way a protein will behave in vivo. Moreover, these methods are not well suited for flexible proteins. When we use solution Nuclear Magnetic Resonance (NMR) spectroscopy [2, 163], a protein is dissolved in a solvent (often water), so that it is free to rotate and tumble naturally. NMR spectroscopy relies on the fundamental quantum mechanical property of nuclear spins. The protein (in solution) is placed in a strong and homogeneous magnetic field. In this magnetic field, atoms with nonzero spin numbers are at different energy levels. When a radio-frequency pulse is applied to the sample at a specific frequency, it matches the energy difference between the spin states. Some of the atoms absorb energy and transition between energy states. The amount by which the energy transitions is called the *chemical shift* of the atom. After this process, the resonant frequency of the nuclei of the atoms can be measured, which is known as *resonance*. Using a Fourier transform we can convert these resonances and chemical shifts to a spectrum. Such spectra allow for the assignment of each peak in the spectrum to a specific atom within the protein. In Section 3.2.3 we see an example of such a spectrum, created using the NOE [164]. Such a spectrum is referred to as a NOESY spectrum, and it allows it to identify through-space interactions within the molecule. The Nuclear Overhauser effect is what happens when two atoms of the same type with similar nuclei (e.g. $^{1}$H, $^{13}$C and $^{15}$N) are close together and engage in *cross-relaxation*, which means that energy is being transferred between their ions. This is visible in the spectrum because, at the intersection of the chemical shifts of the atom pairs, we will see *cross-peaks*, indicating that the offending atoms must be close together in the molecule [165]. This type of spectrum is of great interest to us because it allows us to identify an interval distance between light-weight atoms (such as hydrogen atoms). Aside from this, the chemical shifts in the two-dimensional $^{13}$C and $^{15}$N spectra give us information about the $\phi$ and $\psi$ dihedral angles in the backbone of the proteins. More details about these distance restraints and torsion angles can be found in Section 3.3. The restraints obtained from the NMR experimental

data can then be exploited to generate protein structures. The standard method of doing this is by using the meta-heuristic Simulated Annealing [166, 167], which is discussed in Section 3.4.1. However, as we will see in Sections 3.4.2 and 3.4.3, these distances can be exploited by DG methods as well.

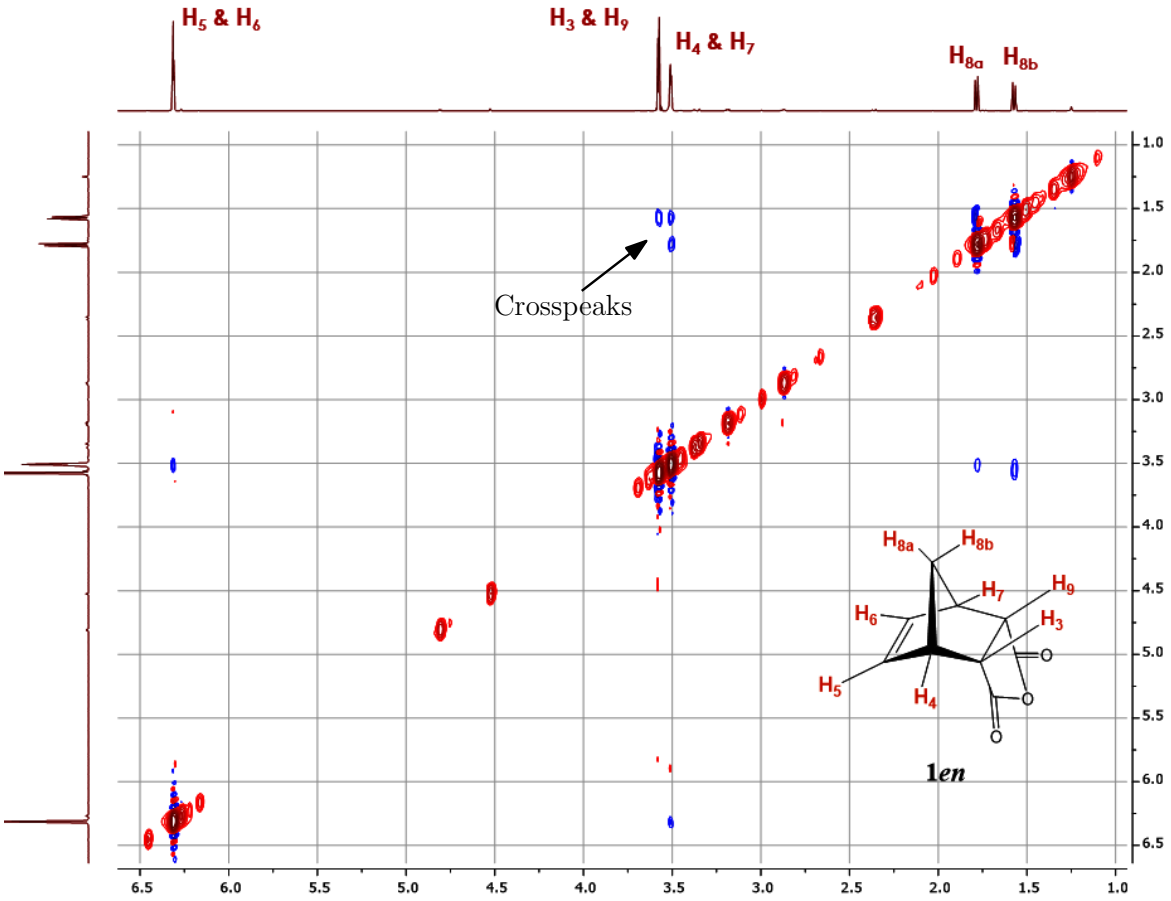



Figure 3.2 – An example of a two-dimensional NOESY spectrum for a small molecule, where the chemical shifts on the axes correspond to $^1$H protons (labeled in ppm). At the top of the spectrum, the different frequencies are labeled with the hydrogen that emits it. In the bottom right there is a two-dimensional diagram of the molecule that is being analysed. The red peaks in the diagonal are the self-peaks for each of the hydrogen atoms. The cross-peaks (blue) away from the diagonal are the ones we are interested in. For example, the indicated cross-peaks in the figure tell us that both $H_8$ atoms are in close proximity to the $H_4$ and $H_7$ atoms in the molecule. We see that there is a peak for $H_{8b}$ and the $H_3$,$H_9$ pair, but no peak between the pair and $H_{8b}$, possibly indicating that $H_{8b}$ is not close to $H_3$ and $H_9$. Figure taken from [168] and modified.

### 3.2.4 Machine learning and AlphaFold

In some cases, when experimental techniques are expensive or otherwise unpractical, other approaches must be used. Some of the most successful methods for protein structure *prediction* are knowledge-based. Knowledge-based methods involve learning and extracting knowledge from existing solved protein structures (using the methods from the previous section) and generalizing this knowledge to unknown protein structures. Note that we use the word prediction instead of determination because we have no guarantee of the correctness of the output. *Machine learning* methods automatize this process of knowledge extraction. At the start, a dataset of known structures is partitioned into two parts: a training set and a test set. The training set is used to train the machine learning model so that it becomes capable of recognizing patterns between the primary structure of the proteins and their tertiary structure. Once it has been trained sufficiently, it can now be used to predict the tertiary structure of the proteins in the test set. Afterward, the accuracy of the model is measured by comparing the computed structures to the known structures in the test set. Once we have identified a model with high enough accuracy, we can start to use it on proteins for which the structures are unknown.

The type of machine learning that has gained the most popularity and has attained the best results is *deep learning* [107] using *neural networks*. Recently, AlphaFold [169] by DeepMind has produced structures with very high accuracy. The process of using deep learning for protein structure prediction generally consists of three steps [170]: (1) an input module, which converts the primary structure of a protein into different features, (2) a neural network that transforms these features to spatial information, and finally (3) an output model that uses to spatial information to compute and refine the three-dimensional conformation of the protein. The resulting structure may be further refined in an additional step if necessary. For each of these steps, different approaches have been considered.

1. The features extracted from the primary structures almost always consist of Multiple Sequence Alignments (MSAs) between homologous proteins [171]. Such alignments can be used directly as raw features (AlphaFold) or they can be converted into different features [172–175].

2. The choice of neural network in the second step is of vital importance. The first neural networks used were shallow convolutional neural networks (CNNs). These are a type of neural networks that work very well in the context of computer vision [107].

The motivation for using CNNs was that the features obtained from the MSAs resemble an image. Modest accuracies were achieved using CNNs [172, 174]. This is because CNNs have several shortcomings when it comes to protein structure prediction. For example, CNNs assume translational invariance as to where patterns appear in the input. This is true for general images, but in the case of proteins, the position in which a pattern occurs in an amino acid sequence is very relevant to its final structure. Next, CNNs were enriched with deep residual networks [173, 175], booking some great improvements. Finally, AlphaFold introduced the use of the attention networks, which was a major breakthrough [169]. Such attention networks are powerful architectures that are capable of selectively focusing on certain parts of the input data that are relevant to the task at hand. Instead of assuming that local patterns are most important, these networks initially place equal weights on all possible interactions, and then later refine these weights during the training phase.

3. As output, the first deep learning models produced binary contact maps [172, 173], based on the distances between amino acids. These binary contact maps must then be converted into folded proteins by using protein modeling software such as Rosetta [133]. Other models improved on this by using discretized distances (using small increments) instead of binary contacts [175, 176]. This improved the quality of the predicted structures. However, an extra step was still necessary to transform these amino acid distance maps into folded protein structures. The final version of AlphaFold improves on these approaches by being able to directly output the three-dimensional structures, based on a method proposed in [177].

## 3.3 Inter-atomic distances and torsion angles

Before we discuss the distance-based methods, we will explore the different types of inter-atomic distance restraints and dihedral angles that we can deduce from the primary structure of a protein. As mentioned before, not only the primary structure but also the spectra obtained from NMR let us derive information about distances and torsion angles. Some of these distance restraints are precise values, while others give us only a lower and upper bound on the distance. As mentioned in Section 1.3, these torsion angles may be converted to distances and included in the DGP instance, so that general DGP methods may use them. However, as we saw in Section 2.5 if we are using branch-and-prune, we may directly exploit the signs of the torsion angles during the branching phase. As part

of this thesis, Java code was written that derives all these distance and torsion angle types directly from STAR-NMR files (see Section 3.5.1). In the Java package the code for parsing STAR-NMR files can be found in the NMR class (see Figure 1.4). The code that represents the protein structure determination problem as a DGP is found in the MDGP class. We will now explore the different distance types and torsion angles.

**Type 1: van der Waals radii**

The van der Waals radius of an atom $u \in V$ is the radius of an imaginary sphere which represents the closest any other atom $v$ may approach $u$ [178]. This means that the sum of these radii can be used to provide an expected lower bound to the distance between any pair of atoms. The van der Waals radius is specific to the type of atom. This radius allows us to describe the electron cloud around the nucleus of an atom as a sphere. However, due to polarisability, these clouds are never *real* spheres, such that a pair of atoms can actually be closer than the sum of their van der Waals radii. In other words, the spheres in the van der Waals model can be considered as *soft* spheres, meaning that they can slightly penetrate one another, which is why generally only a portion of this sum is used as the lower bound on the distances. On the one hand, van der Waals distances are not very informative because they only carry a lower bound, while their upper bound is generally very large (initially set to infinity, it can be improved by using triangular inequalities involving related distances). On the other hand, these distances are very abundant because they can be defined for every pair of atoms that are not bonded. Furthermore, if these restraints are respected, they make sure that clashes are avoided between atoms within the computed molecular structures. In the experiments conducted as part of this thesis, not all van der Waals distances are included in the instances. This is because only a subset is necessary to avoid clashes, and a very large number of distances will greatly slow down the refinement step in MDjeep.

**Type 2: covalent geometry, force fields, and torsion angles**

These distances and torsion angles are based on the *covalent geometry* of the molecules, which refers to the way the different atoms are bonded together. Because we know the primary structure of the protein, we know precisely which atoms are bonded together in which amino acid. This means that we can use information from *force fields* to derive distances and torsion angles. Alternatively, this information can also be extracted from protein structures in the Protein Data Bank (PDB), if the structure of the protein at hand

was already measured or computed by a different technique (when generating artificial instances, see Section 3.5.2). As mentioned in Section 3.1, force field models define a function for the potential energy of a molecule, where the parameters of this function encapsulate information about the forces between atoms. One such parameter that is of interest to us is the bond lengths. For every pair of bonded atoms $(u, v)$ we may derive an exact distance between $u$ and $v$, which differs based on the types of atoms involved as well as where they are located within the amino acid. A second parameter of interest is the bond angles. For every pair of atoms $(u, w)$ that are bonded to a common atom $v$, this parameter describes the angle that the three atoms form. This, in combination with the two bond lengths that we know, allows us to compute the diagonal distance between $u$ and $w$. This bond angle distance is regarded as exact, even though there may be small variations in the covalent geometry of different proteins. Section 3.5.2 gives more detail on this issue of variance. Examples of force fields that one can use are AMBER [142], CHARMM [179], and PARALLHDG [180]. Different force fields may have slightly different parameters. In the Java package described in Section 1.5, the FORCEFIELD class is responsible for deriving these distances, and supports the AMBER and CHARMM forcefields. Apart from bond lengths and bond angles, the primary structure of the protein also lets us deduce values on specific torsion angles. Firstly, we have information about a torsion angle $\omega$ in the backbone of the protein. This is a backbone torsion angle that is formed by a sequence of four atoms between two consecutive amino acids. For $\omega$, the sequence of four atoms consists of the CA and C of the first amino acid and the N and CA of the next amino acid, describing the rotation around the peptide bond between C and N. The angle $\omega$ is almost always close to 180°, except when the peptide bond is *cis*, in which case it is close to 0°. In experiments from the literature, the torsion angles $\omega$ are often fixed directly at 180°, even though in practice there can be quite large variations (see Section 3.5.2). Another type of torsion angles, known as *improper* torsion angles, can taken from force field parameters. These torsion angles regard sequences of atoms $(v_1, v_2, v_3, v_4)$, where we already know the distance $\delta(v_1, v_4)$ from a bond length or bond angle. This means that for general DGP methods, they do not give any extra information. However, exploiting the sign of these torsion angles is very valuable for DDGP methods, as they do give a lot of information about the structure of the proteins at hand (see [181]). For example, the improper torsion angle created by the atom sequence N, CA, C, and HA (all in the same amino acid) tells us whether the amino acid is left- or right-handed. They are included in the instances used in the experiments in this thesis and exploited by Java MDJEEP.

**Type 3: NMR restraints from NOESY spectra**

As described before, the NMR spectra obtained from NMR experiments can be used to derive distance restraints. The spectrum that is relevant for this distance type is the $^{1}$H two-dimensional NOESY spectrum, for which an example is shown in Section 3.2.3. This is a spectrum in which each peak corresponds to a chemical shift of a hydrogen atom (or to noise). Once a cross-peak is identified in such a spectrum, we know that the distance between the two hydrogen atoms that correspond to the peaks should be between 1.8 and 5Å [182]. This lets us enrich the distance graph $G$ with several such interval distances between hydrogen atoms that are relatively close together.

**Type 4: Distances from $\phi$ and $\psi$ backbone torsion angles**

Aside from the distances between hydrogen atoms discussed in the previous paragraph, the NMR spectra can offer us more. The chemical shifts in the two-dimensional $^{13}$C and $^{15}$N spectra give us information about the $\phi$ and $\psi$ dihedral angles in the backbone of the proteins. In the protein backbone, just like the $\omega$ angle mentioned before, these angles are formed between two consecutive amino acids. The $\phi$ angle corresponds to the dihedral angle formed by the atoms N, CA, and C of the first amino acid and the N of the next amino acid, describing the rotation around the CA - C bond. The $\psi$ angle corresponds to the dihedral angle formed by the atom C of the previous amino acid and the atoms N, CA, and C of the next amino acid, describing the rotation around the N - CA bond. The chemical shifts from the NMR spectra, in combination with machine learning tools such as TALOS+ [183], let us predict the secondary structure of the proteins with high accuracy. Knowing the secondary structure allows for the derivation of bounds on the backbone dihedral angles $\phi$ and $\psi$. The values for the torsion angles can then be exploited in the same way as the $\omega$ dihedral angle.

## 3.4 Distance-based approaches

In this section, we will discuss how we can exploit inter-atomic distances (and torsion angles) to compute protein structures. First we will look at the standard method that is used for generating structures from NMR experimental data: Simulated Annealing. SA works by solving the optimization problem given in Definition 7, while observing the distance and angular restraints given by the NMR data. Next, we will look at methods

that regard the protein structure determination problem as an instance of the Distance Geometry Problem where the dimension $K = 3$. In this case, the vertices $V$ of the graph $G = (V, E)$ are the atoms of the protein and the edges $(u, v) \in E$ together with their weight $\delta(u, v)$ describe inter-atomic distances. This variant of the DGP is often referred to as the Molecular Distance Geometry Problem (MDGP) in the literature [3, 8, 10–12, 22, 33, 94, 95, 97, 102]. Note that the distance graphs for these DGP instances, especially those with interval distances, will likely have many possible embeddings. Many DGP tools will find only one such embedding or several possible embeddings. That is why solving the DGP instances in the context of protein structure determination is often referred to as *conformational sampling* because the methods only sample some of the many possible conformations. Different such methods from the literature are reviewed in Section 3.4.2. Another class of algorithms, present within the branch-and-prune framework, is capable of enumerating all possible conformations. This method is discussed in Section 3.4.3, and is the approach that we used in this thesis for generating protein structures.

### 3.4.1 Simulated Annealing

As mentioned in Section 3.1, a protein in vivo will tend to fold in such a way that its total energy is minimized. Therefore, the goal becomes to compute a protein structure that satisfies the identified distance constraints while minimizing the global energy of the molecule. This optimization problem is not straightforward because one must avoid getting stuck in a local minimum. Simulated Annealing (SA) [166, 167] is used to overcome exactly that. SA is a constrained meta-heuristic search that attempts to approximate a global optimum. It was inspired by annealing in metallurgy, where a metal is heated to a high temperature quickly and then slowly cooled, which increases its strength while making it easier to work with. SA attempts to mimic this as a global optimization approach with a high-temperature phase and a cooling phase.

The SA process starts by generating an initial solution, which will likely have a high global energy. In the context of protein structure determination, different ways of producing initial solutions have been proposed, generally based on random generation [184]. In this initial phase we also initialize a temperature control parameter to a high value. Then we will attempt to minimize the energy of our solutions while gradually lowering the temperature as we attempt to reach a state of minimum energy. At each iteration, a new solution is generated and its energy is measured. In the classic SA approach, this new solution is generated randomly. However, in the context of protein structure deter-

mination, other strategies are employed more often used. For example, in [184], a genetic algorithms based method is proposed to choose a neighboring solution. Once we have the next solution, we compute its global energy. If the new energy is lower than the current energy we accept this new solution as better. However, to avoid getting stuck in a local optimum, sometimes a solution with a higher energy is accepted as well. This is done using a specific probability which depends on the temperature control parameter. When the temperature is high, we accept worse solutions (in terms of energy) with a higher probability. This way, as the temperature control parameter declines, we converge towards an (approximate) global minimum.

Several computer programs have been developed based on the above algorithm. Examples of software tools that are used widely in practice are XPLOR-NIH [185], CYANA [186] and GeNMR [187]. XPLOR-NIH is a powerful tool, which can directly take NOESY spectra as input, where it can automatically detect crosspeaks, avoiding peaks arising from noise. It can be used to read input data from X-ray crystallography as well. The software package CYANA also works directly on NOESY spectra [188]. GeNMR was developed to offer a tool that works as a web service. It uses a combination of XPLOR-NIH and Rosetta [133] to generate structures at the back end.

All of these tools provide very good results when tested on real data [189]. These software tools allow for the integration of X-ray crystallography data as well as in some cases CryoEm measurements. Combining these experimental techniques with NMR measurements leads to integrative models that can predict protein structures with very high atomic resolution. Currently, these energy minimization tools are the best approach for structure determination by NMR if one is interested in sampling the conformational space of a protein. However, there is a drawback to these methods. While they generate a set of possible structures with low potential energy, there is no guarantee that these are all the possible low-energy states the protein could take (because of the use of a meta-heuristic). Furthermore, proteins can sometimes move from one state to another and shortly be at a point where their potential energy is not minimal. To model these dynamics other approaches may need to be used.

### 3.4.2 General DGP methods

Several general DGP, discussed in Section 1.4, have been tested in the context of protein structure determination and conformational sampling. However, none of these methods have been tested on distance constraints derived from real NMR experiments.

Instead, these algorithms were tested on artificially generated instances. Some of these instances were randomly generated, and tend to follow a general protein-like structure. For example, several of the meta-heuristics VNS and Multi-start as well as the global optimization method SBB were tested on small random instances, providing good results [94]. However, these instances only had exact distances and do not closely resemble DGP instances derived from NMR experiments at all.

In other experiments, known protein conformations are taken from the Protein Data Bank (PDB) [190], which are structures that are generally measured using one of the experimental techniques that were discussed before. Next, distances are generated following a set of rules, which generally aim to mimic instances from real NMR data. The reason for using these artificial instances is because real NMR data contains a large degree of uncertainty and noise, which cannot be handled by these general DGP methods. The graph decomposition algorithm ABBIE was tested on such instances, but only exact distances were considered [90]. The Build-Up algorithm was tested on similar exact instances with great results [119]. The simple heuristic Stochastic Proximity Embedding (SPE) was shown to work well on artificial instances where both intervals and exact distances were included, providing results that are competitive with other popular conformational sampling methods [191]. GNOMAD was also tested on such instances. In these instances, only distances between "close" atoms were included, which are all distances lower than 10Å [98]. Yet another step closer are test instances with interval and exact distances only between atoms in consecutive amino acids. Both the smoothing-based algorithms, DGSOL and DCA have been tested on such instances with good results [44, 95, 96]. In [103], Monotonic Basin Hopping (MBH) was tested on similar instances and was shown to outperform DGSOL in most cases. The EMBED algorithm was tested on instances with exact and interval data as well, where all pairwise distances smaller than 4Å were included. The resulting structures were of good quality. The Alternating Projections Algorithm was tested on artificial instances that are very close to real data. Here, each of the distance types was aptly simulated. The algorithm was tested on a specific protein (bovine pancreatic trypsin inhibitor) and was shown to work well [84].

General DGP methods can work well for artificially generated instances, including instances with large proteins. However, some of these algorithms work only on instances with exact distances and are thus not useful in the context of instances derived from NMR spectra. Others can work on instances with intervals, but the bounds that were used in the experiments were smaller than they should be and many more distances were included

compared to what is present in real data. It is hard to compare these algorithms fairly, as they may work well for specific instances and not so well for others. In the next section, we will look at methods that exploit the discretizability of the instances related to protein structure determination.

### 3.4.3 Exploiting the discretizability of the instances

The distance types described in Section 3.3 allow for the discretization of the DGP instances related to protein structure determination. In particular, for this, we require the information about bond lengths, bond angles, the $\omega$ angle (**type 2**) and the $\phi$, and $\psi$ torsion angles (**type 4**). These distances and dihedral angles are enough to satisfy the assumptions given in Definition 6. The other geometric information may be used as pruning devices. Different branch-and-prune methods have been tested on instances related to protein structure determination [7, 9–14]. The advantage of this approach is that BP theoretically will allow us to identify all protein structures that satisfy the geometric information captured in the DDGP instances. For all the papers cited above, just like for the general DGP methods, the DDGP instances were artificially generated. As mentioned in the previous section, one cause of this uncertainty is noise in the NMR data, which will be further investigated in Section 3.5, where we present experiments with genuine NMR instances. Secondly, the BP methods regard distances from the covalent geometry as exact, while in practice they may vary a lot. More information about this is given in Section 3.5.2, where we experimentally study the effect of the variations in the covalent geometry.

Initial implementations of branch-and-prune, in 2009, had trouble with small amounts of noise and required the input distances to be of very high precision [192]. This was improved on in the same year [10] by including a tolerance parameter for the pruning step. Test instances were generated by including all distances smaller than 6Å. Different amounts of noise were then introduced into a subset of these distances, creating interval distances. The results showed clear correlations between the amount of noise (the range of the intervals), the number of found solutions, and the quality of the found solutions. Next, in 2013, experiments were done with instances generated with all distances shorter than 5Å and a small amount of noise (0.3Å) [11].

In 2010, MDjeep was first released [7] and was tested on generated instances with only exact distances. Later releases of MDjeep [9, 13, 14] could also handle interval data. In particular, in [9], experiments were done with artificial instances that are getting closer

to genuine NMR instances. The instances were generated with special rules for some of the distance types discussed in Section 3.3. For **type 2**, the bond length and bond angle distances, exact values were used with three decimals. For **type 3**, interval distances with a range of 0.5Å are used. It is important to note here that for these distances from NMR restraints, only distances between hydrogen atoms closer than 5Å were kept. This is a clear improvement as compared to previous experiments, where all distances smaller than 5Å are included. Lastly, for **type 4**, intervals with range 0.1Å were created. The methods were tested on some small proteins and peptides, showing good results. However, note that while these experiments got closer to genuine data, the ranges on the intervals are still quite small compared to what they are in real NMR data. In the next section, we will present results where MDjeep was tested on instances created from genuine NMR data.

## 3.5 Experiments and results

In this section, we will discuss the several experiments conducted throughout this thesis, where we used branch-and-prune methods to compute protein structures.

### 3.5.1 Branch-and-prune with real NMR instances

As mentioned in the previous section, the BP implementations are often only tested on artificially generated data, simulated in a way to resemble real NMR distance data (see for example [14]). This motivated a study on the different distance types, published and presented at the Computational Structural Bioinformatics Workshop, which is a satellite workshop of the IEEE Bioinformatics and Biomedicine conference BIBM [27]. In this study, different experiments were conducted using real data as well as a mix of real and artificial distance data. The goal was to see which type of distances have the largest effect on the quality of the embeddings as well as the feasibility of the BP algorithm. The structures were computed using the C-version of MDjeep. In this section, we will focus on the experiments with the instances from real NMR data. Furthermore, the structures that we will discuss here are computed with the Java implementation of MDjeep, which exploits the signs of the dihedral angles (see Table 2.1 for the differences). In all experiments with proteins presented in this thesis, we will only consider atoms in the backbone of the proteins. The atom names of the atoms considered for each amino acid are: N, CA, C, O, OXT, H1, H2, H and HA. We only consider the backbone atoms, because once

we have the backbone, placing the side chains becomes a straightforward post-processing step. This final step can be done using protein analysis software such as ROSETTA [133].

In order to be able to handle real NMR data, the Java package created for this thesis is capable of reading a STAR-NMR file. STAR-NMR is an extension of the more general STAR format, which is a text-based file format for storing structured data. STAR-NMR files describe distance constraints and torsion angle constraints that result from the spectra obtained from NMR experiments. Furthermore, parsers were added for two different force fields: AMBER [142] and CHARMM [179]. The NMR data, combined with force field parameters and the primary structure, lets the package generate $\text{DDGP}_3$ instances (Definition 6) from real data. Even when there is no NMR data available, or the distance information is sparse, we can still generate discretizable instances. The NMR restraints (**type 3**) are only used for pruning so are not required for discretiztaion. The NMR torsion (**type 4**) angles are used for discretization, but when they are missing they can be replaced by full 360° intervals (more about this later in this section). This instance generation tool was extracted from the Java package and can be found at a public GitHub repository [2]. This tool does not only generate instances from NMR data, but also from X-ray crystallography data (see Section 3.5.2). Once we have the instances generated, we can proceed with branch-and-prune to generate solutions (which are saved as PDB files).

Experiments were conducted on seven small proteins and peptides from the Protein Data Bank [190]: `2jmy`, `1vm2`, `2jp8`, `2jta`, `6nm2`, `6nm3` and `2fbu`. The PDB contains known structures of these proteins, which were generated by NMR in combination with one of the simulated annealing methods described in Section 3.2.3. For example, the known structure for the peptide `2jp8` was created with the software XPLOR-NIH [185]. Table 3.1 shows the number of atoms ($|V|$) and the number of inter-atomic distances available in the generated DDGP instances, grouped by distance type (which sum to $|E|$). Note that the values in this table differ from the ones reported in Table 1 in [27]. This is because, for these experiments, we only consider the backbone, while in [27] the side chains were also considered. The table shows the different properties of the proteins. First, we see the size of the proteins in terms of the number of atoms as well as the number of amino acids. Next, the different number of distances are listed. Only a subset of the van der Waals distances are included, only those between a select number of C atoms. This is because including too many distances slows down the refinement step, and we do not need to consider all type 1 distances in order to avoid clashes.

2. `https://github.com/simonheng/BP_ProteinFileReader`

| **Properties** | 2JMY | 1VM2 | 2JP8 | 2JTA | 6NM2 | 6NM3 | 2FBU |
|---|---|---|---|---|---|---|---|
| Number of amino acids | 15 | 13 | 7 | 10 | 8 | 8 | 12 |
| Number of atoms | 91 | 78 | 43 | 61 | 48 | 48 | 73 |
| Type 1 (van der Waals radii) | 91 | 66 | 21 | 36 | 21 | 21 | 55 |
| Type 2 (Bonds/bond angles) | 350 | 316 | 171 | 61 | 191 | 190 | 296 |
| Type 3 (NMR restraints) | 59 | 37 | 10 | 12 | 16 | 8 | 14 |
| Type 4 (NMR torsion) | 20 | 10 | 0 | 0 | 5 | 6 | 9 |
| Type 3 Average range (Å) | 2.07 | 2.42 | 1.72 | 3.80 | 2.18 | 2.60 | 1.67 |
| Type 4 Average range (°) | 68.2 | 80.0 | - | - | 72 | 113.7 | 65.55 |

Table 3.1 – Different properties of the proteins and peptides considered for these experiments. Includes the number of distances, per type, that can form our DDGP instances.

For the first peptide, 2JMY, we see that there are 15 amino acids. This means that there are 14 $\phi$ and 14 $\psi$ backbone dihedral angles for this protein. However, in the table, we see that the NMR chemical shift prediction only gave us information about 20 out of 28 torsion angles. This is a recurring phenomenon for these peptides. For the two peptides 2JP8 and 2JTA we have no dihedral angle information whatsoever. This is logical because these structures have no clear secondary structure present, and thus TALOS+ is not able to predict these backbone dihedral angles. To still discretize despite missing this information, for each of the absent dihedral angles we include an interval from $-180°$ to $180°$, covering the full range and both signs. Missing backbone torsion angles is one part of the uncertainty present in these instances. Aside from this lack of information, we have very large degrees of noise, also visible in the information presented in the table. The intervals on the NMR restraints (**type 3**) can sometimes be quite large. Even more, the backbone torsion angles predicted with TALOS+ [183] tend to have average ranges of

65° up to even 113°. The uncertainty in these backbone torsion angles means that the search space of the tree will be quite large.

For this experiment, we used the force field CHARMM and ran Java MDjeep with tolerance parameter $\epsilon = 0.01$Å and the maximum number of solutions was capped at 5000. The resolution parameter $\rho$ was set to 2.0Å. When we decrease $\rho$, we cut the sub-arcs more often, thus increasing the chances of finding even better structures, at the expense of increasing the size of the tree. Recall that the effect of the resolution parameter $\rho$ is twofold, not only does it decide the length of the sub-arcs we create when we branch, but it is also used to decide when two consecutive solutions are too close to each other. In fact, because the uncertainty in the instances causes a very large search space, many of these solutions are pruned using this resolution parameter. The value of 2.0Å was chosen for these proteins after preliminary tests because it gives a good balance between the quality of the best solution found and the total size of the tree. To measure the quality of the accepted solutions, we compute the Root mean square deviation (RMSD) between the computed structures and the known structures, after alignment. This straightforward metric is computed for two realizations $x$ and $y$ as follows:

$$\mathrm{RMSD}(x, y) = \sqrt{\frac{1}{|V|} \sum_{v \in V} ||x_v - y_v||^2} \,, \tag{3.1}$$

where $||\cdot||$ is the Euclidean distance norm. For aligning we used a Kabsch alignment algorithm [193]. Our implementation also attempts, together with translations and rotations, to perform a total reflection of the protein models to improve the alignments. The results of the experiments are summarized in Table 3.2.

| **Results** | 2JMY | 1VM2 | 2JP8 | 2JTA | 6NM2 | 6NM3 | 2FBU |
|---|---|---|---|---|---|---|---|
| Best RMSD | 1.63 | 2.02 | 1.55 | 2.00 | 1.66 | 1.49 | 2.52 |
| Worst RMSD | 3.61 | 6.88 | 4.67 | 5.98 | 6.46 | 5.06 | 5.11 |
| Mean RMSD | 3.0 | 4.43 | 2.89 | 4.02 | 2.78 | 3.47 | 4.06 |

Table 3.2 – The resulting RMSD scores for the proteins (in Å).

Firstly, we see that the best found RMSD scores of the computed structures for each of the proteins are quite low. This means that some of the structures that we compute are close to the structures that are in the PDB, which is a very good sign. Secondly, we can

note that the differences between the worst and best RMSD scores are quite large. This is no surprise, given the large amount of uncertainty in the instances. This uncertainty leads to a very large search space, which means there are many possible solutions. Furthermore, each of these possible solutions satisfy all the distance and angle restraints at hand, given the tolerance $\epsilon$, which means they are all valid solutions for the DGP at hand. We will now take a closer look at the computed structures with the best (lowest) RMSD scores.

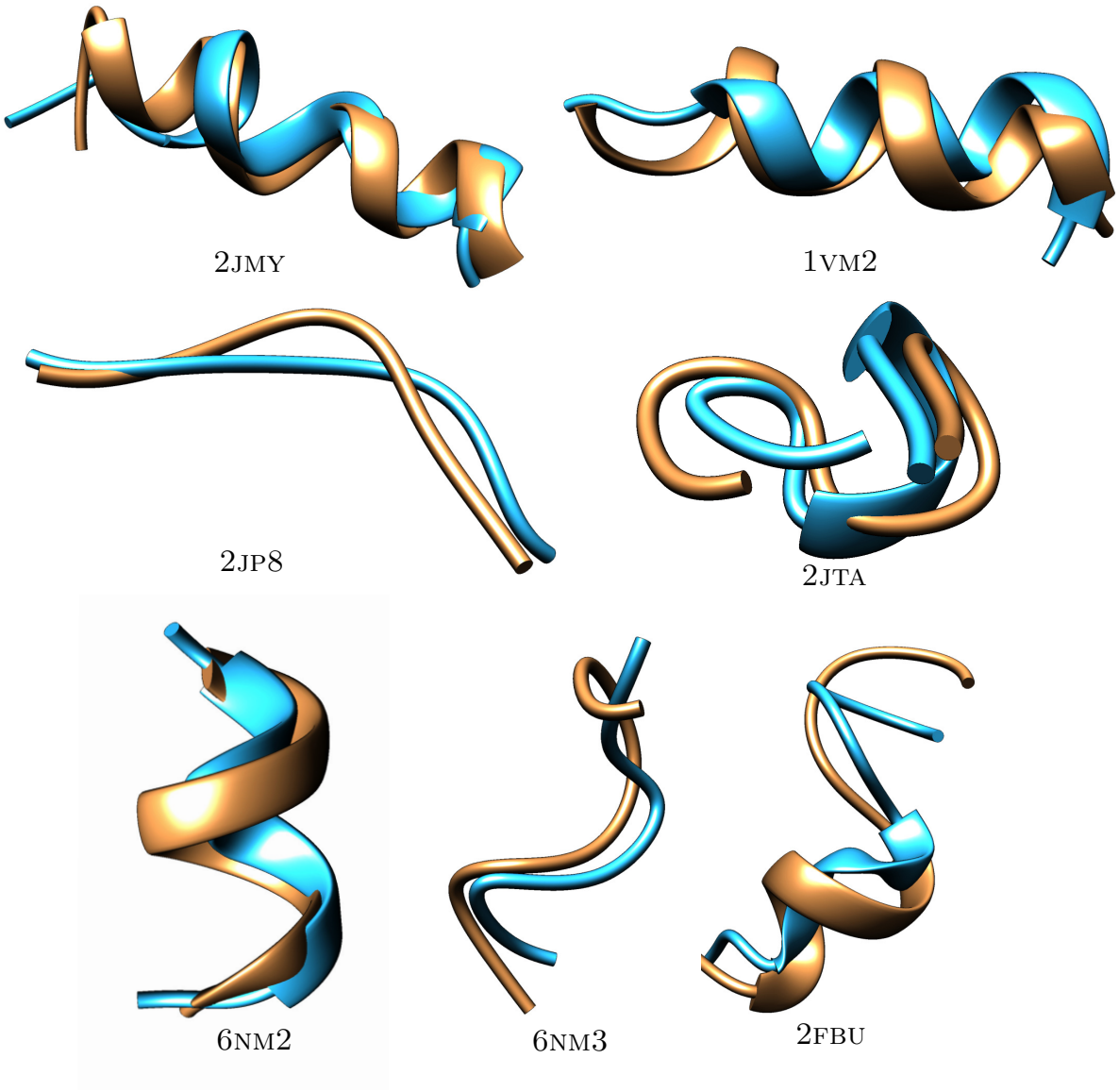



Figure 3.3 – The structures computed by Java MDjeep (blue) with lowest RMSD super-imposed with the structures from the PDB. The structures were not subject to energy minimization or any other post-processing step.

In Figure 3.3 we see the computed structures super-imposed with one of the measured (ground truth) structures from the PDB. Keep in mind that the structures that we see are the raw output of Java MDjeep and were not subject to energy minimization. Visually, the results for some proteins are better for others, which seem to match the scores reported in the table. For instance, the three proteins 2jmy, 6nm2 and 6nm3 had the lowest RMSD scores, and their computed structures look fairly close to the measured structure. The structure computed for 2fbu has the highest RMSD score among the proteins in the testbed, and this is also clearly visible in the figure. The helix in the computed structure does not look like a proper helix, which means that some of the backbone dihedral angles present in the structure are not correct. One interesting result is that in our computed structure, there seems to be a secondary structure element present in the protein 2jta (visualized as a thicker ribbon) which is not present in the measured structure of the protein. From these experiments, we can conclude that for small proteins, branch-and-prune (with refinement) is a useful tool and is capable of computing good-quality structures when using real protein data. For larger proteins, there are still some obstacles to be overcome, as we will see in the next section.

### 3.5.2 Covalent geometry and branch-and-prune

In the previous section, we saw that the uncertainty in genuine NMR instances can cause challenges for branch-and-prune. The obstacles discussed relate to interval distances and torsion angles which are **type 3** and **type 4**. However, in practice, there can also be a problem with the distances that are often regarded by BP methods as exact. These are the distances from force field parameters (**type 2**), which are bond lengths, bond angles as well as the $\omega$ torsion angle (which is often set to either $0°$ or $180°$). In practice, these distances and angles vary a lot from what is given by the force field parameter files. This issue was studied in [134], where we compared ibp-ng with an intermediate version of Java MDjeep. Through the use of the refinement step with SPG, MDjeep was able to identify solutions for all the proteins in the testbed, while ibp-ng was not. However, ibp-ng on average computed structures with better RMSD scores. Since then, Java MDjeep was further updated (using torsion angles for branching), which greatly improved the quality of the computed structures and which lead to much better RMSD values. In this section, we will extend the experiments presented in [134], using the newest version of Java MDjeep. These experiments aim to see the effect of the variations of the covalent geometry on the structures produced by branch-and-prune. In the previous section, we

conducted experiments with proteins for which the structures in the PDB were produced by NMR and simulated annealing. In this section, we will instead use proteins that were analyzed with X-ray crystallography. In particular, we selected 435 proteins from the PISCES server [194]. The criteria for selection were:

— Smaller than 100 residues

— No Proline present (all $\omega$ angles should be close to 180°)

— An identity smaller than 20% between the structures (they are not too similar)

— An X-ray crystallographic resolution better than 1.6Å (the accuracy of the X-ray measurements)

— An R-factor of at least 0.25 (how well the refined structure predicts observed data)

We start by presenting a small data analysis with these proteins which confirms that there are substantial variations present in the bond angles as well as in the $\omega$ torsion angles. This analysis was done on a subset of the proteins in the testbed. More details can be found in [134]. The analysis is summarized in Figure 3.4.

Firstly, we can see that the angles can vary quite strongly between amino acid types, which is not captured by the force field data (dashed line). We also see the deviations based on some Ramachandran regions of the proteins. These regions are defined by combinations of values that the $\phi$ and $\psi$ angles often take [195]. The twelve Ramachandran regions used in this analysis are further illustrated in Figure 3.5, which displays an approximation of the Ramachadran plot. Recall that the $\omega$ angles are generally assumed to be close to 180°. However, we see in Figure 3.4 that for several of these regions, it is especially striking how much the $\omega$ angles can deviate from 180°. In some cases it can even be around 150°. These variations can be a problem for the BP methods. For small proteins, such as those analyzed in the previous section, this may not be a problem. However, when we look at larger proteins, the error introduced by these variations can build up. The proteins that we experiment with in this section are up to ten times larger than the NMR proteins that we analyzed before, and will thus likely exhibit problems with the variations of covalent geometry.

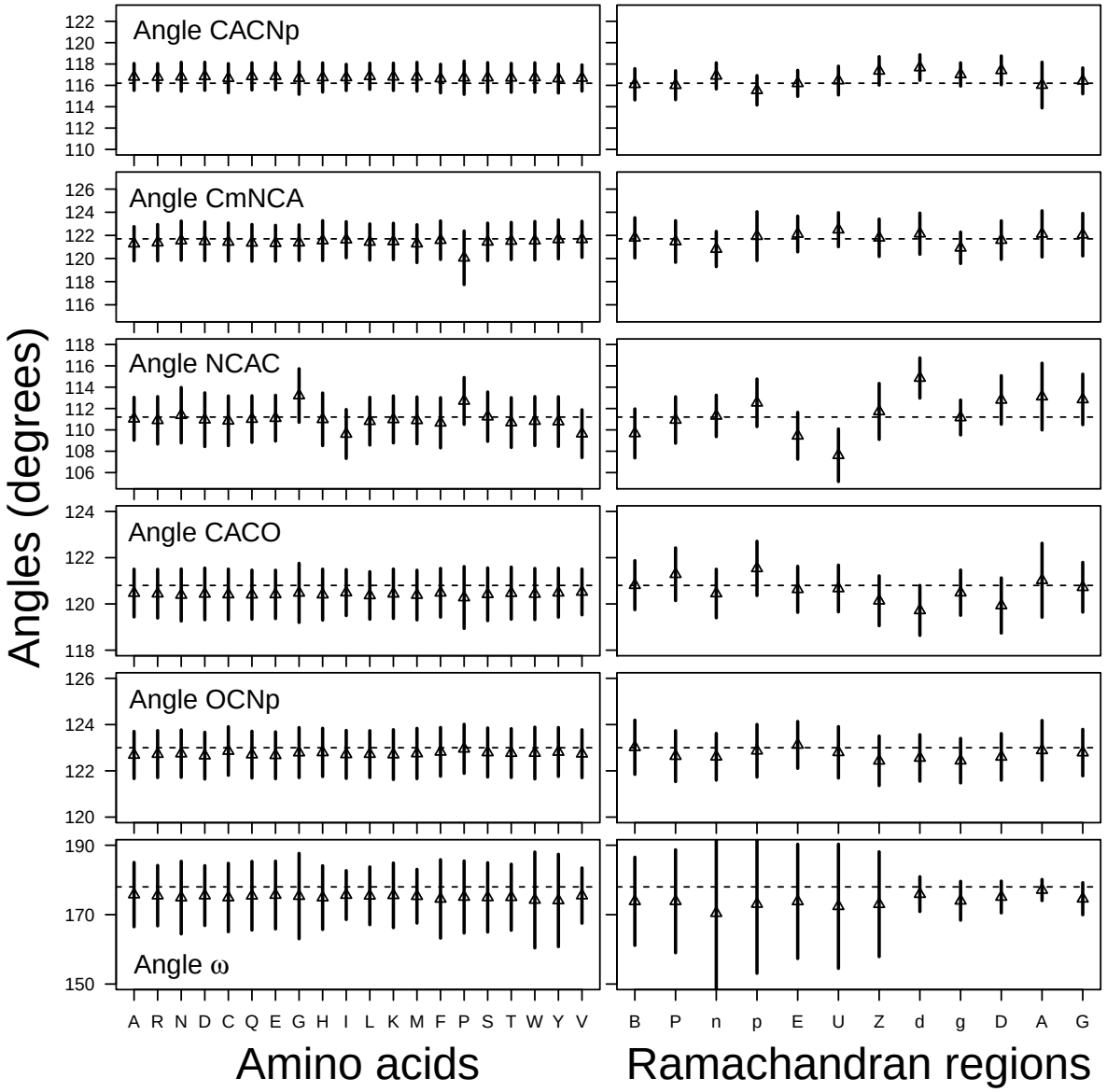



Figure 3.4 – The average values of the bond angles between the heavy backbone atoms (N, CA, C, O) as well as on the torsion angle $\omega$ of the peptide plane. Np denotes the nitrogen atom N of the next residue, Cm denotes the carboxyl carbon C of the previous residue, and CA denotes the CA atom. The distributions of these angle values are uni-modal, which means that we can use their averages as a suitable descriptor. On the left, we see the averages of the angles according to the amino-acid types, while on the right we see them according to the Ramachandran region of the residue. The horizontal dashed lines correspond to the values of the angles from the force field parameters [180].

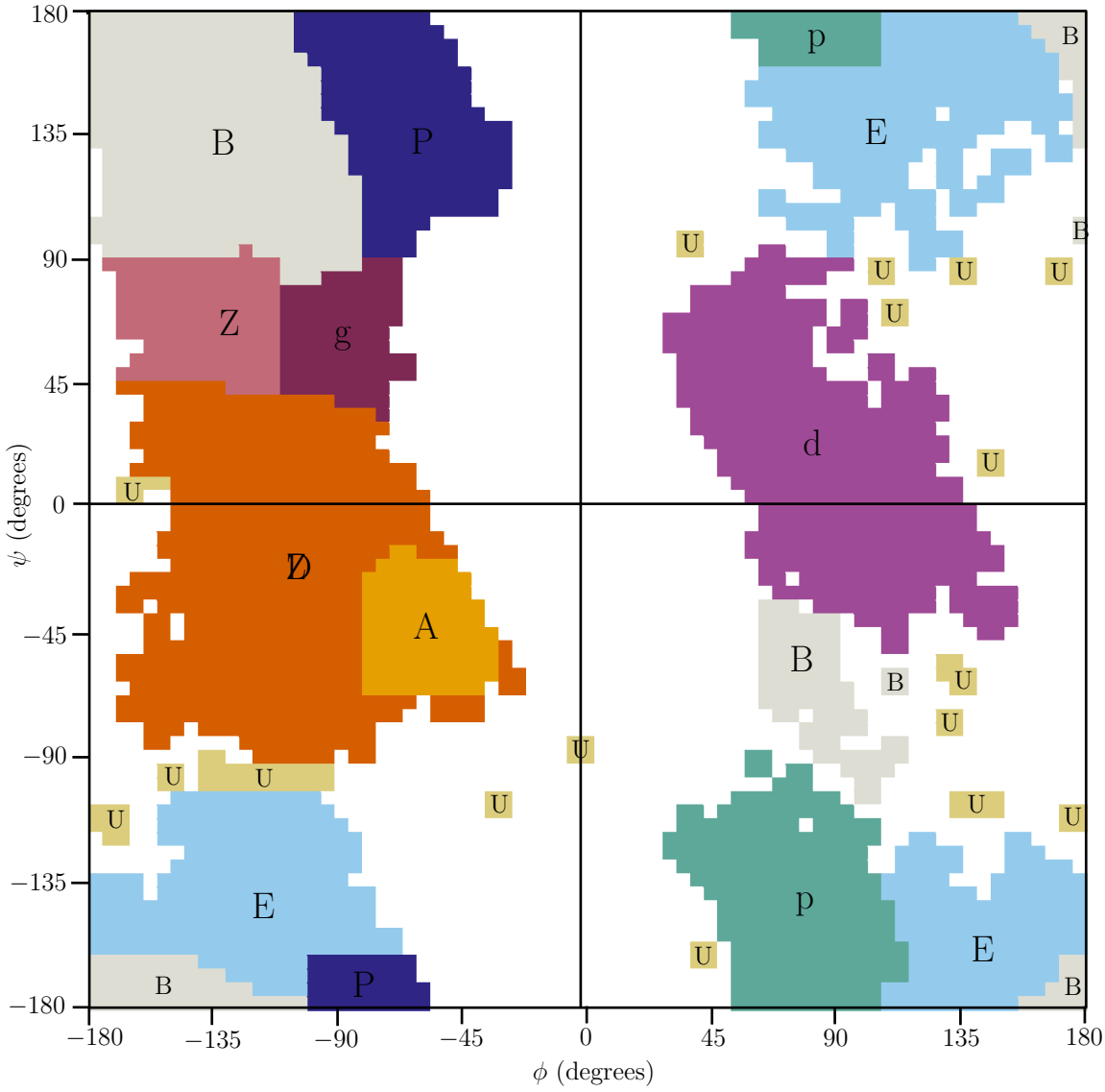



Figure 3.5 – The Ramachandran plot illustrating the 12 different regions used in the data analysis. Note that this figure is only an approximation of the regions based on a 5-degree grid.

To study how these variations can affect branch-and-prune, we ran experiments on two sets of instances, that will be referred to as instances A and B. Both these instances are generated using the same instance generation tool described in the previous section, which can be found on GitHub. The parsing of PDB files is done by the PDB class in the main Java package (see Figure 1.4). Instance A is a completely artificial instance, where we generate distances purely from the PDB files, introducing only a small amount of error. Instance B is an instance where we use genuine (**type 2**) distance information from the force field parameters and set $\omega$ to a small interval around 180°. Note that

since we are dealing with structures from X-ray crystallography, we do not have NMR restraints (**type 3**) in our instances because there are no hydrogen atoms present in the PDB structures. In fact, we should also not have access to torsion angles and distances relating to (**type 4**), because they are generally obtained by making predictions based on the chemical shifts from NMR spectra. However, we require these $\phi$ and $\psi$ torsion angles for the discretization of the instances, which is why they are simulated from the measured structures in the PDB files (with the introduction of some noise). In fact, for instance class B, the only artificial information that is included are these (**type 4**) torsion angles. The instances are further detailed in Table 3.3.

| **Distance/angle type** | Instances A | Instance B |
|---|---|---|
| Type 1<br>(van der Waals radii) | 60% of sum of radii | 60% of sum of radii |
| Type 2<br>(bond lengths and bond angles) | Exact distances and $\omega$ angle taken from PDB (capturing the variations) | From force field parameters, $\omega$ in a 5 degree interval around 180° |
| Type 2<br>(Improper torsion angles) | From force field parameters | From force field parameters |
| Type 3<br>(NMR restraints) | Not present | Not present |
| Type 4<br>(TALOS+ torsion angles) | From PDB placed randomly in 5° intervals | From PDB placed randomly in 5° intervals |

Table 3.3 – Details for the two instance classes that were experimented with. Instances A are completely artificial while instance B mixes artificial with information from force-field parameters.

There are some important details to note about these instance types. Recall that the geometric information captured by **type 2** and **type 4** is used for discretization, and that their distances are used as reference distances in branch-and-prune. For these instances, for every amino acid (except the very first one), we have access to the proper torsion angles $\phi, \psi$, and $\omega$ and we know the improper dihedral angles from the force fields. This means that whenever we are branching, for almost every candidate atom $v$ we know the dihedral angle $\tau$ ($\{u_3, u_2, u_1, v\} \in E_\tau$). Therefore, we will only branch based on the size

of the intervals of these torsion angles, which we know is 5°. Thus, the BP trees that are created for these instances have a small size. For these instances, with our parameters, Java MDjeep will only identify one solution because the other solutions are too close and are thus rejected. These instances are generated in this way specifically because we want to study the effect of the variations of the covalent geometry.

For each of the 435 proteins in the testbed, we created instances A and B and used Java MDjeep to find one solution for every instance. The parameters used were $\epsilon = 0.01$, $\rho = 1.0$ and the force field used was CHARMM [179]. Similarly to the experiments in the previous section, to measure the quality of the computed structures we use the RMSD metric (see Equation (3.1)). The results are visualized in Table 3.4 and further explored in Figure 3.6.

| **Results** | Instance A | Instance B |
|---|---|---|
| Mean | 0.79 | 3.45 |
| Median | 0.65 | 3.37 |
| Standard deviation | 0.62 | 1.78 |
| Worst RMSD score | 5.64 | 11.69 |
| Correlation coefficient size and RMSD | 0.49 | 0.80 |

Table 3.4 – The resulting RMSD statistics for the 435 proteins for instances A and B.

The results for the instances of class A are very good, with a mean of 0.79Å, which is a rather low score for proteins of these sizes. In fact, as we see in Figure 3.6, most of the proteins are in the first bin with RMSD scores between 0 and 2. Logically, for instance class B, the results are much worse, with much higher RMSD scores with a mean of 3.37Å. This was to be expected because the instances of class A rely mostly on data extracted directly from the X-ray structure that we are comparing with. The last line in Table 3.4 is also of interest, which displays the correlation coefficient between the RMSD scores and the size of the protein. We can see that the correlation between size and RMSD is much stronger for instance class B (0.80) than for A (0.49). This confirms the idea that there is a large build-up of errors when we do not account for variations in the covalent geometry. In instance A, there is still an intermediate correlation between the size of the protein and the RMSD. One of the reasons for this is that the uncertainty in the $\phi$ and $\psi$ angles

will also build up, but to a lesser degree.

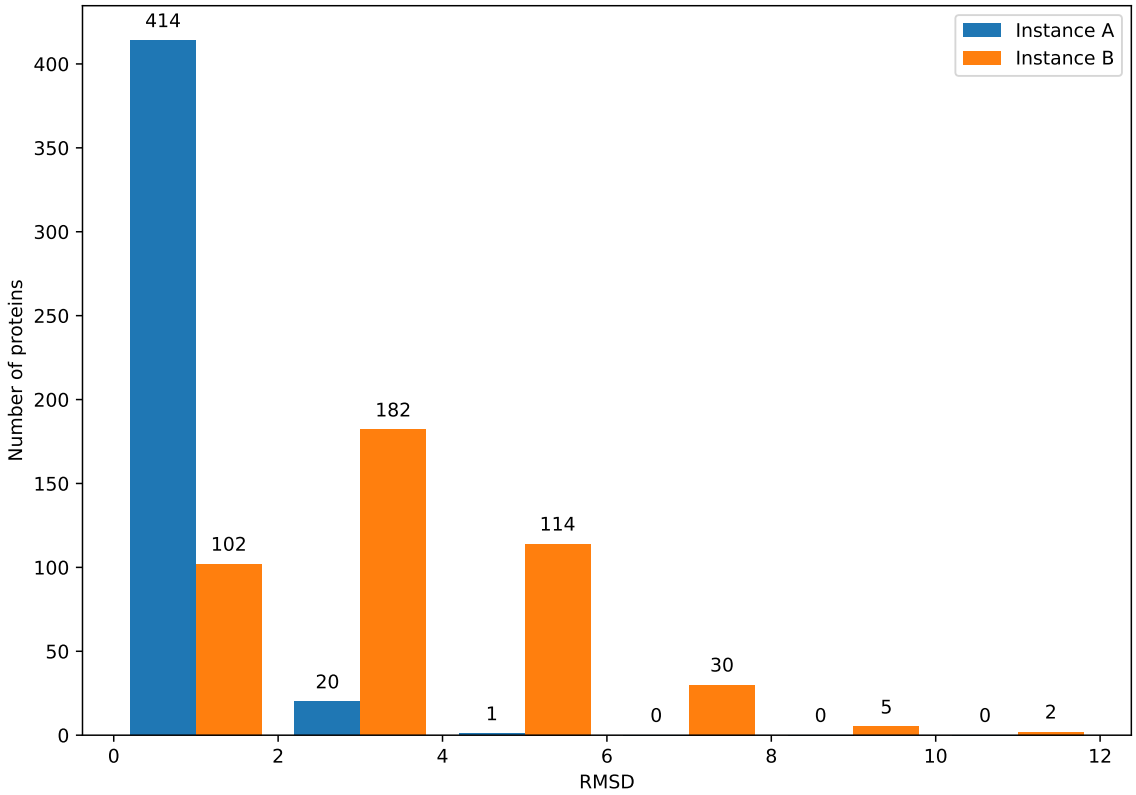
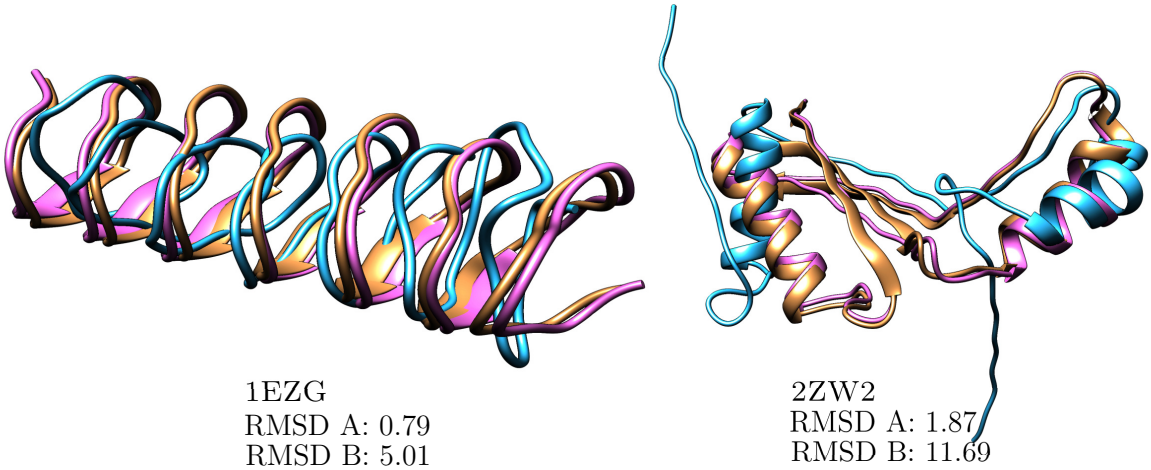



Figure 3.6 – A histogram with the resulting RMSD scores for the two instance classes, with bins of size 2Å.

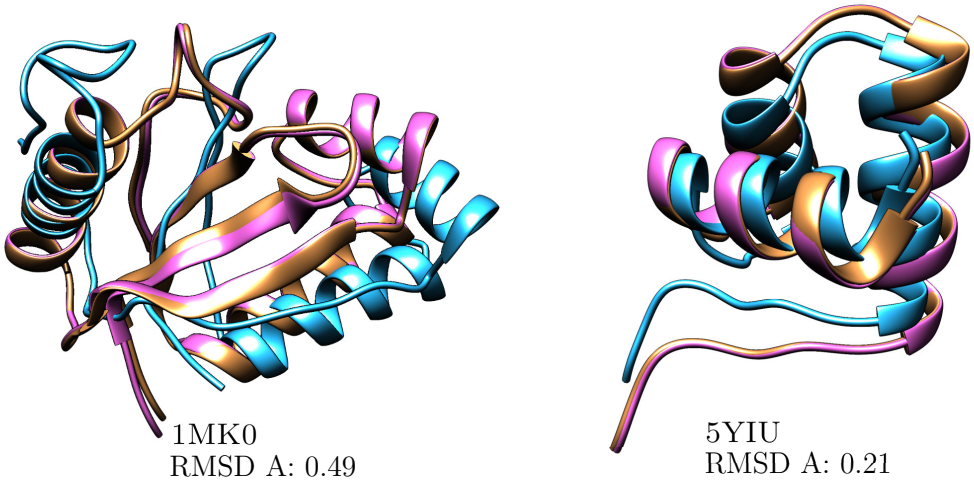
In Figure 3.7 we see four example proteins from the testbed. For each of the four proteins, the two generated structures with instances A and B are overlapped with the original, measured PDB structure. We see that as the RMSD scores suggest, the pink structures (instance A) are very close to the original structures, while the blue (instance B) one is not. In particular, for the protein 2ZW2 (which has the worst RMSD score of all instance B proteins), the blue structure deviates quite significantly. For the protein 5YIU, both structures that were generated look quite close to the original. This protein has 45 amino acids and is thus one of the smaller ones in the testbed, which can be one of the reasons why it was easier to solve. The only difference between the instance classes is the covalent geometry (**type 2**), which for instances A captures the variations, and for instances B is static. These much worse RMSD scores confirm that the variations in the covalent geometry pose a problem for branch-and-prune. However, despite these variations, Java MDjeep was able to find a solution for every protein in the testbed. Considering the small size of the search space, this is a good result, which is most likely due

to the refinement step with SPG (see Section 2.4). This refinement step allows MDjeep to identify solutions even when the distances in the input include error.



1EZG
RMSD A: 0.79
RMSD B: 5.01

2ZW2
RMSD A: 1.87
RMSD B: 11.69



1MK0
RMSD A: 0.49
RMSD B: 7.28

5YIU
RMSD A: 0.21
RMSD B: 2.50

Figure 3.7 – The two structures computed by Java MDjeep for instances A (pink) and B (blue) overlapped with the structure measured with X-ray crystallography (gold). Examples for four proteins: 1EZG, 2ZW2, 1MK0 and 5YIU. Underneath, we see the PDB code of the protein as well as the RMSD of the two produced structures.

## 3.6 Discussion

In this chapter, we discussed different methods for protein structure determination, ranging from experimental methods to distance-based approaches. Several experiments were presented, where the problem of protein structure determination was cast as an

instance of the Discretizable Distance Geometry Problem. These instances were subsequently solved by using a Java implementation of MDjeep which is a branch-and-prune solver written as part of this thesis. When considering real protein data, uncertainty in the input data is a significant issue for the BP methods. In the first set of experiments (Section 3.5.1) we focused on small proteins and peptides, where uncertainty in the experimental NMR data is the key issue. We showed that for these small proteins, despite a large degree of uncertainty, good quality structures may be produced by BP methods. In the next set of experiments, we focused on larger proteins and a second problem of uncertainty. When using BP algorithms, the distances relating to the covalent geometry are regarded as exact, while in practice there may be large deviations in these distances. When we do not capture these variations in the instances, this leads to a build-up of error and protein structures of poor quality. The experiments that were conducted in Section 3.5.2 confirm this hypothesis and show that there is a correlation between the size of the protein and the effect of the variations in the covalent geometry.

Future work on the topic of protein structure determination has several directions. Firstly, one strategy to solve the problem of the variations in the covalent geometry is to no longer rely on force field data for these distances. Instead, perhaps a method involving statistical analysis or machine learning may be used to identify the best distances. A work in which progress was made towards such a method can be found in [196]. It is important to capture some form of variations in these distances. For example, we may want to vary the bond angles based on the secondary structure which they are in, or the type of amino acid. Secondly, including long-range CB distances may improve the quality of the structures when working with larger proteins. These distances can be obtained from the contact maps that are created by machine learning methods. For instance, a previous version of AlphaFold, created these contact maps as a final output (the current version outputs the three-dimensional structure directly [169]).

When we overcome the problem of the variance in covalent geometry, perhaps we can focus on solving larger structures using genuine NMR data. A final direction of future research is visualizing the dynamics of protein structures. This relates directly to the topic of the next chapter, which focuses on the Dynamical Distance Geometry Problem.

Chapter 4

# Dynamical Distance Geometry & Motions

So far we have only considered static problems, where distances between objects are fixed, and where realizations of the distance graph $G$ are snapshots of structures. However, the framework of Distance Geometry may also be of interest when there is a dynamic component present. For instance, using a distance-based model may be useful in the case where we want to simulate objects in a way where specific distances between the objects are enforced. For example in applications where collisions are undesired, such as air traffic control. In this thesis, we focus on the application of human motion retargeting. We are mainly interested in the application of human motions because it opens the path to research on protein dynamics. Like human characters, proteins are skeletal structures, which means that the methods used for both applications are similar.

We will start the chapter with an introduction and definition of the Dynamical Distance Geometry Problem (Section 4.1). Next, we will discuss the application of human motion retargeting in Section 4.2. In Section 4.3 we will discuss experiments where we use Distance Geometry for the motion retargeting problem. Afterward, in Section 4.4, we discuss a new representation for human motions that was inspired by structural biology and that is useful for distance-based motion retargeting. We will end the chapter by discussing the results and exploring future lines of work (Section 4.5).

## 4.1 Introduction

The subclass of the DGP in the literature that relates to dynamic problems is known as the Dynamical Distance Geometry Problem (dynDGP) [4, 5]. The dynamic aspect of the subclass is captured by a set of temporal instants $T$. In the representation of the dynDGP, a vertex of our graph $G$ no longer only represents an object. Instead, it represents an object $v \in V$ at a certain point of time $t \in T$, creating a simple weighted undirected graph $G = (V \times T, E, d)$. Because our list of distances $\mathcal{D}$ may include distances between objects at different times, the mapping $d$ now maps an edge between two objects $u, v$ at times $q, t$ to a distance value:

$$d : \{u_q, v_t\} \in E \longrightarrow \delta(u_q, v_t) \in \mathcal{D}. \tag{4.1}$$

In order to define the dynDGP [26], we extend the optimization variant of the DGP that was given in Definition 2:

**Definition 8** *Given a simple weighted undirected graph $G = (V \times T, E, d)$ and an integer $K \in \mathbb{Z}_+$ the optimization-based dynDGP asks to determine a realization*

$$x : V \times T \longrightarrow \mathbb{R}^K$$

*such that a penalty function $\sigma$ is minimized.*

When working on the dynDGP it is common to choose a penalty function $\sigma$ which assigns varying priorities to different distances. For example, distances between two vertices $u_q$ and $v_t$ where $q$ and $t$ are close may be given more importance. For these instances, the mapping function $d$ does not only assign a distance value to an edge $\{u_q, v_t\}$ but also a priority $\pi(u_q, v_t)$. In the following, we will suppose that these priorities $\pi(u_q, v_t)$ are normalized between 0 and 1. An example of a priority-based penalty function for the dynDGP is:

$$\sigma(x) = \sum_{\{u_q, v_t\} \in E} \pi(u_q, v_t)(\|x_u^q - x_v^t\| - \delta(u, v))^2 \tag{4.2}$$

where $x_t^v$ is the position of the object $v$ at the time $t$. Like the penalty function described in Equation (1.4), the gradient of $\nabla\sigma(x)$ of an incumbent solution $x$ can be computed in an exact way [26].

The dynDGP has many different applications. In general, much work has been done in the context of animations and simulations. Because we are dealing with distances over

time, regarding problems as instances of the dynDGP can be very useful when we want to enforce distances between objects in an animation or simulation. For example, in the context of air traffic management, one evident goal is to make sure that no two airplanes are too close to each other at the same time. In [49, 197], a model with distances between airplanes over time is used for air-conflict resolution. Another practical application is crowd simulation [47, 48]. In this case, the distances in the model make sure that characters in the crowd do not walk into each other, creating collisions.

One main interest for the dynDGP is to model the various changes of protein conformations. When visualizing a protein moving from one conformation to another, it is important that certain inter-atomic distances are enforced. Furthermore, unwanted collisions should be avoided.

In this thesis, we focus on the application of human motion retargeting. The reason for the choice of this topic is that the human motions are akin to protein dynamics in the way that the main characters in the motions both have a skeletal structure. The bones in the human character can be seen as analogous to the bonds in the protein structures. In the next section, we will look at human motions and motion retargeting in more detail.

## 4.2 Human motion retargeting

*Human motion capture* is the process of recording the movement of human actors [198]. This procedure is widely used for Computer Generated Imagery (CGI) in the movie industry but is also important in the context of making animations for video games. Once captured and digitalized, one may want to copy the movement of the actor to an animated character. However, a problem arises when the human actor and the animated character have a different morphology. For example, the animated character may have limb sizes and proportions that are non-anthropoid. In this case, it is a challenge to copy the motion from the human to a character with different proportions. This problem is known in the literature as *motion retargeting* or *motion adaptation* [15–17, 105]. When performing motion retargeting, it is crucial that contacts between joints that were present in the original motion are transferred to the new motion. It is equally important to avoid contacts that were not there to begin with. Therefore, using distance-based models for this problem seems natural and we can consider motion retargeting as an instance of the Dynamical Distance Geometry Problem.

### 4.2.1 Representing human motions

In these motions, we have a skeletal human character that changes its posture over time. The *anatomy* this character can be defined by using a graph $H = (V, B)$, where a vertex $v \in V$ is a joint of the character and the edge $b \in B$ is a bone. These graphs $H$ are trees, in which every joint $v$ has a unique *parent* joint, assigned by a function:

$$p : v \in V \setminus \{v_0\} \to p(v).$$

To finish the representation of our characters, we need the function

$$\chi : v \in V \longrightarrow \chi(v) \in \mathbb{R}^3.$$

This function maps a three-dimensional offset from every joint of the character to its unique parent joint. The value $||\chi(v)||$ corresponds to the length of the bone $\{v, p(v)\} \in B$. This function $\chi$ together with the anatomy $H$ lets us define the morphology of the skeleton $(H, \chi)$ [105].

If we add a fictive root joint $v_0$ to the tree $H$ and fix it at the position $(0, 0, 0)$, we can use the offsets between a joint $v$ and its parent defined by $\chi$ to find a realization of the *initial posture* $x_0$ of the skeleton:

$$x_0 : v \in V \longrightarrow \begin{cases} (0,0,0) & \text{if } v = v_0, \\ \chi(p(v)) + \chi(v) & \text{otherwise.} \end{cases}$$

Figure 4.1 shows an example of a commonly used initial posture for these human motions. The motion itself is then defined by the changing positions of the joints $V$ over time. Time is defined by a series of frames $m$ frames $t \in T$, with $T = \{1, 2, \ldots, m\}$. A straightforward choice for representing the motion of the character would be to use Cartesian coordinates to assign a three-dimensional position to each joint changing over time. In this case, we may extend the function $\chi$ and let it vary for every frame $t$:

$$x_t : v \in V \longrightarrow \begin{cases} (0,0,0) & \text{if } v = v_0, \\ \chi_t(p(v)) + \chi_t(v) & \text{otherwise.} \end{cases} \tag{4.3}$$

However, this simple Cartesian representation has one large drawback: the morphology of the skeleton is not explicitly captured. When we generate new motions (for instance when we perform motion retargeting) there is a chance that the bone lengths will not be

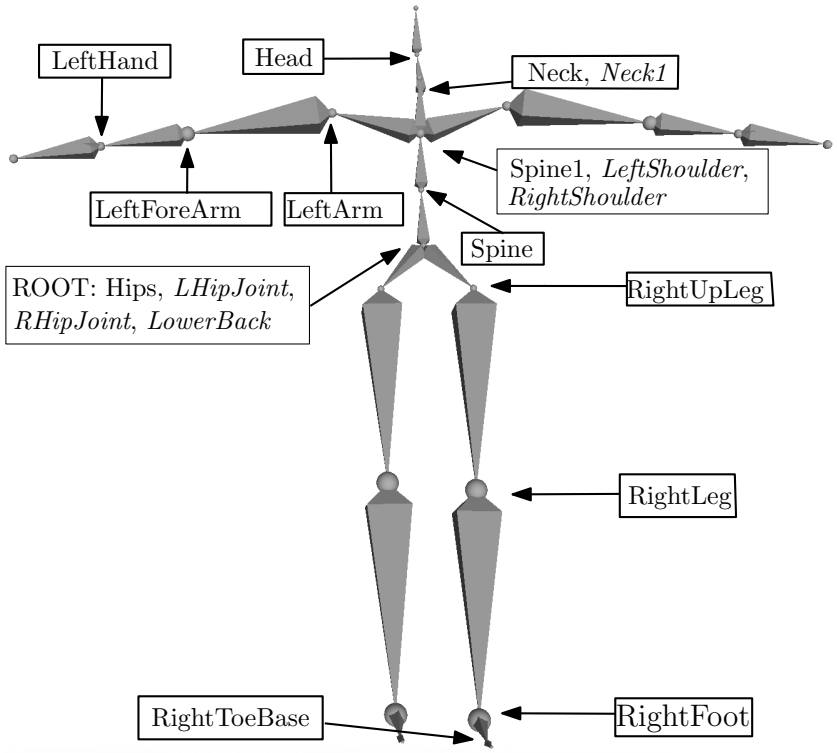



Figure 4.1 – An example of a skeletal structure $(H, \chi)$. The initial T-pose along with the labels are commonly used in in BVH files [199]. Joints with $|\chi| = 0$ are shown next to their parent joint, marked in italics. For the arm, only the left-side joints are labeled, for the legs only the right-side labels are shown. The root of the tree $H$ is the "Hips" joint. Note that this is the original root of $H$, and not the fictive root node $v_0$, which is instead placed at the origin of the coordinate system. This image and other human motion images that we will present in this thesis are generated using the free software Blender [1].

constant throughout the animation. To be able to encapsulate the constant nature of the bone lengths, the standard way of describing human motions utilizes an Euler angle representation $\rho$. At every frame $t$, we assign a triplet of Euler angles $\theta$ (pitch), $\phi$ (roll), and $\eta$ (yaw) to every bone of the skeletal structure representing our character. Together with the known offset between $p(v)$ and $v$, we can define a transformation matrix $M_t(v)$ that takes into consideration both the translation data from $\chi(v)$ and the given Euler angles. A precise expansion of the matrix $M_t(v)$, based on a compound rotation matrix $R$ as well as three translation matrices, can be found in [199]. To compute the position of the joint $v$, all offsets and all Euler angles from the root node of $H$ up to $v$ need to be taken into consideration. The formula capable of converting the Euler angles in absolute

1. `https://www.blender.org`

positions for the joints is therefore:

$$\rho_t : v \in V \longrightarrow \begin{cases} (0,0,0) & \text{if } v = v_0, \\ \prod_{k \in P(v)} (M_t(k)[0,0,0,1])^T & \text{otherwise,} \end{cases} \tag{4.4}$$

where $P(v)$ is set of vertices $u$ that form the unique path from $v_0$ to $v$ over the tree structure of the graph. See [199] for a more detailed examination of the Euler angle representations. In fact, the Euler representation is the one used in the BVH file format, which is discussed in the same reference, and which is the standard format for human motions. In this representation, only the rotation of the bones changes throughout the motion, which means that the bone lengths remain constant. Note that $\rho$ uses 3 degrees of freedom per bone, which is comparable to the degrees of freedom of the Cartesian representation (see Equation (4.3)). However, it is possible to represent these human motions while using fewer degrees of freedom, while still keeping the constant morphology. More detail on this will be given in Section 4.4.

### 4.2.2 Motion retargeting in the literature

When we perform motion retargeting [15], the interest is in adapting an original motion, given for the character $(H, \chi)$, to a character having a different morphology $(H, \hat{\chi})$. Note that both characters have the same anatomy $H$. We will refer to the newly generated motion as the *target motion*. One of the most straightforward ways to generate the target motion is by Euler angle transfer. For each frame, one simply copies the Euler angles ($\theta$,$\phi$, $\eta$) from the original joints to the joints of the target morphology. While this is easily accomplished, it can quickly lead to problems in the output motion. The most common problems that arise when using this method are that desired inter-joint contacts are not preserved in the output method and that unwanted collisions may be introduced. Examples of this are shown in Figure 4.2. To solve this problem, several methods are introduced in the literature.

In [15], a non-linear constrained optimization method was used for motion retargeting. First, important features in the original motion are identified, such as the feet of a character touching the ground. Non-distance-based features were used as well, such as the vector between two points having a specific orientation. For each of these features, a constraint is introduced. Solutions are obtained by scaling the original motion while observing the various constraints. A method that is closer to Distance Geometry was

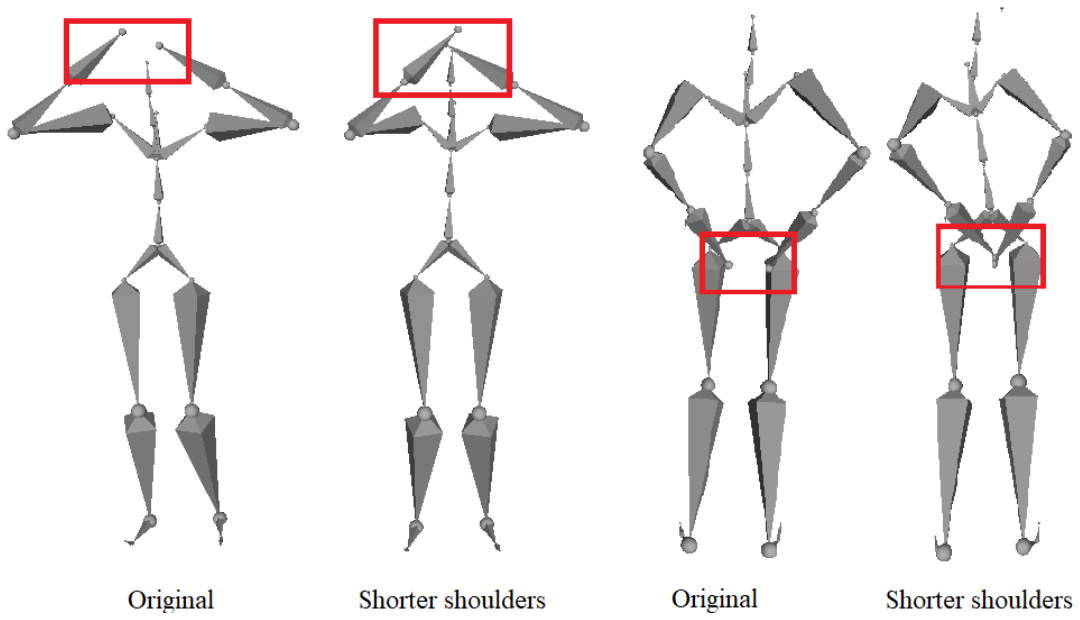



Figure 4.2 – Simple examples of frames in motions retargeted using the Euler angle transfer method, where the motion is retargeted to a character with shorter shoulders. Two frames are selected of a motion with a character dancing the well-known "Macarena" dance. On the left, we see a frame where the character puts their hands on their head. In the retargeted motion we see that a collision is introduced, where the hands are inserted in the head. On the right, we see another frame where an undesired contact is introduced in the retargeted motion.

proposed in [16]. Here, *interaction meshes* are used to represent the spatial relationships between nearby body parts. Then, target motions are generated by transforming these interaction meshes directly using Laplacian deformation techniques. In [17], many other methods are surveyed, including approaches that make use of statistics to perform motion retargeting.

We will now focus on methods that are based on Distance Geometry and which regard motion retargeting as an instance of the Dynamical Distance Geometry Problem, where inter-joint distances are considered and the temporal instants correspond to the frames $T$. The distance-based approach is natural considering that one of the goals of motion retargeting is to retain contacts that were present in the original motion and avoid collisions that were not. This can be achieved by enforcing certain distances between joints in the target motion. In the literature, only inter-joint distances at the same frame $t$ were considered [105, 106], and instead of solving the dynDGP in one shot, it was solved as a series of static DGP instances. At every frame $t$, we have the DGP instance $G = (V, E, d)$, where the objects $V$ are the joints of the human character and the edges $E$ combined with

the mapping $d$ represent inter-joint target distances. Some of these distances relate to the bones $B$ of the anatomy of the skeleton, while others are between joints that are connected via a series of bones in the anatomy. The target distances for the target motion are obtained by transforming the inter-joint distances of the original motion. The transformation is done by first normalizing the distances in the original motion, using the morphology of the original skeleton. Then, the target distances are computed by "denormalizing" the distances using the morphology of the target character. When we have computed all the distances, instead of solving the dynDGP instance in one shot, at every frame, we solve an instance of a static DGP. Afterward, the resulting poses are combined into one smooth animation using standard animation techniques.

The input for the method is the original human motion as a BVH file (the standard motion capture format) as well as the morphology $(H, \hat{\chi})$ of the target character. An overview of the full procedure is given below.

1. For each frame $t$:
   (a) Derive Cartesian coordinates for each joint $v$, obtaining the pose $x_t$ in the original motion
   (b) For every pair $(u, v)$ compute the target distance $\delta(u, v)$ by normalizing and denormalizing the distances in the original motion
   (c) Assign priorities $\pi(u, v)$ to the target distances
   (d) Solve the DGP instance arising from these target distances, obtaining the (Cartesian) pose $\hat{x}_t$ for the target motion
2. Combine resulting poses $\hat{x}_t$ into one animation

The first step (a) in the procedure is straightforward and involves converting from Euler space to Euclidean space and computing several distances. Step (b), the normalization and denormalization step, is more complicated. To achieve this, we start by computing all shortest paths $P(u, v) = \{w_1, \ldots, w_k\}$ between pairs of distinct vertices, where $w_1 = u$, $w_k = v$ and, for every $i = 1, \ldots, k-1$, we have $\{w_i, w_{i+1}\} \in E$ (notice that the term "shortest" makes reference to the number of edges that need to the crossed by the path to walk from the vertex $u$ to the vertex $v$ of the anatomy graph $H$). To normalize a distance $\delta(u, v)$, it is divided by the weight $s(u, v)$ of the *shortest path* $P(u, v)$ connecting $u$ to $v$ in the morphology. The weight $s(u, v)$ of the chain $P(u, v)$ can be computed by combining

the lengths of the bones we need to cross to get from $u$ to $v$ in the anatomy:

$$s(u,v) = \sum_{i=1}^{(|P(u,v)|-1)} ||\chi(w_i) - \chi(w_{i+1})||. \tag{4.5}$$

Note that the weight of a chain $s(u,v)$ in the original morphology $\chi$ is not the same as the weight $\hat{s}(u,v)$ in the target morphology $\hat{\chi}$, because the bone lengths are different between the morphologies. For each pair of joints $(u,v)$, the denormalized target distance (at frame $t$) is computed as:

$$\delta(u,v) = \hat{s}(u,v) \cdot \frac{||x_u^t - x_v^t||}{s(u,v)}, \tag{4.6}$$

where $x_v^t$ is the position of $v$ at frame $t$ in the original motion.

The kinetic chains $P(u,v)$ are also used for step (c) where we assign priorities $\pi(u,v)$ to each target distance. The idea is that distances that relate to joints that are close together in the anatomy should have higher priorities. This means that the smaller the cardinality of the path $P(u,v)$, the higher the priority $\pi(u,v)$ should be. We start by identifying the longest chain $P_{max}$ between any two joints in the anatomy, and use this to determine the priority for any pair of joints $(u,v)$:

$$\pi(u,v) = \frac{|P_{\max}| - |P(u,v)| + 2}{|P_{\max}|} \tag{4.7}$$

Note that when $\delta(u,v)$ relates to a bone length, $|P(u,v)| = 2$, which means that $\pi(u,v) = 1$, which is the maximal priority value. After we have all the distances and priorities, the DGP can be solved by minimizing the penalty function in Equation (4.2). Because the gradient of this penalty function $\sigma$ can be computed exactly, a logical choice is an optimization method that relies on gradient descent. In [105, 106] a SPG method similar to the refinement step for MDJEEP was used. For more information about SPG, see Section 1.4.5. When solving for $x_t$, the starting point of the optimization is the result for the previous frame $x_{t-1}$.

This distance-based approach gives good results and results in retargeted motions where undesired collisions are avoided and desired contacts are retained (see [105, 106]). However, there is one drawback to this method and results as presented in the literature. Throughout the process, the poses $x_t$ and $\hat{x}_t$ are represented in the Cartesian way (see Equation (4.3)), instead of using the popular Euler angle method. This way, the penalty function $\sigma$ is easily computed and its gradient is exact. However, as we noted in

the previous section, when we use this representation we cannot guarantee that the bone lengths are constant. In fact, SPG may slightly perturb the bone lengths at each frame. This is partly mitigated by the fact that large priority values $\pi(u, v)$ are used when the edge $\{u, v\}$ corresponds to a bone $b$ in the anatomy. In this case SPG will be less likely to change the length of $b$. In the next section, we present a different approach that solves this problem entirely, where the poses $x_t$ are represented directly in Euler space.

## 4.3 Distance-based motion retargeting in Euler space

In this section, we will present a method that extends and improves the motion retargeting approach from [105, 106]. More details on the extension and the new experiments are found in [18]. Several new ingredients are added to the approach. In general, the goal was to address two issues with the original distance-based method. Firstly, because we have no inter-frame distances in the instances, the step from one frame to another can theoretically be quite large. This leads to potential issues for the last step of the method, where the poses are combined into one smooth animation. The second problem arises when we use the Cartesian representation for poses $x_t$, as this way we allow the method to change the bone lengths from frame to frame.

To address the first problem, new inter-frame distances are introduced into the instances, between joints in subsequent frames. We include all distances between pairs of joints in subsequent frames, where the distance values are taken straight from the original motion. This means that the DGP instances at each frame no longer only contain the target distances that result from the normalization and denormalization process. Despite these new inter-frame distances, we still solve the problem frame by frame. The distance function $\delta$ for the DGP instance at frame $t$ will then look like this:

$$\delta : \{u_q, v_t\} \in E \longrightarrow \begin{cases} \dfrac{\hat{s}(u,v)}{s(u,v)} ||x_u^t - x_v^t||, & \text{if } t = q, \\ ||x_u^q - x_v^t||, & \text{if } u = v \text{ and } q = t-1, \end{cases}$$

Apart from introducing these new distances, the way priorities $\pi$ are assigned is also changed. Firstly, the newly introduced inter-frame distances are assigned the maximal priority of 1. For the other distances, the revisited approach improves the priority calculation by exploiting the information given by the *interaction distance* [200] between two

joints $u$ and $v$ at a certain time frame $t \in T$. The interaction distance allows us to predict the distance that the two joints *will have* if their current relative movement (computed by comparing the current joint positions with the positions of the same joints at the previous frame) will not change in the subsequent frames. When two joints are moving one towards the other, their relative distances over time are important for performing the adaptation. This is because they can guide the movement towards a joint contact that we want to preserve. Alternatively, the relative distance over time will indicate potential self-contacts in approaching joints that we wish to avoid. Therefore, we also assign a higher priority to the distances between joints $u$ and $v$ for which the corresponding interaction distance $I(u_t, v_t)$ is smaller than a given positive threshold $\Delta$. To sum up, the new $\pi$ function has the following form:

$$\pi : \{u_q, v_t\} \in E \longrightarrow \begin{cases} 1, & \text{if } q = t - 1, \\ 1, & \text{if } q = t \text{ and } I(u_q, v_t) < \Delta), \\ (|P_{\max}| - |P_{uv}| + 2)/|P_{\max}|, & \text{otherwise.} \end{cases}$$

To address the problem regarding the flexibility of the bone lengths, we instead represent the poses of the target motion directly in Euler angle space. SPG then works by permuting the Euler angle variables, which makes sure that we will not be able to change the length of the bones of the character in the target motion. Furthermore, it makes it possible to easily output the BVH files for the resulting target motion. However, it complicates several of the steps of SPG. When we have an incumbent solution $\hat{x}_t$ and we want to compute the penalty function $\sigma(\hat{x}_t)$, we first have to convert from Euler space to Cartesian space so that we may compute the Euclidean distances (see Equation (4.4)). Therefore, the gradient $\nabla\sigma(\hat{x}_t)$ is no longer easily computed, because we have to also capture the conversion from Euler space to Cartesian space in the derivation. In practice, instead of using the exact derivation, we relied on *numerical differentiation*, using *finite differences* methods [121]. Specifically, we used the five-point stencil. While this kind of differentiation is slow, we do not have many vertices (joints) $V$ in the instances, so for this application, it works quite well.

This new motion adaptation method was programmed in Java, and several experiments were conducted on human motions from the Graphics Lab Motion Capture Database, [2] provided by Carnegie Mellon University. Our Java code accepts a BVH file in input,

2. `https://mocap.cs.cmu.edu`

containing the original motion, and outputs the retargeted motion in the same format. We will not report the numerical values of the stress function for the frames of the obtained motions because they do not always reflect the visual correctness of the postures and the movements. Instead, we compare the motions that we generate to the motions that are obtained by Euler angle transfer. The first motion that we consider is the "Macarena" dance (database entry code 135_35, see Figure 4.3). At a certain point, the character is supposed to place both hands on its head. The original frame is shown in the left-most image in Figure 4.3.

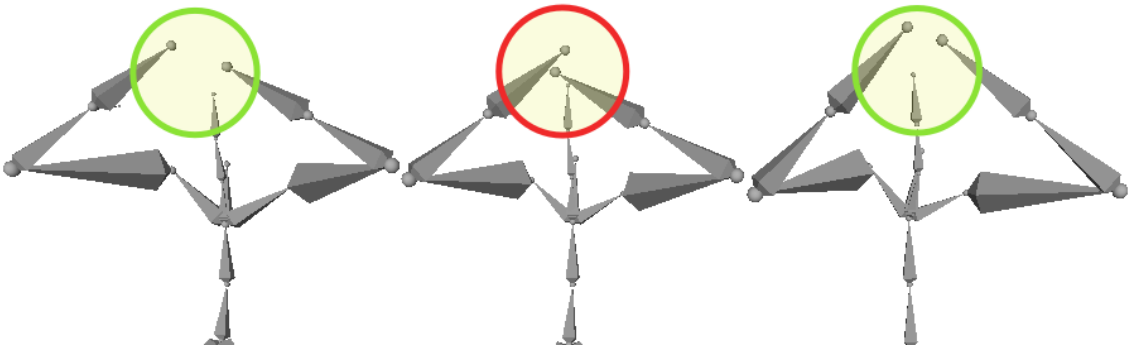


Figure 4.3 – A frame from a motion representing the macarena dance. From left to right: the posture in the original frame; the posture modified by simply transferring the original Euler angles to the different morphology; the posture obtained by our distance-based motion adaptation. The shoulders are 30% shorter in the target character as compared to the original animation.

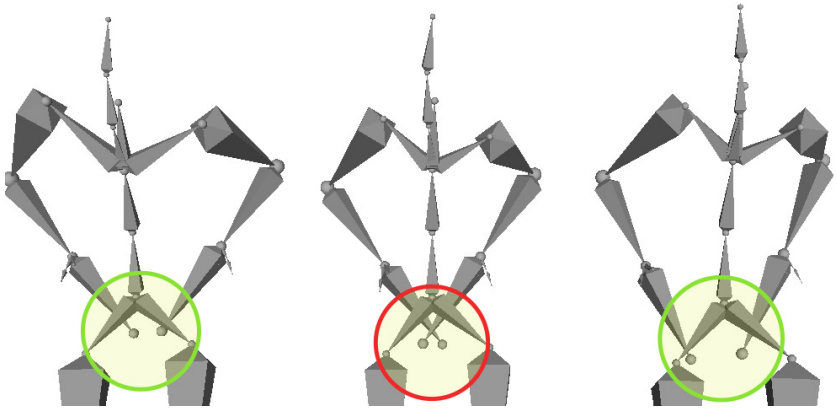


Figure 4.4 – Another frame for the macarena dance. From left to right: the posture in the original frame; the posture modified by simply transferring the original Euler angles to the different morphology; the posture obtained by our distance-based motion adaptation. The shoulders are 30% shorter in the target character. The distance between the two hands does not exactly correspond to the original distance in our solution, but the local distances around the hands are well preserved, allowing the viewer to essentially perceive the same posture.

We can notice that, when the shoulders are reduced in length (−30%) and the original Euler angles are simply transferred to the new character (see central image), the two hands approach each other too much, and if one imagines where the head of the character is supposed to be, the viewer has the impression that the hands penetrate the character head. In the right-most image in Figure 4.3, our solution shows a correct adaptation of the Euler angles to preserve the distance between the hands. In the same motion, the hands of the character are placed on its back a few frames later. Another undesired effect in the animation appears if we simply transfer the Euler angles. Figure 4.4 compares the original, the angle-transfer result, and the result obtained with our approach. We see that in our result, the distance between the hands is retained.

The next motion shows a character that feels like it is cold. To warm up the hands, it rubs them together (entry code 79_68, see Figure 4.5). When the adapted character with broader shoulders tries to warm up the hands, its two hands are too far from each other (see central image in Figure 4.5). The right-most image shows instead that this artifact is not present in our retargeted motion. These two retargeted motions, as well as a third one, can be fully viewed in a YouTube video [3].

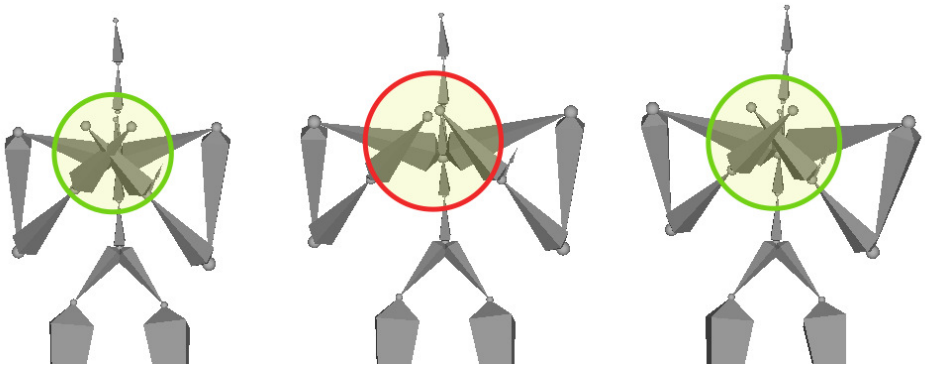


Figure 4.5 – A frame for the character feeling cold and hence rubbing its hands. From left to right: the posture in the original frame; the posture modified by simply transferring the original Euler angles to the different morphology; the posture obtained by our distance-based motion adaptation. The shoulders are 30% longer in the target character. In the angle-transfer solution, the two hands can hardly touch one another.

Finally, one may wonder whether we can still obtain such good results when the modifications on the bone lengths are more extreme (more than 30% modification of the original length). In this case, as expected, the results get worse and worse. To better deal with these more important changes, we have implemented an intermediate skeleton

3. `https://www.youtube.com/watch?v=V5tkvTNRf1E`

approach (see Figure 4.6). The idea of using intermediate skeletons to improve the results of motion retargeting was initially proposed in [201] in an inverse kinematics approach; we have simply re-implemented it in our context. Instead of attempting to retarget a motion with large morphology changes, the idea is to perform intermediate retargetings in a sequence, in order to *smoothly* approach to the desired morphology. Every skeleton in the sequence has one or more intermediate morphologies between the original $\chi$ and the target $\hat{\chi}$. In the experiment depicted in Figure 4.6, we can see that the intermediate skeleton approach (with 5 intermediate morphologies) can improve the quality of our retargeted motions when the changes in the morphology are more drastic.

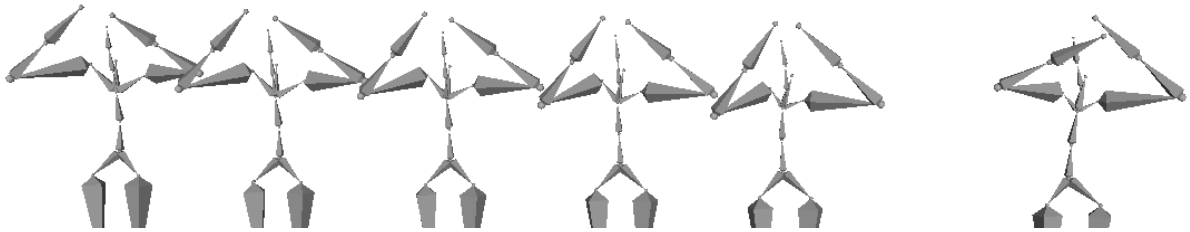


Figure 4.6 – On the right-most image, the result of retargeting the macarena dance frame shown in Figure 4.3 when the shoulders are 50% shorter. The sequence on the left side of the figure shows the 5 steps to move from a 10% modification to the final 50% modification, with a change of 10% per step, necessary to obtain a good-quality solution when the changes in the morphology are more important.

## 4.4 A representation for human motions inspired by structural biology

Aside from these experiments, research was done on a new representation for human motions [18, 19]. In this section, we will discuss this new representation. Furthermore, we will present a statistical analysis of a motion capture database to underline the relevance of this new representation to motion retargeting.

### 4.4.1 The vector-torsion representation

It appears that the Euler representation for human motions in motion capture may have been inspired by a remarkably similar representation in robotics. Various representations are used in robotics, such as the Denavit-Hartenberg representation [202]. One of the popular representations uses a triplet of Euler angles to describe the orientation

of a robot arm, which combined with the three-dimensional position of the end-effector makes it possible to generate a realization for the arm [203–205]. This makes for a *minimal* representation which uses 6 degrees of freedom. The use of Euler angles seems to work well for controlling a robot arm, which was conceived to directly execute the corresponding angle variations. However, the use of Euler angles leads to representational singularities for the problems of kinematic and dynamic control [206]. A singularity that can occur using the Euler angle representation is a gimbal lock [207]. This occurs when the middle rotation in the sequence makes the rotation axes of the first and third rotations parallel. When these axes are parallel, a change to either the first or third Euler angle will lead to the same rotation of the object in question (i.e., the robot arm). Essentially, such a singularity leads to the loss of a degree of freedom. To overcome this problem, an alternative, *non-minimal* representation in robotics was conceived as well, to avoid this problem. Similarly, in the context of human motions and motion capture, the usefulness of the Euler angles is debatable. We identify four drawbacks of the popular Euler angle representation for motions. Firstly, as outlined in [208], and explained above, Euler angle sequences admit different singularities, which may cause problems (such as a gimbal lock) for the representation when the angles approach the values leading to these singularities. Secondly, for a given bone, i.e. an edge $\{u, v\} \in E$, various combinations of Euler angles can place the joint $v$ in the same position [209]. Thirdly, the Euler representation is not optimal in terms of degrees of freedom. As we will see, it is possible to represent, in fact, the same postures by employing two degrees of freedom only, instead of three, for most of the skeleton bones. A final drawback of the use of Euler angles is that many of such angle triplets correspond to unnatural human postures for human joints, which cannot be filtered out, at least by simply imposing constraints on their values. This problem was already encountered and partially addressed in [210], where the focus was specifically on the human shoulders.

The representation that we will discuss here aims to address these problems. It was inspired by structural biology and uses two angles that we have seen before in the previous chapters: a vector and a torsion angle. These angles give rise to the name: the *vector-torsion* representation [18, 19]. This pair of angles will sometimes replace the triplet of Euler angles employed for the representation of a joint. Differently from an Euler angle, which depends only on the joint $v$ itself and on its parent $p(v)$, the *vector angle* $\zeta_v$ depends on $v$, $p(v)$, as well as on $(p \circ p)(v)$. Moreover, the *torsion angle* $\tau_v$ depends on $v$, $p(v)$, $(p \circ p)(v)$, as well as on $(p \circ p \circ p)(v)$. Despite this extra dependence, only the main joint

$v$ is indicated as a subscript of the angle names $\zeta_v$ and $\tau_v$; this is done to have a lighter notation. However, it will be supposed that, every time these two angles are taken into account, the necessary ancestors of $v$ all exist.

Given a morphology $(H, \chi)$, we can derive these angles by following the natural vertex order present in the anatomy $H$ [18, 19]:

**Definition 9** *Given a skeletal structure $(H, \chi)$ and one realization $x$, the vector angle $\zeta_v$ for the joint $v$ in this realization is the **smallest** angle (in the range $[0, 180°]$) formed by the line passing through $x((p \circ p)(v))$ and $x(p(v))$, and the line passing through $x(p(v))$ and $x(v)$.*

We can remark that variations on values of the vector angle $\zeta_v$ imply movements of the joint $v$. However, as for the Euler angles (because three of them are necessary to reconstruct the motion for each joint), one vector angle, alone, cannot be used to fully represent the motion of $v$. We therefore couple the $\zeta_v$ angle with a torsion angle $\tau_v$, as described in [18, 19].

**Definition 10** *Given a skeletal structure $(H, \chi)$ and one realization $x$, the torsion angle $\tau_v$ for the joint $v$ in this realization is the **clockwise** angle (in the range $[0, 360°]$) formed by the plane defined by $x((p \circ p \circ p)(v))$, $x((p \circ p)(v))$ and $x(p(v))$, and the plane defined by $x((p \circ p)(v))$, $x(p(v))$ and $x(v)$.*

When a realization $x$ preserves the morphology of the character, we can use this pair of angles combined with the bone lengths (defined by $\chi$) to find the Cartesian coordinates of any joint $v$ that has at least three ancestors. The two angles are illustrated in Figure 4.7 Note that we must use the *clockwise* and not the smallest angle for $\tau_v$ so that $\tau_v$ is in the $[0, 360°]$ range. In fact, we see in Figure 4.7 that the vector-torsion angle pairs $(90°, 90°)$ (middle) and $(90°, 270°)$ (right) would not be distinguishable if we instead used the *smallest* angle for $\tau_v$. Using these angles, it is clear to see that the combination of $\zeta_v = 90°$ and $\tau_v = 270°$ leads to an unnatural position for this joint.

Not all joints $v \in V$ have the required number of three ancestors. Let $R \subset V$ be the joints with fewer than three ancestors. Every joint $r \in R$ must be represented using the triplet of Euler angles, leading to a hybrid representation:

$$\rho' : (v, t) \in V \setminus \{v_0\} \times T \longrightarrow \begin{cases} \left(\theta^t_{p(v),v}, \phi^t_{p(v),v}, \eta^t_{p(v),v}\right) \in [0, 2\pi)^3, & \text{if } v \in R, \\ (\zeta^t_v, \tau_v) \in [0, 2\pi)^2, & \text{if } v \in V \setminus R. \end{cases}$$

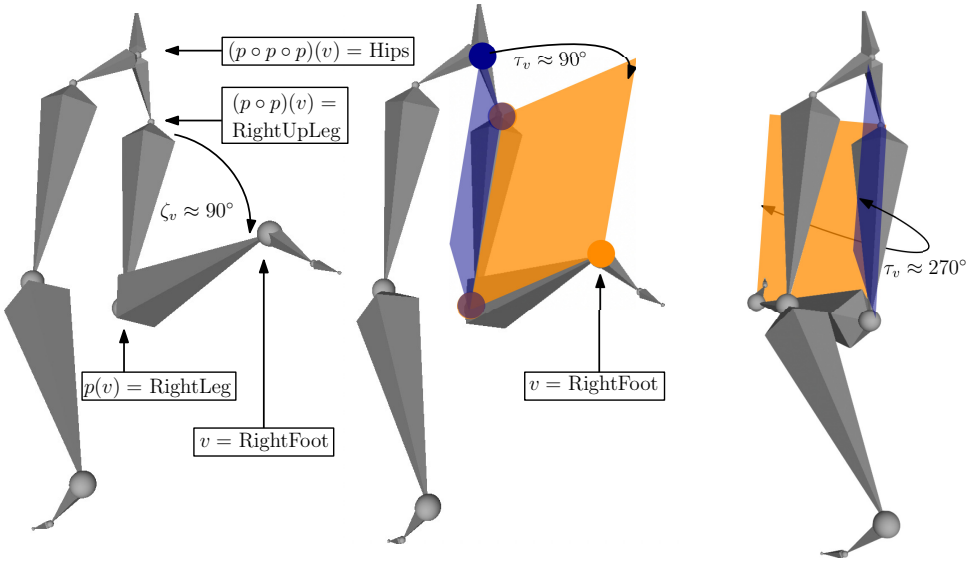



Figure 4.7 – An example of $\zeta_v$ and $\tau_v$. Visible are the legs of a human skeleton, taken from a BVH file describing a running motion. The left side illustrates the vector angle $\zeta_v$, which is the angle between the RightFoot, the RightLeg, and the RightUpLeg joints. The center shows how we arrive at the torsion angle $\tau_v$ using two planes defined by RightFoot and its three ancestors RightLeg, RightUpLeg, and Hips (or RightHips). On the right, we see what happens when $\tau_v = 270°$, while $\zeta_v$ remains 90°.

The value that the vector-torsion representation has to offer for representing motions is that it lets us avoid several of the drawbacks of the Euler representation $\rho$ that were mentioned before. Namely, we see the following advantages:

— The combination of a vector angle with a torsion angle cannot lead to any representation singularities

— There exists a bijective correspondence between the value of the angles and the positions in space for the joints

— It exhibits only two degrees of freedom for encoding the same information comprised in a triplet of Euler angles

— It allows us to empirically constrain the feasible (and mostly continuous!) regions in the vector-torsion angle space where only natural postures for the human skeleton can be found.

The last point is of increased importance in the context of motion retargeting, as we will see in the next section.

### 4.4.2 Motion analysis

In this section, we present an analysis of the Graphics Lab Motion Capture Database, which is the same database that we used in the experiments discussed in Section 4.3. The database contains 2436 motion files which sum to a total of more than four million frames. In the following, we will refer to human joints with labels such as Hips, RightShoulder, and LeftLeg, which are taken directly from the BVH files in the database. Using the vector-torsion representation we conducted a statistical analysis of the vector angles $\zeta_v^t$ and torsion angles $\tau_v^t$ of every applicable joint $v$ at every frame $t$ of these four million frames. Using the resulting data from this experiment, we generated a heat-map scatter plot for each joint $v$. This analysis, just like the representation itself, also has its inspiration in structural biology. The plots that we will present are similar to the well-known Ramachandran plots, which analyze combinations of the $\phi$ and $\psi$ torsion angles in proteins.

Even though it is not possible to define the vector and torsion angles for the joints having fewer than three ancestors (see previous section) we performed the analysis on these joints as well. There is no need in the analysis to build up human postures, but only to *look at* the four million postures contained in the database. Therefore, for the joints missing a sufficient number of ancestors, we have simply defined a different set of "reference joints" in the graph $G$, that we subsequently use for defining the vector and torsion angles. For example, Figure 4.8 shows how we can still compute the vector-torsion angle pair for the RightLeg and LeftUpLeg, which have only two and one ancestor(s) respectively. Throughout the analysis, for joints such as these, we make sure to always use the same combination of joints when computing the angles, so that we obtain consistent results. Joints $v$ with $|\chi_v| = 0$ are omitted and they are not counted as ancestors of other joints either. For example, the LHipsJoint and the RHipsJoint (see Figure 4.1) are placed in the same position as the Hips joint, and the associated Euler angles are supposed to define the orientation for all joints that form the left and right leg, respectively. The vector-torsion representation does not use these particular joints because the two angles cannot be computed for them. Table 4.1 shows the complete set of joints for which we performed our analysis, together with the list of three reference joints used for computing the vector and the torsion angles.

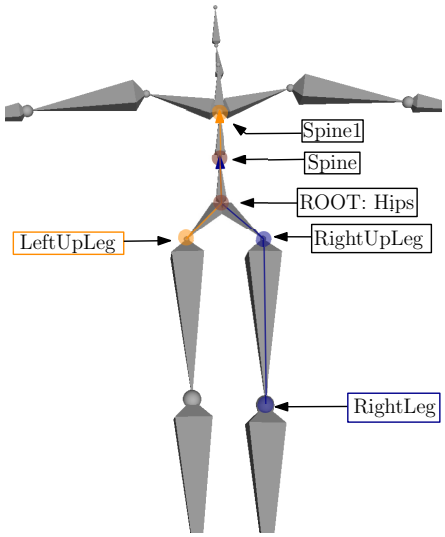



Figure 4.8 – The RightLeg and the LeftUpLeg are two examples of joints that do not have enough ancestors to be represented by the vector-torsion angle pair. We can still use other consecutive reference vertices in the anatomy $H$ to compute these angles and perform the same analysis for these joints. In the figure, we show the two sets of joints used to define the angles for the RightLeg and the LeftUpLeg. For the former, we used the joints RightUpLeg, Hips, and Spine (the dark blue path). For the latter, we used the joints Hips, Spine, and Spine1 (the orange path).

| *joint* | *ref#1* | *ref#2* | *ref#3* | all ancestors? |
|---|---|---|---|---|
| Head | Neck | Spine1 | Spine | Yes |
| Neck | Spine1 | Spine | Hips | Yes |
| Spine1 | Spine | Hips | RightUpLeg | No |
| Spine | Hips | RightUpleg | RightLeg | No |
| LeftArm | Spine1 | Spine | Hips | Yes |
| RightArm | Spine1 | Spine | Hips | Yes |
| LeftForeArm | LeftArm | Spine1 | Spine | Yes |
| RightForeArm | RightArm | Spine1 | Spine | Yes |
| LeftHand | LeftForeArm | LeftArm | Spine1 | Yes |
| RightHand | RightForeArm | RightArm | Spine1 | Yes |
| LeftUpLeg | Hips | Spine | Spine1 | No |
| RightUpLeg | Hips | Spine | Spine1 | No |
| LeftLeg | LeftUpLeg | Hips | Spine | No |
| RightLeg | RightUpLeg | Hips | Spine | No |
| LeftFoot | LeftLeg | LeftUpLeg | Hips | Yes |
| RightFoot | RightLeg | RightUpLeg | Hips | Yes |
| LeftToeBase | LeftFoot | LeftLeg | LeftUpLeg | Yes |
| RightToeBase | RightFoot | RightLeg | RightUpLeg | Yes |

Table 4.1 – The three reference joints for each joint that was involved in the analysis. The reference joint with the smallest numerical label is the closest; the one with the largest numerical label is instead the farthest. In some cases, the reference joints are not the joint ancestors given by the graph structure.

We will present 18 scatter plots for different joints, starting at the head of the skeleton, and working our way down to the feet. The vector angles are shown on the $x$-axis and the torsion angles on the $y$-axis. In these plots, points tending to the warmer colors correspond to pairs of angles that were found more frequently.

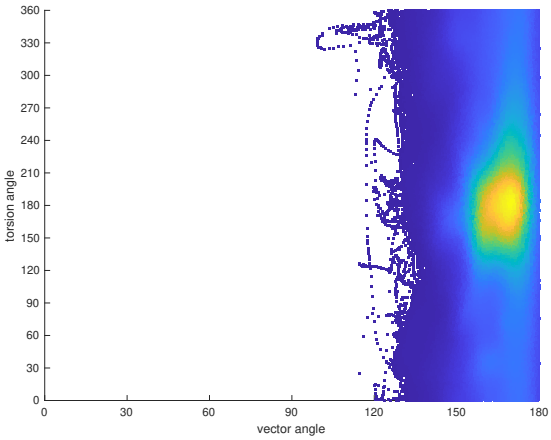



Figure 4.9 – The plot for the Head.

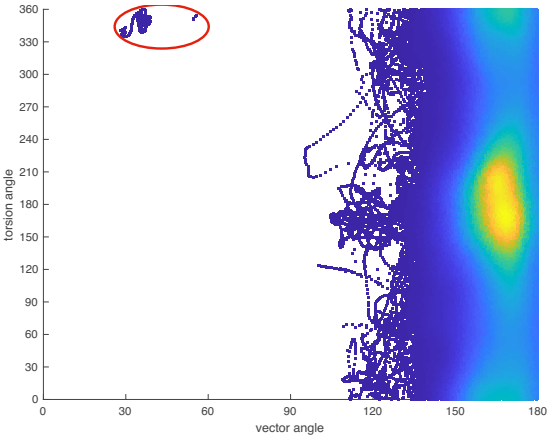



Figure 4.10 – The plot for the Neck. The encircled points correspond to the unnatural position seen on the right in Figure 4.27

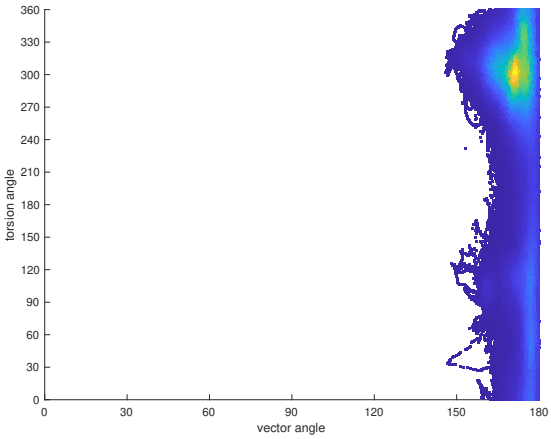



Figure 4.11 – The plot for the Spine1.

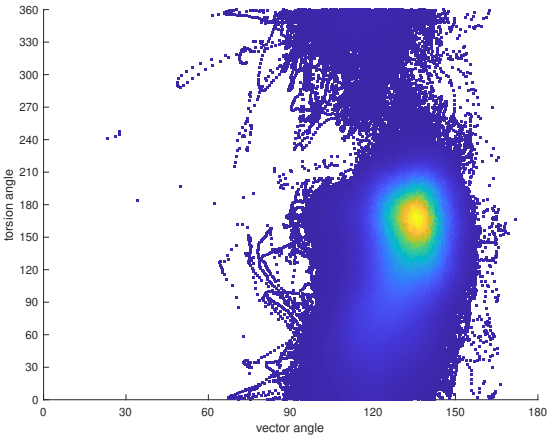



Figure 4.12 – The plot for the Spine.

Figure 4.9 shows the scatter plot for the human head. As expected, this joint is not able to perform much movement in the space defined by the vector and torsion angle. Roughly speaking, only one-third of this space is feasible for the head joint. Moreover, the warmer part of the scatter plot indicates that the most common posture for this joint is when the two angles are close to 180°, which is compatible with an erected posture for the upper body part. While the Neck joint (Figure 4.10) exhibits a pattern very similar

to the one of the Head joint, we notice that the two joints involved in the modeling of the human spine (see Figure 4.11 and 4.12) admit an even smaller feasible space. This is particularly true for the Spine1 joint, where a large part of the scatter plot remained white which means that the combinations of vector and torsion angles in these areas are completely infeasible for a human spine.

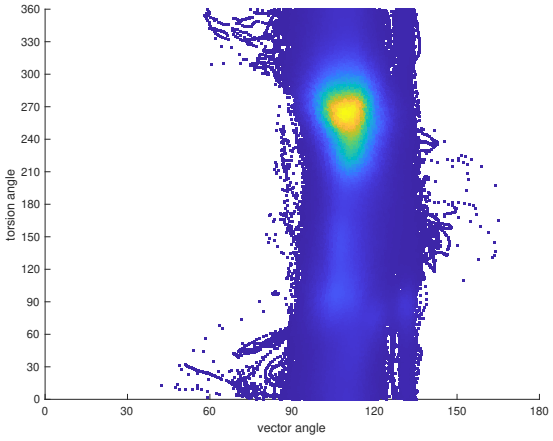



Figure 4.13 – The plot for the LeftArm.

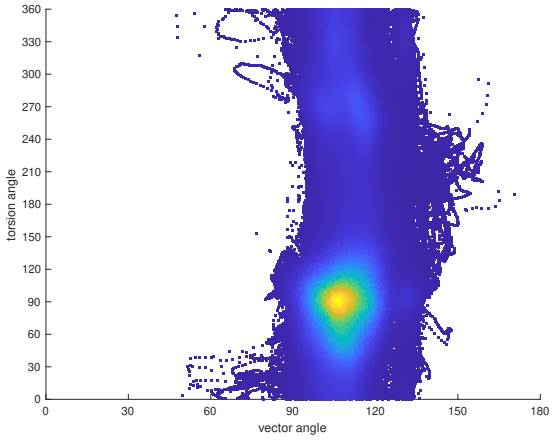



Figure 4.14 – The plot for the RightArm.

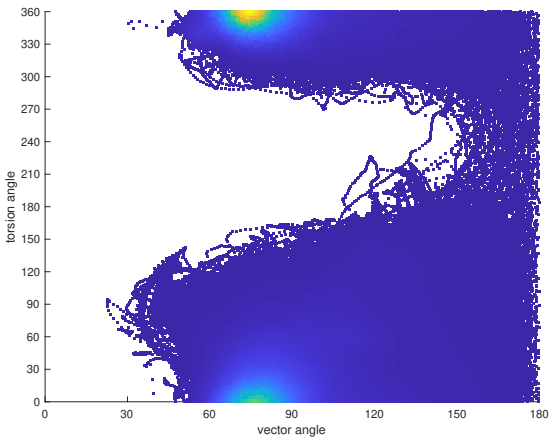



Figure 4.15 – The plot for the LeftForeArm.

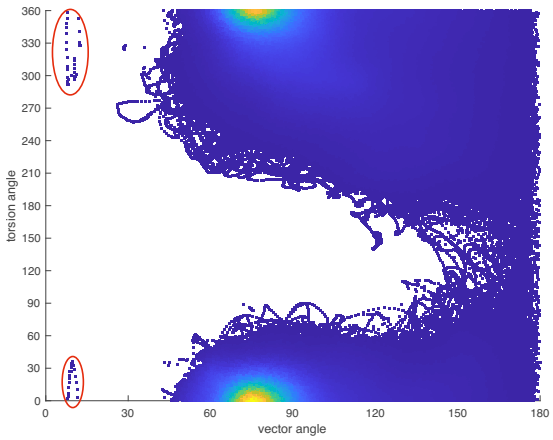



Figure 4.16 – The plot for the RightForeArm. The encircled points correspond to the unnatural position seen on the right in Figure 4.27.

The plots for the LeftArm ( Figure 4.13) and RightArm ( Figure 4.14) show a quite constrained pattern as well, which is similar to those found for some of the previous joints but shifted in the center of the vector angle axis (the $x$-axis). This means that the angle formed by the spine and one of these two joints is often close to 90°. In fact, we see that these two joints share the same global Cartesian positions with the LeftShoulder

and RightShoulder joints. We can also remark that the two scatter plots are symmetric w.r.t. the axis parallel to the $x$-axis and passing through the torsion angle value 180°. The expected flexibility for the human arms is reflected in the scatter plots in Figure 4.15 and 4.16, related to the LeftForeArm and RightForeArm joints, respectively. This is the first pair of joints so far for which the "coloured areas" can cover more than 50% of the two-dimensional space. Yet, there are still particular combinations of vector and torsion angles that correspond to unnatural postures. As discussed in more detail later, the outliers found in the scatter plot concerning the RightForeArm correspond to errors that we have identified in the motion capture database. Again, the two scatter plots for the two forearms are symmetric with one another.

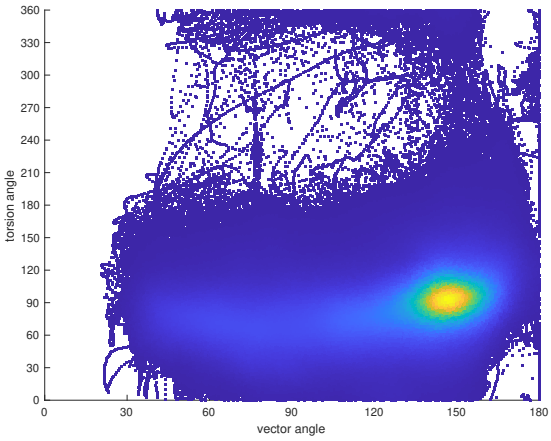



Figure 4.17 – The plot for the LeftHand.

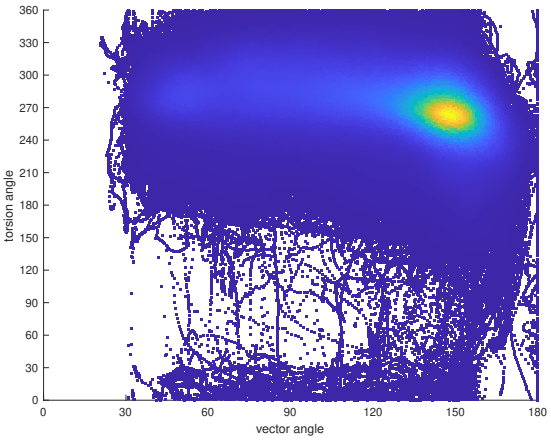



Figure 4.18 – The plot for the RightHand.

Similarly, Figure 4.17 and 4.18, depicting the scatter plots related to the LeftHand and RightHand joints respectively, show a quite large range of movement possibilities for these joints. This was expected as well. Moreover, the sparsely populated blue areas in these two scatter plots seem to suggest the extremely high flexibility of the human hand: even if sometimes very uncommon (a few frames of the database may contain them), there exist very special (and still natural) postures that the human hand can take. Therefore, if we take into consideration in full these low-populated areas, we can state that the scatter plots related to the human hands are the ones that almost cover the entire two-dimensional space.

When stepping down over the joints forming the human legs, we can observe similar patterns. For the upper joints in Figure 4.19 and 4.20, concerning the LeftUpLeg and the RightUpLeg joints, respectively, we can notice that the patterns are similar to those observed for the two upper arm bones. Similarly, we have the pattern of the LeftLeg

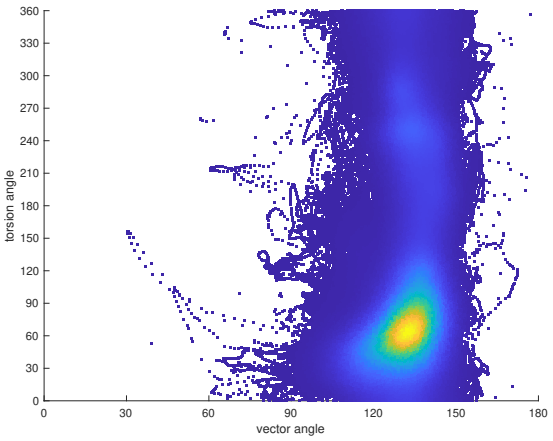



Figure 4.19 – The plot for the LeftUpLeg.

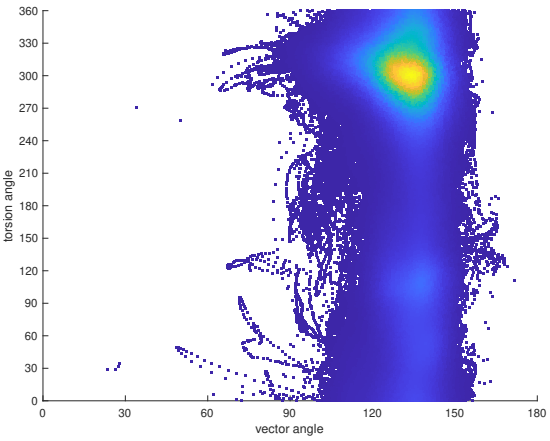



Figure 4.20 – The plot for the RightUpLeg.

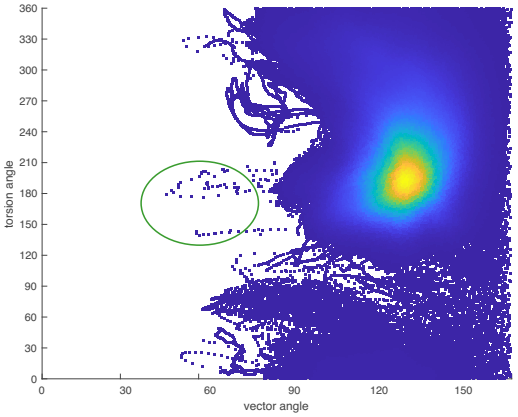



Figure 4.21 – The plot for the LeftLeg joint. The points in the green circle correspond to the uncommon poses in Figure 4.28.

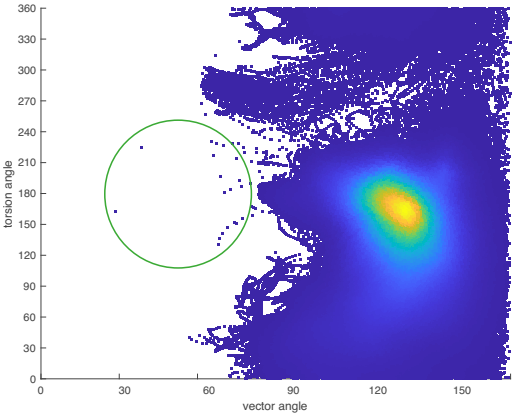



Figure 4.22 – The plot for the RightLeg joint. The points in the green circle correspond to the uncommon poses in Figure 4.28.

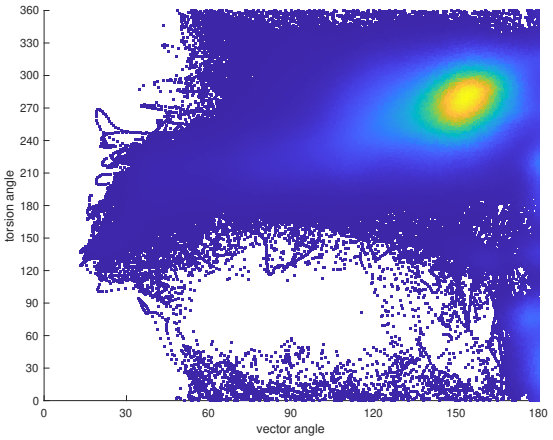



Figure 4.23 – The plot for the LeftFoot.

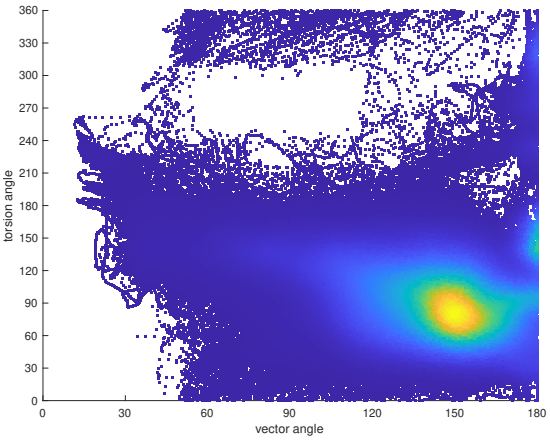



Figure 4.24 – The plot for the RightFoot.

joint in Figure 4.21, and the pattern of the RightLeg joint in Figure 4.22. The human feet also exhibit large movement possibilities, as we have observed for the hands, but the low-populated areas in the scatter plots for the feet are much more sparse. This can be a consequence of the fact that the human feet lost, during evolution, part of their movement possibilities, but the similarity to the hands is still visible in our figures. The scatter plot for the LeftFoot joint is in Figure 4.23; the scatter plot for the RightFoot joint is in Figure 4.24.

Finally, the LeftToeBase and RightToeBase joints, in Figure 4.25 and 4.26 respectively, show that they are the only joints that can span the entire two-dimensional space, but the region is not continuous and most of the vector-torsion combinations are placed around the center of the plot, where $\zeta_v = 90°$ and $\tau_v = 180°$.

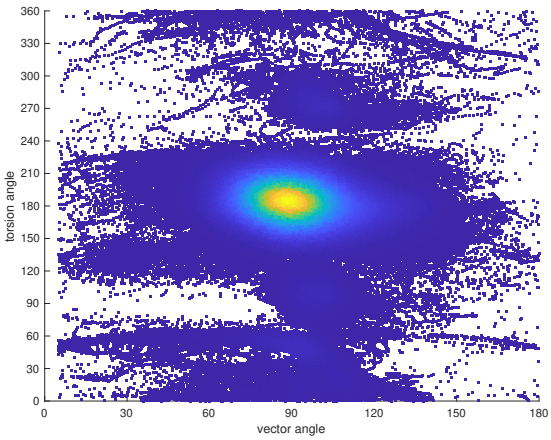



Figure 4.25 – The plot for the LeftToeBase.

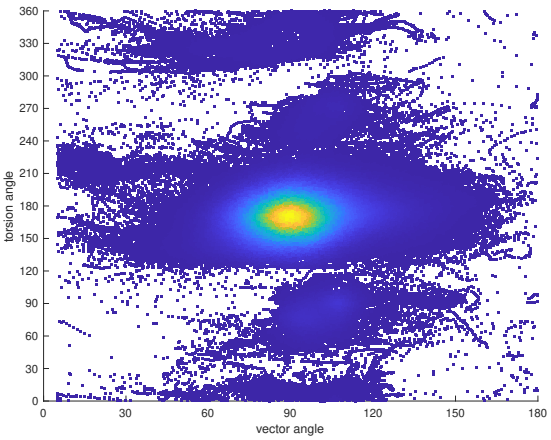



Figure 4.26 – The plot for the RightToeBase.

To sum up, the analysis shows that there exist large differences in flexibility between different joints, and the feasible regions tend to vary a lot based on the nature of each joint. In general, the vector angle seems to be the most restrictive factor. For joints like the Head, Neck, and Spine joints, we see that the feasible vector angle regions are around the 180° mark and do not vary much. The plots for these joints are quite similar, and this makes sense when we look at their respective ancestor joints in the skeleton (see Figure 4.1) Joints on the right and left sides of the human body have very comparable regions, except for the fact that the values of the torsion angles are generally inverted. This is a result of the fact that we use a clockwise rotation to compute the torsion angles between the two planes defined by the quadruplets of joints. This gives rise to the symmetry property mentioned above. Various of the white regions in these scatter plots appear to be in logical positions intuitively. For instance, the example of the inverted knee (the vector-torsion

angle pair $(90°, 270°)$) in Figure 4.7 can be found in a (small) white region in the plot the RightFoot joint.

One observation from the plots is that there exist various data points that appear to be outside the "warmer", more frequent regions. We looked at several of these deviating data points and classified them into two categories: motion capture errors and uncommon postures. The motions containing motion capture errors should be eventually excluded from the data sets when using the results of the analysis. We provide some examples for each of these categories and show that it is rather uncomplicated to distinguish between the two. Figure 4.27 shows two instances of an erroneous pose in a human motion from the database, where the angles lie far outside the normal ranges. Finally, we showcase two examples of outliers that correspond to possible, yet rare, human poses. The examples affect vector-torsion angles in at least two joints: LeftLeg and RightLeg. Figure 4.21 and 4.22 show the two plots affected by these postures, with the outlying points encircled in green. What these two poses have in common, is that for both the left and right sides of the skeleton, the vector angles (between the Leg, UpLeg, and Hips joints) are quite small. While the figure only shows two examples, there are many similar cases, for example in the scatter plot of the UpLeg joints, there are some outlying data points where the vector angles are smaller than 70°. These, and other data points, do correspond to valid human postures.

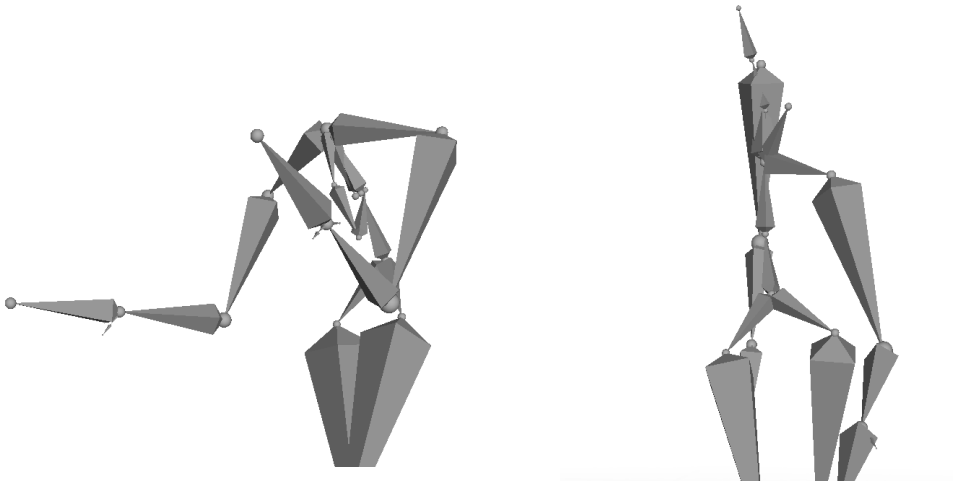


Figure 4.27 – The two motion capture errors. On the left, we see a bug for the Neck (motion: #138_31, frames: 70–688). The corresponding points in the scatter plot are encircled in red in Figure 4.10. On the right, we see an error for the RightForeArm (motion: #143_38, frames: 625–660). The corresponding points in the scatter plot are encircled in red in Figure 4.16

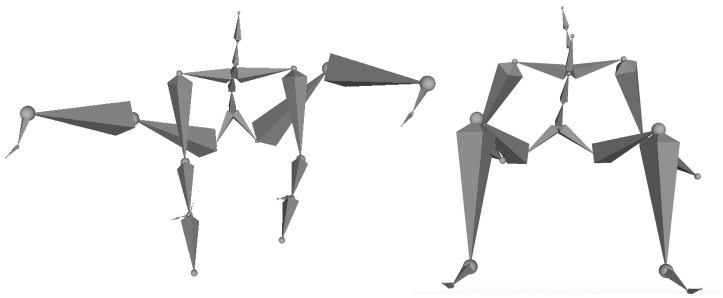


Figure 4.28 – Examples of outlying data points that correspond to valid human postures. Left: a frame of the character landing after a wide-legged jump (motion: #121_19, frame: 847). Right: a frame of the character in a sitting position (motion: #111_09, frame: 357).

## 4.5 Discussion & Future work

In this chapter, we discussed several new results in the context of the Dynamical Distance Geometry Problem. In particular, we focused on the application of human motions and motion retargeting. We expanded on a distance-based method for motion retargeting, including new inter-frame distances, priorities based on the interaction distances between joints, and optimization in Euler space. Next, we looked at a new representation for human motions, inspired by the vector and torsion angles from structural biology. This representation has fewer degrees of freedom than the standard representation used in the literature. Furthermore, it allowed for a detailed analysis of a large database of human motions. This analysis lets us identify natural positions for the different joints in the human body.

This leads directly to the first area of future work. The results from this analysis can be used for motion retargeting. The constraints that we can obtain from the scatter plots may be used during the optimization, making sure that our retargeted motions will never contain the character taking an unnatural position. Furthermore, the results from the analysis may be used to generate random motions that look natural. Next, these results from the research on human motions should be applied to simulating protein dynamics and animating conformational samplings of protein structures, which was one of the main sources of our interest in the topic.

Chapter 5

# Adaptive Maps & Linear Programming

In the context of molecular biology, we saw that not only including distances but also torsion angles in the DGP instances can be beneficial. In this chapter, we will look at the application of adaptive maps, where we generate maps where the distances between the Points of Interest (POIs) on the map are not Euclidean but instead based on subjective measures of proximity, which depend on the user. One of the challenges when generating adaptive maps is to move the POIs in a way such that these *subjective distances* are satisfied while making sure the new map still resembles the old, general-purpose map. We will see that this problem of POI relocation can be formulated as a type of DGP, where we include constraints based on local orientations. These same constraints allow us to linearise the problem, making Linear Programming suitable for solving the problem.

We start with an introduction in Section 5.1. Next, in Section 5.2 we present a linear program (LP) which can solve the problem of POI relocation for adaptive maps. Computational experiments with this new LP are presented in Section 5.3. In Section 5.4 we present preliminary work on a method that exhaustively performs virtual axes rotations to find the combination of local orientations that lead to the best objective value. Finally, in Section 5.4 we end with some discussions.

## 5.1 Introduction

Geographical maps have been in use by human civilizations for centuries, and are widely used in our everyday life. Early maps in history range from drawings etched on cave walls to extensive maps produced by ancient empires such as Babylon, Greece, and Rome [211]. These historic maps as well as most commonly used modern maps are topographic, which, once we have a 2D map, means that the relation between depicted elements is based on their Euclidean distance. A key point of these maps is that they are general-purpose, i.e., they are meant for widespread public use. In contrast to such a general-purpose map, an *adaptive map* is a map that is modified based on selected features that matter to a specific user [20, 21]. An example of this is a touristic map fine-tuned for pedestrians who have trouble climbing up or going down steep slopes. These users would much prefer routes between the POIs with low inclinations and would like to have a map capable of reflecting their preferences visually. Such an adaptive map can be created using *subjective distances* between a set of Points of Interest (POIs) on the original topographical map. The first reference to this type of map seems to be in 1983 [212] and in [213], where the idea was applied for generating metro maps. More recent works include [20, 21], which coined the term adaptive maps. In these works, the generation of adaptive maps was separated into three steps:

— Identifying the POIs and generating subjective distances between them

— Relocating POIs based on the subjective distances

— Modifying the original map so that the new POI locations are reflected

For this thesis, we focused only on the second step, relocating the POIs, while attempting to best satisfy the subjective distances. The input to the problem is the output of step 1: the two-dimensional positions of the POIs in the original general-purpose map and the pairwise subjective distances.

We can reformulate this problem as a type of Distance Geometry Problem. We have a graph $G = (V, E, d)$, where the vertices correspond to the POIs and the edges and their weights relate to the subjective proximities. If we were to solve the POI relocation problem using only this information, there is a chance that the given subjective distances will force the POIs to move in such a way that their (relative) output positions will not at all match their corresponding positions in the original map. This would make the third step of modifying the original map very difficult. Therefore, we would like the set of output POI positions after relocation to be topologically homeomorphic to the original

map. By using the positions of the POIs in the original map (which can be seen as an initial realization $x_0$), we can include extra information in our instances: local orientations. As previously described in Section 1.3, these are defined using four standard quadrants (NW, NE, SW, and SE). In order to include these orientations, $G$ must be a directed graph. We can then partition our edge set $E$ into the four quadrants. For example, when we have $(u, v) \in$ NW, it means that $v$ is to the *North-West* of $u$ in the original map. Of course, the information about the orientations should remain symmetric, i.e., if we have $(u, v) \in$ NW, we must have $(v, u) \in$ SE. Note that if a pair of POIs $(u, v)$ have the same $x$ or $y$-coordinate, $(u, v)$ will be in two of these quadrants (i.e., NW and NE), which is not a problem for our definition or our methods. When a realization $x$ satisfies all the local orientations, we say that $x$ is topologically homeomorphic to $x_0$. Using this extra information, we can define POI relocation as a variant of the optimization based DGP (see also [214]):

**Definition 11** *Given a simple weighted directed graph G = (V,E,d) and an initial realization $x_0$ of G, where*

- *V is a set of POIs*
- *E represents the presence of proximity information and is partitioned into the four subsets (NW, NE, SW, and SE) to encode the local orientations given in $x_0$*
- *the weight function d provides the numerical values $\delta$ for the proximity measures*
- *For simplicity, $x_v$ is the x-coordinate and $y_v$ is the y-coordinate of POI v in a realization x*

*find a realization $x : v \longrightarrow \mathcal{R}^2$ such that a penalty function $\sigma$ is minimized,*

$$s.t. : \begin{cases} \forall (u,v) \in NW, & x_u \geq x_v \quad and \quad y_u \leq y_v, \\ \forall (u,v) \in NE, & x_u \leq x_v \quad and \quad y_u \leq y_v, \\ \forall (u,v) \in SW, & x_u \geq x_v \quad and \quad y_u \geq y_v, \\ \forall (u,v) \in SE, & x_u \leq x_v \quad and \quad y_u \geq y_v, \end{cases} \tag{5.1}$$

Because we are dealing with maps in dimension 2, $\sigma$ (see Equation (1.4)) can be defined as:

$$\sigma(x) = \sum_{(u,v)\in E} \left( \sqrt{(x_v - x_u)^2 + (y_v - y_u)^2} - \delta(u,v) \right)^2. \tag{5.2}$$

Note that the definition of the graph $G$ allows the proximity measures $\delta(u, v)$ and $\delta(v, u)$ to be different because $G$ is directed. However, we will assume that the distance information in our instances is symmetric.

Using this definition, the POI relocation problem can be seen as a particular DGP subclass of instances, where the additional constraints on the relative orientations are enforced. In general, adding constraints to a known problem is likely to increase its complexity. In our case, while the POI relocation problem remains in the same complexity class of the generic DGP, these new constraints as well as the use of a linearizable distance measure will allow us to formulate the POI relocation problem as a linear program (LP) consisting of only real variables.

## 5.2 A linear program for POI relocation

As discussed in Section 1.4.6, Linear Programming may be used for general Distance Geometry Problems when we substitute the Euclidean distance norm with a linear one. In [31], the $L_\infty$ (see Equation (1.11)) norm was used and binary variables were introduced. In [32], both the $L_\infty$ and $L_1$ (see Equation (1.12)) norms were employed. However, this came at the price of adding binary variables and non-linear constraints which means that the complexity of the resulting algorithm is non-polynomial. Instead, in the case of the POI relocation problem as defined in Definition 11, we will see that the addition of the relative orientation constraints allows us to use continuous variables only.

We will substitute the Euclidean distance norm by using the $L_1$ norm. Because the dimension $K = 2$, the $L_1$ norm (also known as taxi or Manhattan distance) looks like this:

$$L_1(u, v) = |x_v - x_u| + |y_v - y_u|, \tag{5.3}$$

where the symbol $|\cdot|$ represents the absolute value of a real number. We use this distance norm to define auxiliary variables $z_{uv}$. The linear model that we propose in [214] aims to minimize the sum of these auxiliary variables:

$$\min \sum_{(u,v)\in E} z_{uv}, \tag{5.4}$$

where $z_{uv}$ is an upper bound for the absolute difference between the $L_1$ distance in our

incumbent solution $x$ and the proximity information $\delta$ in the input:

$$z_{uv} \geq ||x_v - x_u| + |y_v - y_u| - \delta(u, v)|, \tag{5.5}$$

Absolute values of computations can be problematic for LP solvers to deal with because their inclusion can cause the optimization problem to be non-convex and non-smooth. However, the relative orientations that we have in the input actually allow us to avoid the need for absolute values. This can be done by including two constraints on $z_{uv}$ for every quadrant. For each quadrant, the combination of these two constraints is equivalent to the inequality in Equation (5.5):

$$\begin{cases} \forall (u,v) \in \text{NW}, & z_{uv} \geq y_v - y_u + x_u - x_v - \delta(u,v) \\ \forall (u,v) \in \text{NW}, & z_{uv} \geq \delta(u,v) - y_v + y_u - x_u + x_v \\ \forall (u,v) \in \text{NE}, & z_{uv} \geq y_v - y_u + x_v - x_u - \delta(u,v) \\ \forall (u,v) \in \text{NE}, & z_{uv} \geq \delta(u,v) - y_v + y_u - x_v + x_u \\ \forall (u,v) \in \text{SW}, & z_{uv} \geq y_u - y_v + x_u - x_v - \delta(u,v) \\ \forall (u,v) \in \text{SW}, & z_{uv} \geq \delta(u,v) - y_u + y_v - x_u + x_v \\ \forall (u,v) \in \text{SE}, & z_{uv} \geq y_u - y_v + x_v - x_u - \delta(u,v) \\ \forall (u,v) \in \text{SE}, & z_{uv} \geq \delta(u,v) - y_u + y_v - x_v + x_u \end{cases} \tag{5.6}$$

The objective function (5.4), the constraints (5.6), and the relative orientation constraints (5.1) together define the linear model for our problem.

### 5.2.1 Implementing the model

The model was implemented into Java, using the LP solver CPLEX [120], which has an interface for Java using the Concert Technology. The code for the model is found in the LinearSolver class in the Java package (see Figure 1.4). In the model, we have two variables $x_v$ and $y_v$ for every POI $v$. Since we require that $\delta(u, v) = \delta(v, u)$, we only need to use one auxiliary variable $z_{uv}$ for the edges $\{u, v\}$ and $\{v, u\}$. This means the total number of variables in the model is $2|V| + \frac{1}{2}|E|$.

For every variable $z_{uv}$ we have two constraints, which means there is a total of $|E|$ of such constraints. In a naive implementation of the model, one could include a relative orientation constraint for every pair of vertices $(u, v)$. However, this can be improved if

we use two vertex orders $O_x$ and $O_y$. In the first order $O_x$, the POIs are sorted ascending by their $x$-coordinate in the original map $x_0$. In $O_y$, they are ordered ascending by their $y$-coordinate. Using these orders, we can simplify the relative orientation constraints. For a vertex $v$, let $u$ be its successor in $O_x$ and $w$ be its successor in $O_y$. If we then include the two constraints $v_x \leq u_x$ and $v_y \leq w_y$ for all the vertices $v$, (as long as the successor $u$ or $w$ exists), we only need $2(|V| - 1)$ constraints to cover all relative orientations. This leads to a total of $2(|V| - 1) + |E|$ constraints in the model.

The model at hand may be solved either by primal or dual simplex methods, and both can be used when employing the solver CPLEX. The model has more constraints than variables, which means that the dual solver should be more efficient (as per the CPLEX manual [120]). Therefore, we used the dual simplex method for the experiments in the next two sections.

## 5.3 Experiments: the city of Granada

To test the model and its implementation, we used a 10-point POI relocation instance available from [21]. These 10 POIs, and their initial coordinates (listed in Table 2 in [21]), are based on real Points of Interest in the city of Granada in Spain. These experiments were published in [214]. The subjective proximity information $\delta(u, v)$ between these POIs instances are shown in Table 5.1, which are based on walking distance. The walking distance is likely to differ from the Euclidean distance, while paths along streets resemble more L1 norm distances. To present the data visually, Figure 5.1 shows a colored map depicting the POI distance variations (when comparing the original Euclidean distances and the given proximity distances) in the instance, as well as the local orientations of the POIs. As mentioned in the previous section, we used IBM's CPLEX solver to obtain solutions to the implemented model. For this 10-point instance, the result was obtained in 0.072 seconds on a MacOS (Ventura), 32GB RAM, and with CPU Apple M1 Max (10 cores). In order to superimpose the obtained coordinates with the original map, we first center both point sets at the origin by subtracting the centroid from each point. Then, we scale both point sets so that their distances are within the same range. The obtained result, which has an objective function value equal to 10273, is shown in Figure 5.2. It is not trivial to visually validate the result. We will do this by contrasting the heatmap given in Figure 5.1 with the displacement of the POIs in our solution.

| | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
|---|---|---|---|---|---|---|---|---|---|---|
| P1 | 0 | 1553 | 2783 | 3614 | 4240 | 5674 | 6972 | 6064 | 7408 | 5387 |
| P2 | 1553 | 0 | 1674 | 3867 | 2975 | 5936 | 7234 | 6326 | 6807 | 4693 |
| P3 | 2783 | 1674 | 0 | 3427 | 2118 | 5098 | 6396 | 5409 | 5950 | 3837 |
| P4 | 3614 | 3867 | 3427 | 0 | 2471 | 2571 | 4609 | 3745 | 5222 | 3292 |
| P5 | 4240 | 2975 | 2118 | 2471 | 0 | 3600 | 4684 | 3172 | 3840 | 1727 |
| P6 | 5674 | 5936 | 5098 | 2571 | 3600 | 0 | 2569 | 3076 | 5083 | 3416 |
| P7 | 6972 | 7234 | 6396 | 4609 | 4684 | 2569 | 0 | 2692 | 4966 | 3903 |
| P8 | 6064 | 6326 | 5409 | 3745 | 3172 | 3076 | 2692 | 0 | 2415 | 1698 |
| P9 | 7408 | 6807 | 5950 | 5222 | 3840 | 5083 | 4966 | 2415 | 0 | 2303 |
| P10 | 5387 | 4693 | 3837 | 3292 | 1727 | 3416 | 3903 | 1698 | 2303 | 0 |

Table 5.1 – The walking distances between 10 pairs of POIs, that we use in our computational experiment. Reproduced from [21] (Table A1)

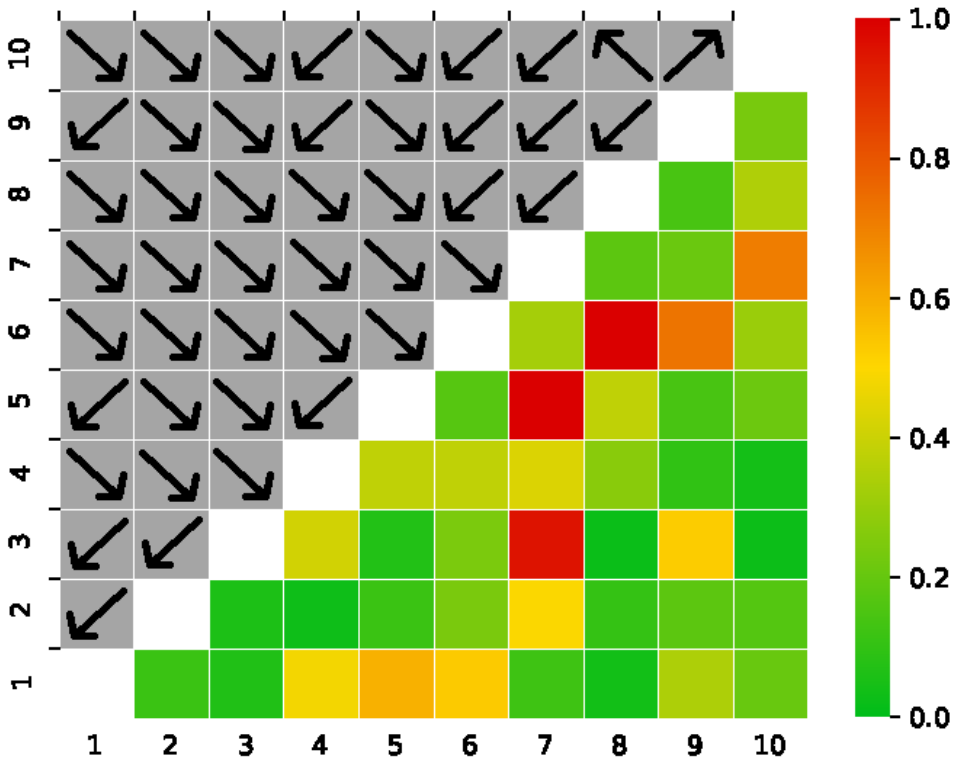



Figure 5.1 – Information related to the 10-point instance in [21]. Top left: a visual representation of the local orientations of the POIs in the original realization, which our linear model is supposed to preserve in the found solution. Bottom right: a heatmap, where the colors encode the difference in values between the Euclidean distances (in the original realization $x_0$) and the given proximity distances (walking distances, see Table 5.1). The lower the difference, the smaller the value attributed to the corresponding square in the heatmap; all difference values are normalized between 0 and 1.

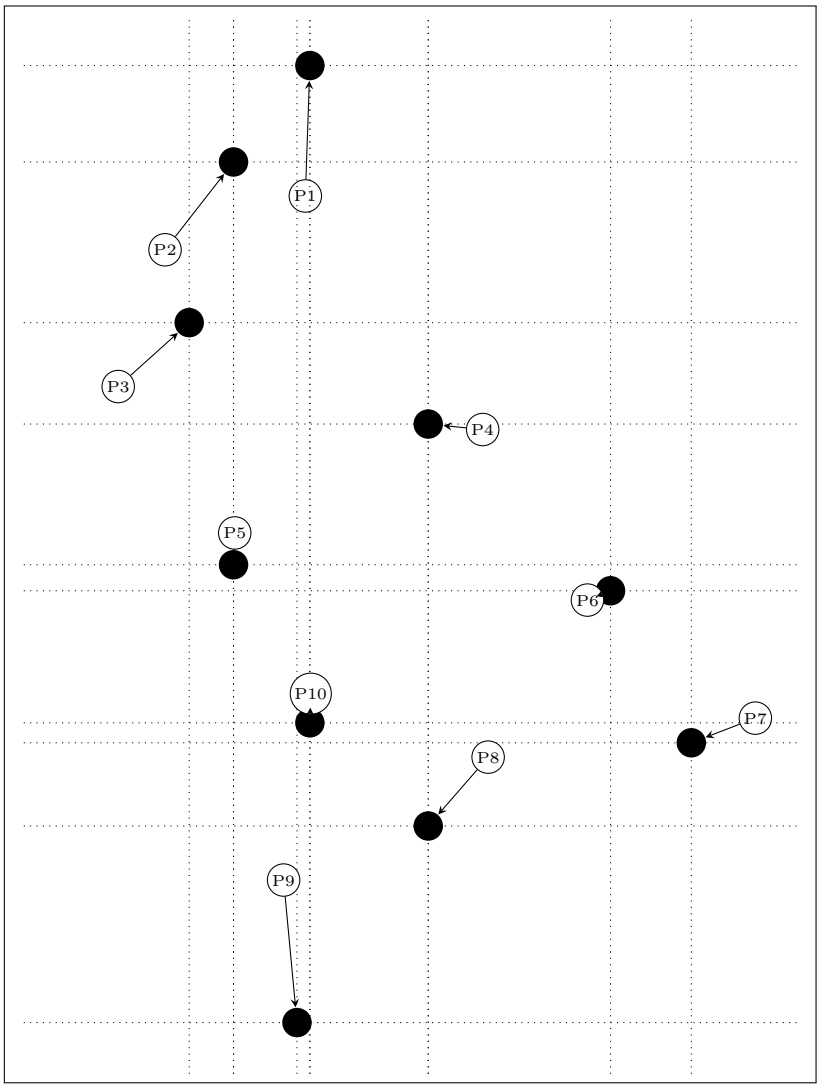



Figure 5.2 – The solution found by our linear model to the POI relocation problem related to data in Table 5.1. The points $P_*$ indicate the original locations of the POI in the initial map. In black, we see the new positions for the POIs.

Let us first look at the pair P6 and P8, which correspond to a red area in our heatmap. This means that there is a large difference between the walking distance given and the Euclidean distance in $x_0$. The distance between these two POIs is supposed to get larger in the adaptive map, and we can notice that this happens in our solution. We can remark, however, that the locations of some other POIs are subject to even larger modifications, apparently contradicting our input data. For example, P9 is moved a lot, while in the heatmap P9 seems to be associated with more green squares (lower differences between walking distances and Euclidean distances). However, such POIs are actually displaced in

other to to maintain their relative distances to other POIs which were moved to account for the more red areas in the heatmap. For example, because P8 is moved South because of the "red" edge {P6,P8}, P9 has to be moved South accordingly. Let us now consider the point P3. Two of the edges related to P3 are "red" indicating a quite important variation in the corresponding relative distances. However, the location of P3 does not seem to have changed much in our adaptive map. This behavior is most likely due to the introduced orientation constraints. In fact, P7 belongs to the SE quadrant of P10, and the little movement it is allowed to take seems to be the consequence of the fact that P10 moved a little in the South direction. Any other movement for P7 in the allowed quadrant would have most likely caused larger distance penalties with its closest neighbors. In general, we can observe that all local orientations are well respected.

## 5.4 Rotating the axes

In this section, we will present some ongoing work on the problem of POI relocation. This work and the experimental results have not been published. However, we are preparing a journal publication based on these preliminary results and experiments, pending some theoretical work.

The directions of the axes (North-South and East-West) that we used in the previous sections are based on the directions of the axes in the original map. To obtain a linear program these directions need to be fixed so that we may use them in the relative position constraints, which allows for the linear calculation of the $L_1$ distances. However, it may be observed that we are not required to use these specific directions of the axes. In fact, different LPs may be obtained if we choose other directions as axes. These other LPs could perhaps lead to a better objective value, which means that they yield a better approximation of the subjective distances $\delta(u, v)$ that are given in the input. This introduces the problem of finding which system of axes will yield a LP with a minimal objective value. Different axes may be obtained by rotating the original map $x_0$ with an angle in the range of $[0, \pi/2]$. However, not every rotation in this interval will lead to a different linear program. In fact, we only need to consider $\frac{1}{2}n(n-1)+1$ cases. Recall the two vertex orders $O_x$ and $O_y$, where the POIs are sorted according to their $x$ and $y$-coordinates in $x_0$. When two rotation angles are sufficiently close such that they lead to the same orders $O_x$ and $O_y$, the corresponding LPs will be identical and yield the same objective value. Therefore, a different LP is only obtained when the rotations lead to different orders $O_x$

and $O_y$. Furthermore, we only obtain a different order if there is at least one pair of POIs $(u, v)$ such that the rotation makes either the $x$ or $y$-axis cross the line defined by $u$ and $v$.

To identify all rotations that lead to a different LP, for every pair $(u, v) \in V \times V$, we compute the positive angle $\alpha$ between the line defined by $(u, v)$ and the $x$-axis. Then, we should force $\alpha$ in the range $[0, \pi]$ by subtracting $\pi$ if necessary. If $\alpha > \pi/2$ it relates to a change in the order $O_x$. if $\alpha \leq \pi/2$, the rotation relates to a change in $O_y$. Next, to find out the solution with the best objective value, we can try all possible $\frac{1}{2}n(n-1)+1$ rotations which relate to a change in the orders. We can avoid performing the rotations numerically by instead rotating the axes virtually. If we keep a sorted list of the angles $\alpha$, we do not need the rotation of $x_0$ when we move from one LP to the next one. In fact, we only need to invert one of the inequalities relating to the relative orientation constraints.

### 5.4.1 Experiments

Several preliminary experiments were conducted with the method described above. First, the set of 10 POIs from Section 5.3 was analyzed. For each of the rotations in the range $[0, \pi/2]$, we solved the corresponding LPs, leading to the scatter plot shown in Figure 5.3.

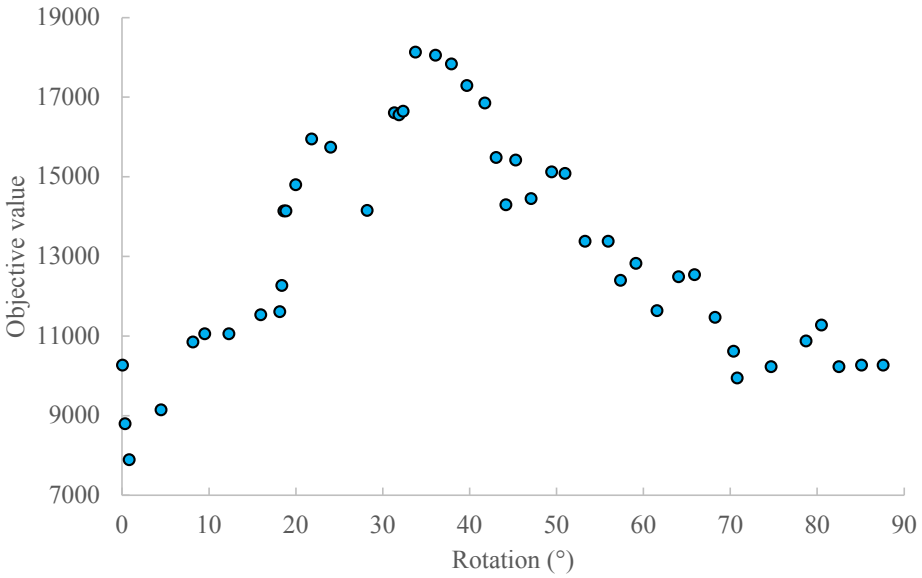



Figure 5.3 – An analysis of the data set with 10 POIs (see Table 5.1). The $x$-axis: the virtual axes rotation in degrees. On the $y$-axis: the objective value of the corresponding LP.

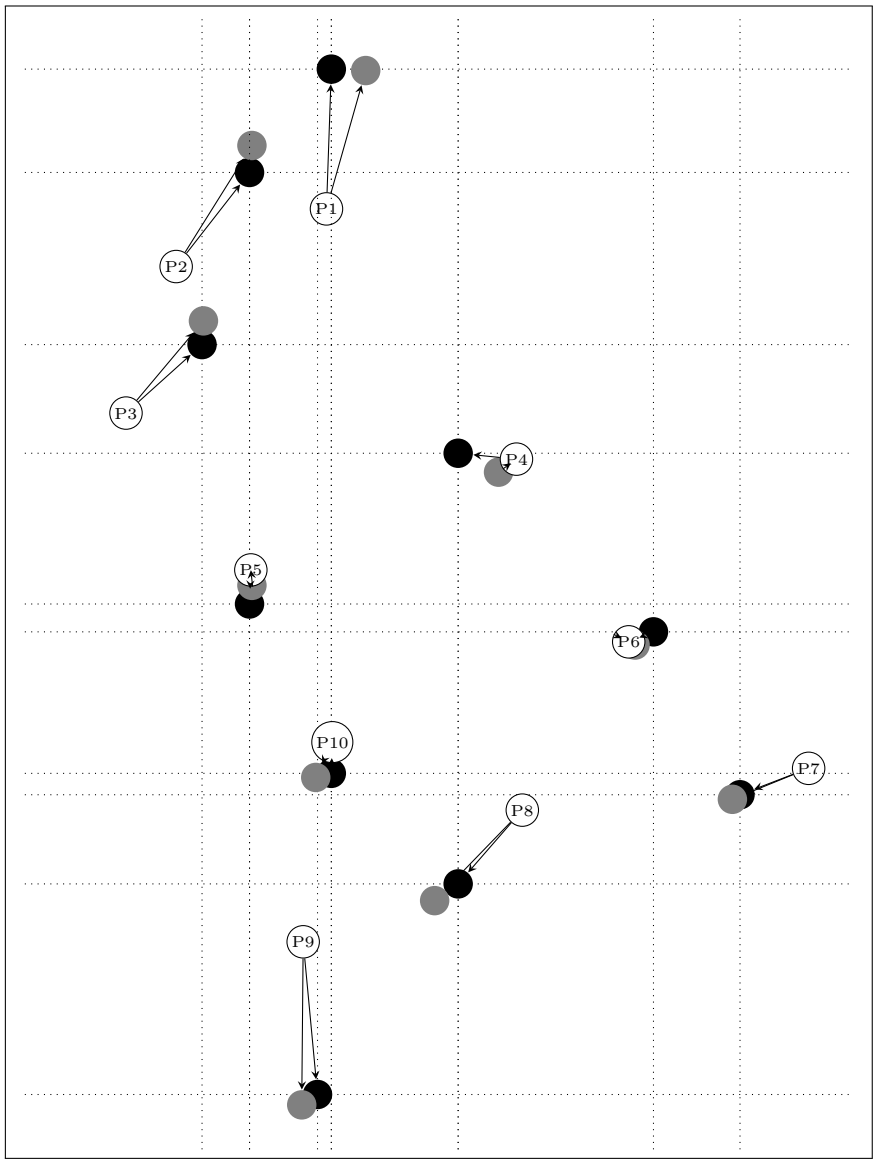



Figure 5.4 – Overlaps the original points (labeled) with the two sets of resulting points. In black, we see the solution with an objective value of 10273, and in gray we see the solution with an objective value of 7896. Gridlines are added for the points in black, showing the relative constraints relating to the pair (P4,P8) and (P1,P10) are swapped.

We see that the different rotations lead to large variations in the objective value. The objective value (as reported in the Section 5.3) is 10273 when we do not perform any virtual rotation of the axes. The rotation leading to the best objective value is a very small rotation of 0.79°, which leads to a solution with an objective value of 7896. To study this more deeply, consider Figure 5.4, where we contrast the original position of

the POIs, with the two solutions. The solution from the LP with the virtual rotation was rotated back by 0.79° to match the axes of the original map.

The virtual rotation of 0.79° changes $O_x$, by swapping the order of two pairs of points: (P1,P10) and (P4,P8). Swapping P1 and P10 has the immediate effect of creating a larger (horizontal) distance between them in the gray solution, as compared to the black one. In the original $x$-order given by $x_0$, P4 is to the left of P8 but in the new one, we force P4 to be to the right of P8. This is visible in Figure 5.4, where we see that in the black solution with an objective value of 10273, P4 and P8 are almost on the same line, while in the new gray solution they have moved apart by quite a bit. Flipping the inequality in the two constraints decreases the error on the distances in different ways. For example, the absolute error of the distance between P2 and P4 is decreased by 351 (P2 and P4 are further away in the new (gray) solution, which better matches the subjective distances in Table 5.1). The pair (P2,P3) has a similar story, decreasing the objective value by 374. All these changes result in a large improvement of the objective value.

| Instance | P10 | P15 | P21 | P31 | P41 | P51 | P61 | P71 | P81 | P91 |
|---|---|---|---|---|---|---|---|---|---|---|
| LPs solved | 46 | 106 | 211 | 466 | 821 | 1276 | 1831 | 2486 | 3241 | 4096 |
| Objective<br>No rotation | 10273 | 14039 | 49.6 | 145.3 | 314.4 | 338 | 573 | 754.4 | 1040.9 | 1428.7 |
| Objective<br>Best rotation | 7896 | 11400 | 42.5 | 131.8 | 292.1 | 318.9 | 500.3 | 735.1 | 994.3 | 1344.7 |
| Improvement | 23.1% | 18.8% | 14.3% | 9.3% | 7.1% | 5.7% | 12.7% | 2.6% | 4.5% | 5.9% |
| Best rotation (°) | 0.78 | 19.5 | 24.1 | 85.6 | 30.5 | 4.2 | 15.2 | 14.3 | 24.8 | 14.8 |
| Total time (s) | 0.36 | 0.69 | 1.8 | 8.4 | 37.7 | 127.9 | 343.8 | 829.4 | 1819.4 | 3600.7 |
| Total time (s)<br>Warm restart | 0.24 | 0.37 | 0.54 | 1.37 | 4.5 | 12.3 | 32.6 | 79.5 | 179.6 | 365.4 |

Table 5.2 – Results of using the exhaustive rotation method on various point sets.

These experiments were also extended to larger data sets, with more POIs, all extracted from the city of Granada. In total, tests were run on 10 data sets, with the following number of POIs: 10, 15, 21, 31, 41, 51, 61, 71, 81, and 91. Note that the subjective distances used in the various point sets were not all of the same type, which means the absolute values of the objective function cannot be compared between the data sets. To test the efficiency of the exhaustive virtual rotation method, we used it on the above data sets, solving many different LPs in the process. For the largest point set, with 91 POIs,

a total of 4096 LPs was solved. The results of these experiments are shown in Table 5.2. The tests were run on the same system described in Section 5.3 and using the academic version of the CPLEX solver.

In the table, we see that rotating the axes can lead to a significant improvement of the objective value, which validates the use of the method. One drawback seems to be that the larger the size of the set of POIs, the larger the number of rotations that we should try to find the best axes for the problem. This is especially reflected in the penultimate row of the table, where we see that the total running time of the method goes up to one hour. However, in practice, the running time can be much reduced by the use of *warm restarts* when solving the LPs. When we go from one virtual rotation to the next, only one inequality is inverted in the new LP compared to the previous one. Therefore, the solutions between the consecutive LPs should be rather close. If we supply the solver with the solution from the previous LP, it is able to use it as a starting point, making it find the next solution much faster. In the final column, we see the effect of including warm restarts in the procedure: the total running time is generally improved by a factor of 10. Apart from these, we can see that the relative improvement tends to decrease with the size of the instance. This is logical because the relative effect of each POI on the objective function decreases with the size of the instances.

## 5.5 Discussion

In this chapter, we used Distance Geometry to study the problem of POI relocation, which is one of the necessary steps when generating adaptive maps. Similar to the topic of structural biology, the instances for POI relocation contain more information than just pairwise distances. In the case of adaptive maps, the instances also contain relative orientations between POIs, which are included so that the generated, adapted maps have a topology that is similar to the original map. These local orientations are forced upon the generated maps in the form of linear constraints, and thus allow us to formulate a linear program for the problem. We presented in-depth experiments for a data set of 10 POIs. Aside from this, we saw that rotating the axes will lead to different local orientation constraints, which in turn lead to different linear programs. These new LPs may have better objective values, which means that the subjective distances in the input can be better approximated. We presented several preliminary experiments using an exhaustive virtual rotation method. The experiments show promising results.

Currently, work is being done on improving this method, by reducing the number of LPs that we need to solve. It is likely unnecessary to exhaustively check all possible virtual rotations. Perhaps through *sensitivity analysis* [215, 216], we can find certain rules that tell us beforehand whether or not a specific LP will lead to an optimal solution that is better than a previously solved LP. Theoretical research into this topic is ongoing and is expected to result in a submission to a journal.

Chapter 6

# Discussion, ongoing and future work

In this last chapter, we will summarize the results presented in the thesis. For each of the topics discussed, we will look at various avenues for further research. Finally, we will look at two projects which are ongoing work and to which only minor contributions were made as part of this thesis. For the first project (Section 6.2.1), we will look at binary representations of combinatorial problems and how we may use meta-heuristics to find solutions. This is of interest in the context of Distance Geometry because solutions to instances of the DDGP subclass may be represented using a binary string. For the second topic (Section 6.2.2), we will consider the DGP in dimension 1. In particular, we focus on one specific type of "paradoxical" instances, which appear to be trivial but are in fact hard to solve by branch-and-prune methods. We propose alternative methods, based on a matrix-by-vector reformulation, which can solve these paradoxical instances more efficiently.

## 6.1 Discussion and future work

In this thesis, we have discussed various aspects of Distance Geometry. The main focus was the application of structural biology using Discretizable Distance Geometry methods. A common theme in the work presented in this thesis is that not only Euclidean distances but also other geometric concepts can be exploited by DG methods.

In Chapter 2 we showed that if we include not only distance information in the instances, but also torsion angles (including their sign), this will help the branch-and-prune methods that exploit the discretizability of the instances. Specifically, the inclusion of the torsion angles will decrease the size of the tree that represents the search space. In Chapter 3, we presented two sets of experiments for the application of protein structure determination, exploiting this idea in practice. We showed that it can be desirable to not only include inter-atomic distances in the input instances, but also both proper and improper dihedral angles. We saw that when we encapsulate this information in our instances, BP methods can produce relatively good quality structures for small proteins while relying on (noisy) NMR data. However, a second set of experiments shows that there are still challenges to overcome. This is because, in the context of the BP framework, the geometric information relating to the covalent geometry of the proteins is generally regarded as constant. However, in practice, we see that there are quite large variations in this data. Despite the error-tolerance of the BP methods used, the discrepancies found within the covalent geometry make it so that the methods tend to fall short for larger proteins. Future work should focus on overcoming this problem. One strategy may be to attempt to capture these variations in the input. This could be done by using machine learning or statistical analysis in order to identify variations in the distances, for example, based on the position of the amino acid in the primary structure of the protein. This way, we may be able to use precise distances that better match the real distances in a given conformation of a protein. The second problem, the uncertainty in the instances, may be improved by including more distances. One possible new distance type that we could include is based on pairwise CB distances. These distances can be obtained from contact maps, which are produced by machine learning methods used in the context of protein structure determination [172, 173]. Contact maps give us information about CB atoms that are close in space but may be far from each other in the primary structure of the protein. This means that these distances can be very useful, as they will let us prune away large parts of the search space tree. However, the use of distances such as

these presents many new challenges. Some preliminary experiments show that the pruning distances may be too strict and branch-and-prune will end up pruning every branch of the tree. In these same tests, it appears that SPG is not powerful enough to permute the incumbent solution such that these distances are satisfied. This points to an additional point of future work, which also relates to the refinement step. We saw in this thesis that including torsion angles and their sign is very beneficial for the BP methods. However, in the current state, SPG only minimizes the error on the distances in the instances and does not consider the torsion angle information at all. Including this information in the refinement step could lead to an improvement of the generated structures. The current implementation of SPG relies on an exact derivation of the penalty function for the gradient descent. Extending this derivation to also include a violation of the dihedral angles is not a trivial task. Furthermore, this computation may slow down the algorithm. We did some tests with numerical differentiation methods, but these tend to be very slow. A possible tool that could be looked into is automatic differentiation.

Aside from these specific directions and points of improvement, a more general interesting direction of research is combining the discrete, BP method with machine learning. In recent years, the most promising methods for protein structure determination have been based on machine learning and have achieved very positive results. However, these machine learning methods have their advantages and disadvantages. Perhaps if we could combine our discrete method with some machine learning aspects, we could take the best things from both worlds. A straightforward example would be to use BP as a post-processing step after a machine learning method has produced a contact map. However, more intricate combinations could produce better results.

In Chapter 4 we studied dynamic Distance Geometry, focusing on human motions. Our interest in human motions stems from the fact that the characters are represented using skeletal structures. Researching these human motions lays the groundwork for studying protein dynamics because proteins, like humans, also have a skeletal structure. Several experiments were presented on the subject of using DG methods for the problem of motion retargeting, where we wish to copy the motion from one actor to another character with a different morphology. Furthermore, we proposed a new (vector-torsion angle) representation for human motions, which has its roots in molecular biology. This new representation uses fewer degrees of freedom than the standard Euler angle representation. We showed that the vector-torsion angle representation is a natural fit for the application of motion retargeting. Future work directly on the topic of human motions includes expanding the

DG methods for motion retargeting. Furthermore, the vector-torsion angle representation (combined with the statistical analysis presented in Section 4.4 could potentially be used to generate random motions that guarantee that the actor always has a natural pose. Lastly, the conducted research should be extended to the subject of protein dynamics. This way, proteins moving between conformations could be visualized in an accurate way such that certain important pairwise distances (such as bond lengths and van der Waals radii) are not compromised.

In Chapter 5 we studied the application of adaptive maps. For this application, similarly to structural biology, the DGP instances also include geometric information other than Euclidean distances. For adaptive maps, we use linear distance norms and constraints based on local orientations. This special kind of DGP instances can be solved using linear models and linear programming. We presented experiments in which we solved these linear models with promising results. Furthermore, we discussed ongoing work in which different combinations of local orientations are tried by rotating the directions of the axes. In future work, we will conduct an in-depth theoretical analysis of the concept of rotating the axes and present more detailed experiments. Aside from continuing the research with a focus solely on adaptive maps, it could also be interesting to study whether these findings have any impact on other applications, such as structural biology. Perhaps the local orientations and the linear programs can be interesting for problems in that field as well.

## 6.2 Ongoing work

In this section, we will shortly discuss some ongoing work, to which the author of the thesis has contributed. This work spans two main topics which have not yet been discussed so far in the thesis. The first topic that we will discuss is binary representations of combinatorial problems and a meta-heuristics project that can be used to solve such problems. Note that the DDGP is one such combinatorial problem for which we can use a binary representation. The second topic covers the DGP in dimension 1 and two approaches that can solve a specific type of instances in this dimension.

### 6.2.1 Binary representations

This work concerns a project known as BINMETA [1] [217], which is conceived particularly to study meta-heuristic searches for global optimization problems for which a suitable binary representation can be supplied. In such a binary representation of a solution to an optimization problem, the smallest piece of information that we consider is a *bit*, while a bit string of fixed length $n$ provides the full representation. Two solutions are different if at least one of their bits has an opposite value. The distance between two solutions can be measured by using the *Hamming distance* [218], which counts the number of bits that are different at the same positions in the two strings. Binary strings are useful for representing problems when they admit a discrete search space.

Recall that in Chapter 2 we saw that under certain assumptions, the DDGP admits a discrete search space (see Section 2.1). In fact, the DDGP is a global optimization problem for which a binary representation comes naturally. An example of a binary tree representing the search space of an instance of the DDGP with $|V| = 6$ and $K = 3$ is given in Figure 6.1.

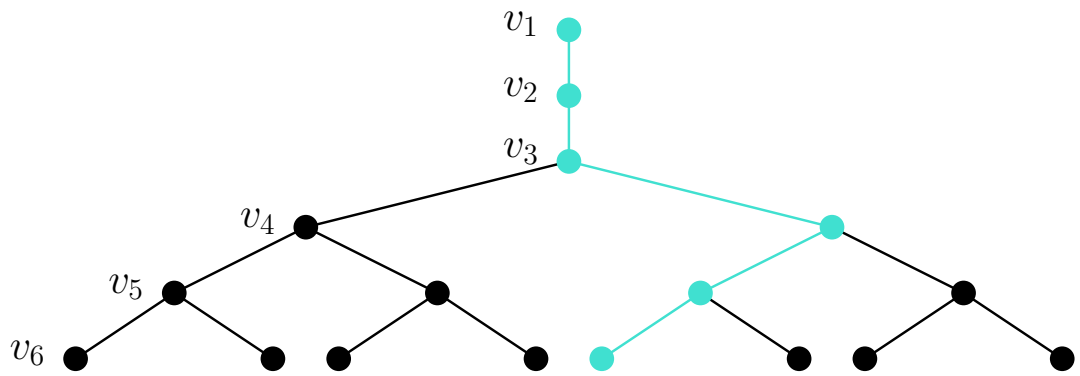



Figure 6.1 – An example of a tree representing the search space of an instance of the DDGP with $|V| = 6$ and $K = 3$. A path in the tree, corresponding to a possible solution to the instance is highlighted. The binary string that corresponds to the highlighted path is 100.

Any path in this tree from the root to a leaf node corresponds to a solution for the instance at hand. Such a path can be represented with a binary string in several ways. An example of a simple binary representation scheme is to assign a bit for each level (vertex) of the tree. If a bit is set to 0, it means that at the corresponding level in the tree, we select the left candidate. If the bit is 1, we choose the right candidate. Because the first $K$ vertices are fixed, we only need $|V| - K$ bits to represent a solution. The bit string that

1. `https://github.com/mucherino/binMeta`

represents the highlighted solution in the tree in Figure 6.1 is 100. Every permutation of such a bit string will correspond to a different path in the tree. For each such solution, we know that all the reference distances are satisfied. Therefore, the quality of such a solution may be quantified by using a penalty function that minimizes the violation of the pruning distances.

The binMeta project groups implementations of meta-heuristics that can be used to solve global optimization problems as long as they admit a binary representation. A contribution was made to this project as part of the thesis. The well-known meta-heuristic search Variable Neighbourhood Search (VNS) was adapted to binary solutions and Hamming space and added to the list of binMeta implementations. More details can be found in [219]. Future work can be done in various directions. Firstly, the different meta-heuristic implementations should be tested in detail on instances of the DDGP. The binary representation of the DDGP can be especially useful when the DDGP is regarded as a series of nested subproblems, such as in [220]. The binary representations and the meta-heuristics may be valuable in this context because we can perform *contractions* in the bit strings, collapsing the solution for a subproblem into one bit. Apart from studying the DDGP, in general, it may be interesting to investigate the meta-heuristics in the context of problems with constraints. As mentioned before, a binary string may represent a solution for the DDGP in which not all pruning distances are satisfied. Similarly, other problems exist for which not all binary strings will represent feasible solutions, where all constraints are satisfied. Particular attention should be given to the conception of the binary representation, and the possibility of an a priori study of the implied landscapes on the objective function.

### 6.2.2 The DGP in dimension 1

In 1979, Saxe proved that the DGP is NP-complete when $K$ is set to 1 [24]. As mentioned briefly in Chapter 1, the main application of the DGP in this dimension is the clock synchronization problem [61, 71, 72], where we are given a set of clocks (represented by vertices $v \in V$ and offset measurements between the clocks (the edges $\{u, v\} \in E$ and the values of the weights $d(u, v)$).

As part of this thesis, contributions were made to a specific subclass of the DGP in dimension 1. This subclass is referred to as *paradoxical* [28–30]. These kinds of instances are represented by a graph $G$ that is a cycle graph, for which a vertex order can trivially be identified. The reason that we refer to this subclass as paradoxical is twofold. Firstly,

constructing a solution for this subclass seems relatively easy: most vertices $v$ depend only on one predecessor (reference vertex) $u$. Therefore, if we were to solve such instances using branch-and-prune, for each $v$, two new positions $x_v$ and $x'_v$ are easily identified, allowing us to build up the binary discretization tree. However, the absence of pruning distances up to the last vertex makes this class of instances very hard. Only at the very last layer of the tree, do we have access to the pruning distance between the first and last vertex, which means only then we are able to validate and prune the incorrect branches in the tree. Therefore, we first have to construct a tree of size $2^{|V|-1}$, before we can select the 2 correct solutions from all the generated realizations (paths in the tree). To avoid this issue, we proposed matrix-by-vector reformulation of this paradoxical class in [29]. This reformulation lends itself to parallelization and in two separate works we presented two different methods to solve this matrix-by-vector multiplication.

The first method, proposed in [29], is based on optical processing. Optical computing schemes have been proposed in the scientific literature to attempt to find the solutions for NP-hard problems [221–224]. We presented a new architecture for the realization of an optical processor, based on the modulation of an optical wavefront by spatial light modulators. This architecture is able to perform fast matrix-by-vector multiplication, which means that it could be used to solve instances of the paradoxical subclass of the DGP in dimension 1. Future work on this topic includes putting the optical scheme into practice and using it to solve some small instances. More details can be found in [29].

The second method, presented in [28], is implemented in C and CUDA so that the computations may be employed on the GPU. We tested this implementation with randomly generated paradoxical instances. In these experiments, the GPU implementation was compared to a branch-and-prune implementation for dimension 1. The results show that the GPU method is around 16 times faster than the BP implementation for these paradoxical instances, showing the usefulness of the matrix-by-vector reformulation.

A first line of future work exists in extending the practical experiments for the DGP in dimension 1. On a more theoretical note, it would be interesting to extend the study of the paradoxical instance class to higher dimensions.

# AUTHOR'S CONTRIBUTIONS

[1] S. B. Hengeveld, T. Malliavin, J. Lin, L. Liberti, and A. Mucherino, *A Study on the Impact of the Distance Types Involved in Protein Structure Determination by NMR*, Computational Structural Bioinformatics Workshop (CSBW21), IEEE International Conference on Bioinformatics and Biomedicine (BIBM21), pp. 2502–2510, 2021.

[2] S. B. Hengeveld, M. Merabti, F. Pascale, and T. E. Malliavin, *A Study on the Covalent Geometry of Proteins and Its Impact on Distance Geometry*, International Conference on Geometric Science of Information, pp. 520–530, 2023.

[3] S. B. Hengeveld, T. E. Malliavin, L. Liberti, and A. Mucherino, *Collecting Data for Generating Distance Geometry Graphs for Protein Structure Determination*, Proceedings of Roadef23, 2023.

[4] S. B. Hengeveld and A. Mucherino, *On the Representation of Human Motions and Distance-based Retargeting*, 16th Conference on Computer Science and Intelligence Systems (FedCSIS21), pp. 181–189, 2021.

[5] S. B. Hengeveld and A. Mucherino, *On the Feasible Regions Delimiting Natural Human Postures in a Novel Skeletal Representation*, 17th Conference on Computer Science and Intelligence Systems (FedCSIS22), pp. 175–179, 2022.

[6] S. B. Hengeveld, F. Plastria, A. Mucherino, and D. A. Pelta, *A linear program for points of interest relocation in adaptive maps*, International Conference on Geometric Science of Information, pp. 551–559, 2023.

[7] S. B. Hengeveld and A. Mucherino, *Variable Neighborhood Search in Hamming Space*, To appear in Lecture Notes in Computer Science, Proceedings of the 14th International Conference on Large-Scale Scientific Computations (LSSC23), 2023.

[8] S. B. Hengeveld and A. Mucherino, *A GPU Approach to Distance Geometry in 1D: an Implementation in C/CUDA*, 17th Conference on Computer Science and Intelligence Systems (FedCSIS22), pp. 333–336, 2022.

[9] S. B. Hengeveld, N. R. da Silva, D. S. Gonçalves, P. H. S. Ribeiro, and A. Mucherino, *An optical processor for matrix-by-vector multiplication: an application to the distance geometry problem in 1D*, Journal of Optics, vol. 24, p. 015 701, 2021.

[10] S. B. Hengeveld and A. Mucherino, *The Discrete Side of Distance Geometry: a Focus on the 1-dimensional Case*, Proceedings of Discrete Math Days, 2022.

# BIBLIOGRAPHY


[1] A. M.-C. So and Y. Ye, *Theory of semidefinite programming for sensor network localization*, Mathematical Programming, vol. 109, pp. 367–384, 2007.

[2] R. Harris, *Nuclear Magnetic Resonance*. 1971.

[3] A. Mucherino, C. Lavor, L. Liberti, and N. Maculan, *Distance Geometry: Theory, Methods and Applications*. 2013.

[4] A. Mucherino and D. S. Gonçalves, *An Approach to Dynamical Distance Geometry*, Geometric Science of Information, pp. 821–829, 2017.

[5] A. Mucherino, J. Omer, L. Hoyet, P. Giordano, and F. Multon, *An application-based characterization of dynamical distance geometry problems*, Optimization Letters, vol. 14, pp. 1–15, 2020.

[6] A. Mucherino, C. Lavor, and L. Liberti, *The Discretizable Distance Geometry Problem*, Optimization Letters, vol. 6, pp. 1671–1686, 2012.

[7] A. Mucherino, L. Liberti, and C. Lavor, *MD-jeep: an Implementation of a Branch and Prune Algorithm for Distance Geometry Problems*, Lectures Notes in Computer Science 6327, K. Fukuda et al. (Eds.), Proceedings of the 3rd International Congress on Mathematical Software (ICMS10), Kobe, Japan, pp. 186–197, 2010.

[8] C. Lavor, L. Liberti, N. Maculan, and A. Mucherino, *The Discretizable Molecular Distance Geometry Problem*, Computational Optimization and Applications, vol. 52, pp. 115–146, 2012.

[9] A. Mucherino, J.-H. Lin, and D. S. Gonçalves, *A coarse-grained representation for discretizable distance geometry with interval data*, Bioinformatics and Biomedical Engineering: 7th International Work-Conference, IWBBIO 2019, pp. 3–13, 2019.

[10] A. Mucherino and C. Lavor, *The Branch and Prune Algorithm for the Molecular Distance Geometry Problem with Inexact Distances*, Proceedings of the International Conference on Computational Biology, vol. 58, pp. 349–353, 2009.

[11] C. Lavor, L. Liberti, and A. Mucherino, *The interval Branch-and-Prune algorithm for the discretizable molecular distance geometry problem with inexact distances*, Journal of Global Optimization, vol. 56, pp. 855–871, 2013.

[12] C. Lavor, M. Souza, L. M. Carvalho, D. S. Gonçalves, and A. Mucherino, *Improving the sampling process in the interval branch-and-prune algorithm for the discretizable molecular distance geometry problem*, Applied Mathematics and Computation, vol. 389, p. 125 586, 2021.

[13] A. Mucherino, D. Gonçalves, L. Liberti, J.-H. Lin, C. Lavor, and N. Maculan, *MD-jeep: a New Release for Discretizable Distance Geometry Problems with Interval Data*, IEEE Conference Proceedings, Federated Conference on Computer Science and Information Systems (FedCSIS20), Workshop on Computational Optimization (WCO20), pp. 289–294, 2020.

[14] A. Mucherino and J.-H. Lin, *An efficient exhaustive search for the discretizable distance geometry problem with interval data*, 2019 Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 135–141, 2019.

[15] M. Gleicher, *Retargetting Motion to New Characters*, Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, pp. 33–42, 1998.

[16] E. S. L. Ho, T. Komura, and C.-L. Tai, *Spatial Relationship Preserving Character Motion Adaptation*, ACM Trans. Graph., vol. 29, no. 4, 2010.

[17] S. Guo, R. Southern, J. Chang, D. Greer, and J. Zhang, *Adaptive motion synthesis for virtual characters: a survey*, The Visual Computer, vol. 31, no. 5, pp. 497–512, 2015.

[18] S. B. Hengeveld and A. Mucherino, *On the Representation of Human Motions and Distance-based Retargeting*, 16th Conference on Computer Science and Intelligence Systems (FedCSIS21), pp. 181–189, 2021.

[19] S. B. Hengeveld and A. Mucherino, *On the Feasible Regions Delimiting Natural Human Postures in a Novel Skeletal Representation*, 17th Conference on Computer Science and Intelligence Systems (FedCSIS22), pp. 175–179, 2022.

[20] M. Torres Anaya, D. Pelta, and J. L. Verdegay, *A Proposal for Adaptive Maps*, pp. 657–666, 2018.

[21] M. Torres Anaya, D. Pelta, J. L. Verdegay, and C. Cruz, *Towards adaptive maps*, International Journal of Intelligent Systems, vol. 34, 2018.

[22] L. Liberti, C. Lavor, N. Maculan, and A. Mucherino, *Euclidean Distance Geometry and Applications*, SIAM Review, vol. 56, pp. 3–69, 2014.

[23] Y. Yemini, *Some Theoretical Aspects of Position-location Problems*, 20th Annual Symposium on Foundations of Computer Science, pp. 1–8, 1979.

[24] J. Saxe, *Embeddability of Weighted Graphs in K-space is Strongly NP-hard*, Procedings of $17^{th}$ Allerton Conference in Communcations, Control and Computing, pp. 480–489, 1979.

[25] P. Duxbury, L. Granlund, S. Gujarathi, P. Juhas, and S. Billinge, *The unassigned distance geometry problem*, Discrete Applied Mathematics, vol. 204, pp. 117–132, 2016.

[26] A. Mucherino, *On the Discretization of Distance Geometry: Theory, Algorithms and Applications*, HDR Monograph, University of Rennes 1, 2018.

[27] S. B. Hengeveld, T. Malliavin, J. Lin, L. Liberti, and A. Mucherino, *A Study on the Impact of the Distance Types Involved in Protein Structure Determination by NMR*, Computational Structural Bioinformatics Workshop (CSBW21), IEEE International Conference on Bioinformatics and Biomedicine (BIBM21), pp. 2502–2510, 2021.

[28] S. B. Hengeveld and A. Mucherino, *A GPU Approach to Distance Geometry in 1D: an Implementation in C/CUDA*, 17th Conference on Computer Science and Intelligence Systems (FedCSIS22), pp. 333–336, 2022.

[29] S. B. Hengeveld, N. R. da Silva, D. S. Gonçalves, P. H. S. Ribeiro, and A. Mucherino, *An optical processor for matrix-by-vector multiplication: an application to the distance geometry problem in 1D*, Journal of Optics, vol. 24, p. 015 701, 2021.

[30] S. B. Hengeveld and A. Mucherino, *The Discrete Side of Distance Geometry: a Focus on the 1-dimensional Case*, Proceedings of Discrete Math Days, 2022.

[31] G. M. Crippen, *An Alternative Approach to Distance Geometry using $L_\infty$ Distances*, Discrete Applied Mathematics, vol. 197, pp. 20–26, 2015.

[32] C. D'Ambrosio and L. Liberti, *Distance Geometry in Linearizable Norms*, Geometric Science of Information, pp. 830–837, 2017.

[33] R. S. Lima and J. Martínez, *Solving molecular Distance Geometry Problems using a continuous optimization approach*, Distance Geometry: Theory, Methods, and Applications, pp. 213–224, 2012.

[34] J. C. Gower, *Euclidean Distance Geometry*, vol. 7, 1982, pp. 1–14.

[35] A. Y. Alfakih, A. K. Khandani, and H. Wolkowicz, *Solving Euclidean Distance Matrix Completion Problems Via Semidefinite Programming*, Computational Optimization and Applications, vol. 12, pp. 13–30, 1999.

[36] I. Dokmanic, R. Parhizkar, J. Ranieri, and M. Vetterli, *Euclidean Distance Matrices: Essential theory, algorithms, and applications*, IEEE Signal Processing Magazine, vol. 32, no. 6, pp. 12–30, 2015.

[37] G. Crippen and T. Havel, *Distance Geometry and Molecular Conformation*, Chemometrics, vol. 15, 1988.

[38] B. Alipanahi, N. Krislock, A. Ghodsi, H. Wolkowicz, L. Donaldson, and M. Li, *Determining protein structures from NOESY distance constraints by semidefinite programming*, Journal of Computational Biology, vol. 20, pp. 296–310, 2013.

[39] W. Glunt, T. L. Hayden, and M. Raydan, *Molecular conformations from distance matrices*, Journal of Computational Chemistry, vol. 14, pp. 114–120, 1993.

[40] F. Almeida, A. Moraes, and F. Gomes-Neto, *An Overview on Protein Structure Determination by NMR: Historical and Future Perspectives of the use of Distance Geometry Methods*, Distance Geometry: Theory, Methods, and Applications, Nov. 2013.

[41] T. E. Malliavin, A. Mucherino, and M. Nilges, *Distance Geometry in Structural Biology: New Perspectives*, Distance Geometry: Theory, Methods, and Applications, pp. 329–350, 2013.

[42] I. T. Jolliffe, *Discarding Variables in a Principal Component Analysis. I: Artificial Data*, Journal of the Royal Statistical Society. Series C (Applied Statistics), vol. 21, pp. 160–173, 1972.

[43] J. J. Moré and Z. Wu, *Global continuation for distance geometry problems*, SIAM Journal on Optimization, vol. 7, pp. 814–836, 1997.

[44] J. J. Moré and Z. Wu, *Distance geometry optimization for protein structures*, Journal of Global Optimization, vol. 15, p. 219, 1999.

[45] X. Fang and K.-C. Toh, *Using a Distributed SDP Approach to Solve Simulated Protein Molecular Conformation Problems*, Distance Geometry: Theory, Methods, and Applications, A. Mucherino, C. Lavor, L. Liberti, and N. Maculan, Eds., pp. 351–376, 2013.

[46] E. F. Pettersen *et al.*, *UCSF Chimera—a visualization system for exploratory research and analysis*, Journal of computational chemistry, vol. 25, no. 13, pp. 1605–1612, 2004.

[47] D. Helbing, I. Farkas, and T. Vicsek, *Simulating dynamical features of escape panic*, Nature, vol. 407, no. 6803, pp. 487–490, 2000.

[48] A.-H. Olivier, A. Marin, A. Crétual, and J. Pettré, *Minimal predicted distance: A common metric for collision avoidance during pairwise interactions between walkers*, Gait & posture, vol. 36, no. 3, pp. 399–404, 2012.

[49] J. Omer, *A space-discretized mixed-integer linear model for air-conflict resolution with speed and heading maneuvers*, Computers & Operations Research, vol. 58, pp. 75–86, 2015.

[50] L. Asimow and B. Roth, *The rigidity of graphs*, Transactions of the American Mathematical Society, vol. 245, pp. 279–289, 1978.

[51] M. Laurent, *Cuts, matrix completions and graph rigidity*, Mathematical Programming, vol. 79, pp. 255–283, 1997.

[52] A. Alfakih, *Graph rigidity via euclidean distance matrices*, Linear Algebra and its Applications, vol. 310, no. 1-3, pp. 149–165, 2000.

[53] B. Jackson and T. Jordán, *Connected rigidity matroids and unique realizations of graphs*, Journal of Combinatorial Theory, Series B, vol. 94, no. 1, pp. 1–29, 2005.

[54] G. Laman, *On graphs and rigidity of plane skeletal structures*, Journal of Engineering mathematics, vol. 4, no. 4, pp. 331–340, 1970.

[55] B. Hendrickson, *Conditions for unique graph realizations*, SIAM journal on computing, vol. 21, no. 1, pp. 65–84, 1992.

[56] L. Hennberg, *Statik der starren systeme*, A. Bergstraesser, vol. 1, 1886.

[57] L. Cremona, *Le figure reciproche nella statica grafica*. 1879.

[58] P. Biswas, T.-C. Lian, T.-C. Wang, and Y. Ye, *Semidefinite programming based algorithms for sensor network localization*, ACM Transactions on Sensor Networks (TOSN), vol. 2, pp. 188–220, 2006.

[59] N. Krislock and H. Wolkowicz, *Explicit sensor network localization using semidefinite representations and facial reductions*, SIAM Journal on Optimization, vol. 20, pp. 2679–2708, 2010.

[60] P. Biswas, T.-C. Liang, K.-C. Toh, Y. Ye, and T.-C. Wang, *Semidefinite programming approaches for sensor network localization with noisy distance measurements*, IEEE transactions on automation science and engineering, vol. 3, no. 4, pp. 360–371, 2006.

[61] Y.-C. Wu, Q. Chaudhari, and E. Serpedin, *Clock Synchronization of Wireless Sensor Networks*, IEEE Signal Processing Magazine, vol. 28, pp. 124–138, 2011.

[62] G. Mao, B. Fidan, and B. D. Anderson, *Wireless sensor network localization techniques*, Computer networks, vol. 51, no. 10, pp. 2529–2553, 2007.

[63] M. Cao, B. D. Anderson, and A. S. Morse, *Sensor network localization with imprecise distances*, Systems & control letters, vol. 55, no. 11, pp. 887–893, 2006.

[64] S. Rai and S. Varma, *Localization in wireless sensor networks using rigid graphs: a review*, Wireless Personal Communications, vol. 96, no. 3, pp. 4467–4484, 2017.

[65] W. S. Torgerson, *Multidimensional scaling: I. Theory and method*, Psychometrika, vol. 17, no. 4, pp. 401–419, 1952.

[66] J. B. Tenenbaum, V. d. Silva, and J. C. Langford, *A global geometric framework for nonlinear dimensionality reduction*, science, vol. 290, no. 5500, pp. 2319–2323, 2000.

[67] M. C. Hout, M. H. Papesh, and S. D. Goldinger, *Multidimensional scaling*, Wiley Interdisciplinary Reviews: Cognitive Science, vol. 4, no. 1, pp. 93–103, 2013.

[68] F. Wang and J. Sun, *Survey on distance metric learning and dimensionality reduction in data mining*, Data mining and knowledge discovery, vol. 29, no. 2, pp. 534–564, 2015.

[69] Y. Hong, Q. Li, J. Jiang, and Z. Tu, *Learning a mixture of sparse distance metrics for classification and dimensionality reduction*, 2011 International Conference on Computer Vision, pp. 906–913, 2011.

[70] L. Liberti, *Distance geometry and data science*, Top, vol. 28, no. 2, pp. 271–339, 2020.

[71] N. M. Freris, S. R. Graham, and P. R. Kumar, *Fundamental Limits on Synchronizing Clocks Over Networks*, IEEE Transactions on Automatic Control, vol. 56, no. 6, pp. 1352–1364, 2011.

[72] P. Veríssimo and M. Raynal, *Time in Distributed System Models and Algorithms*, Advances in Distributed Systems: Advanced Distributed Computing: From Algorithms to Systems, S. Krakowiak and S. Shrivastava, Eds., 2000.

[73] A. Bahr, J. J. Leo nard, and M. F. Fallon, *Cooperative localization for autonomous underwater vehicles*, The International Journal of Robotics Research, vol. 28, no. 6, pp. 714–728, 2009.

[74] Q. Chen, K. You, and S. Song, *Cooperative localization for autonomous underwater vehicles using parallel projection*, 2017 13th IEEE International Conference on Control & Automation (ICCA), pp. 788–793, 2017.

[75] E. D. Demaine *et al.*, *The Distance Geometry of Music*, Computational geometry, vol. 42, no. 5, pp. 429–454, 2009.

[76] F. Beck, M. Burch, S. Diehl, and D. Weiskopf, *A taxonomy and survey of dynamic graph visualization*, Computer graphics forum, vol. 36, no. 1, pp. 133–159, 2017.

[77] I. Herman, G. Melançon, and M. S. Marshall, *Graph visualization and navigation in information visualization: a survey*, IEEE Transactions on visualization and computer graphics, vol. 6, no. 1, pp. 24–43, 2000.

[78] R. de Freitas, B. Dias, N. Maculan, and J. Szwarcfiter, *Distance geometry approach for special graph coloring problems*, arXiv preprint arXiv:1606.04978, 2016.

[79] I. J. Schoenberg, *Remarks to maurice frechet's article"sur la definition axiomatique d'une classe d'espace distances vectoriellement applicable sur l'espace de hilbert*, Annals of Mathematics, pp. 724–732, 1935.

[80] K. Pearson, *On lines and planes of closest fit to systems of points in space*, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, vol. 2, pp. 559–572, 1901.

[81] A. E. García, *Large-amplitude nonlinear motions in proteins*, Phys. Rev. Lett., vol. 68, pp. 2696–2699, 1992.

[82] R. H. Keshavan, A. Montanari, and S. Oh, *Matrix Completion from a Few Entries*, IEEE transactions on information theory, vol. 56, pp. 2980–2998, 2010.

[83] P. Jain, P. Netrapalli, and S. Sanghavi, *Low-rank matrix completion using alternating minimization*, Proceedings of the forty-fifth annual ACM symposium on Theory of computing, pp. 665–674, 2013.

[84] R. Reams, G. Chatham, W. Glunt, D. McDonald, and T. Hayden, *Determining protein structure using the distance geometry program apa*, Computers & Chemistry, vol. 23, no. 2, pp. 153–163, 1999.

[85] L. Grippo and M. Sciandrone, *On the convergence of the block nonlinear gauss–seidel method under convex constraints*, Operations research letters, vol. 26, no. 3, pp. 127–136, 2000.

[86] M. J. Sippl and H. A. Scheraga, *Solution of the embedding problem and decomposition of symmetric matrices.*, Proceedings of the National Academy of Sciences, vol. 82, no. 8, pp. 2197–2201, 1985.

[87] N. Krislock and H. Wolkowicz, *Explicit sensor network localization using semidefinite representations and facial reductions*, SIAM Journal on Optimization, vol. 20, no. 5, pp. 2679–2708, 2010.

[88] H. Shi and Q. Li, *A facial reduction approach for the single source localization problem*, Journal of Global Optimization, pp. 1–25, 2022.

[89] D. K. Agrafiotis, *Stochastic Proximity Embedding*, Journal of computational chemistry, vol. 24, no. 10, pp. 1215–1221, 2003.

[90] B. Hendrickson, *The molecule problem: exploiting structure in global optimization*, SIAM Journal on Optimization, vol. 5, no. 4, pp. 835–857, 1995.

[91] P. Biswas, K.-C. Toh, and Y. Ye, *A distributed SDP approach for large-scale noisy anchor-free graph realization with applications to molecular conformation*, SIAM Journal on Scientific Computing, vol. 30, no. 3, pp. 1251–1277, 2008.

[92] P. Biswas and Y. Ye, *Semidefinite programming for ad hoc wireless sensor network localization*, pp. 46–54, 2004.

[93] M. Cucuringu, A. Singer, and D. Cowburn, *Eigenvector synchronization, graph rigidity and the molecule problem*, Information and Inference: A Journal of the IMA, vol. 1, no. 1, pp. 21–67, 2012.

[94] C. Lavor, L. Liberti, and N. Maculan, *Computational experience with the molecular distance geometry problem*, Global optimization: scientific and engineering case studies, pp. 213–225, 2006.

[95] L. T. Hoai An, *Solving large scale molecular Distance Geometry Problems by a smoothing technique via the Gaussian transform and D.C. programming*, Journal of Global Optimization, vol. 27, no. 4, pp. 375–397, 2003.

[96] L. T. H. An and P. D. Tao, *Large-scale molecular optimization from distance matrices by D.C. optimization approach*, SIAM Journal on Optimization, vol. 14, no. 1, pp. 77–114, 2003.

[97] L. Liberti, C. Lavor, N. Maculan, and F. Marinelli, *Double variable neighbourhood search with smoothing for the molecular distance geometry problem*, Journal of Global Optimization, vol. 43, pp. 207–218, 2009.

[98] G. A. Williams, J. M. Dugan, and R. B. Altman, *Constrained global optimization for estimating molecular structure from atomic distances*, Journal of computational biology, vol. 8, no. 5, pp. 523–547, 2001.

[99] M. Souza, A. E. Xavier, C. Lavor, and N. Maculan, *Hyperbolic smoothing and penalty techniques applied to molecular structure determination*, Operations Research Letters, vol. 39, no. 6, pp. 461–465, 2011.

[100] S. Kucherenko and Y. Sytsko, *Application of deterministic low-discrepancy sequences in global optimization*, Computational Optimization and Applications, vol. 30, pp. 297–318, 2005.

[101] L. Liberti and M. Drazic, *Variable neighbourhood search for the global optimization of constrained NLPs*, Proceedings of GO Workshop, Almeria, Spain, vol. 2005, 2005.

[102] A. Grosso, M. Locatelli, and F. Schoen, *Solving molecular distance geometry problems by global optimization algorithms*, Computational Optimization and Applications, vol. 43, pp. 23–37, 2009.

[103] D. J. Wales and J. P. Doye, *Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms*, The Journal of Physical Chemistry A, vol. 101, no. 28, pp. 5111–5116, 1997.

[104] D. S. Gonçalves and A. Mucherino, *A distance geometry procedure using the Levenberg-Marquardt algorithm and with applications in biology but not only*, International Work-Conference on Bioinformatics and Biomedical Engineering, pp. 142–152, 2022.

[105] A. Mucherino, D. Gonçalves, A. Bernardin, L. Hoyet, and F. Multon, *A Distance-Based Approach for Human Posture Simulations*, 12th Conference on Computer Science and Intelligence Systems (FedCSIS17), pp. 441–444, 2017.

[106] A. Bernardin, L. Hoyet, A. Mucherino, D. Gonçalves, and F. Multon, *Normalized Euclidean Distance Matrices for Human Motion Retargeting*, Proceedings of the 10th journal Conference on Motion in Games (MIG '17), 2017.

[107] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. 2016.

[108] S. Ruder, *An overview of gradient descent optimization algorithms*, arXiv preprint arXiv:1609.04747, 2016.

[109] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, *On optimization methods for deep learning*, Proceedings of the 28th international conference on international conference on machine learning, pp. 265–272, 2011.

[110] S.-i. Amari, *Backpropagation and stochastic gradient descent method*, Neurocomputing, vol. 5, no. 4-5, pp. 185–196, 1993.

[111] E. G. Birgin, J. M. Martínez, and M. Raydan, *Spectral projected gradient methods: review and perspectives*, Journal of Statistical Software, vol. 60, pp. 1–21, 2014.

[112] L. Grippo, F. Lampariello, and S. Lucidi, *A nonmonotone line search technique for Newton's method*, SIAM journal on Numerical Analysis, vol. 23, pp. 707–716, 1986.

[113] H. Zhang and W. W. Hager, *A nonmonotone line search technique and its application to unconstrained optimization*, SIAM journal on Optimization, vol. 14, pp. 1043–1056, 2004.

[114] J. Barzilai and J. M. Borwein, *Two-point step size gradient methods*, IMA journal of numerical analysis, vol. 8, pp. 141–148, 1988.

[115] E. G. Birgin, J. M. Martínez, and M. Raydan, *Nonmonotone spectral projected gradient methods on convex sets*, SIAM Journal on Optimization, vol. 10, no. 4, pp. 1196–1211, 2000.

[116] Q. Dong and Z. Wu, *A linear-time algorithm for solving the molecular distance geometry problem with exact inter-atomic distances*, Journal of Global Optimization, vol. 22, pp. 365–375, 2002.

[117] Q. Dong and Z. Wu, *A Geometric Build-up Algorithm for Solving the Molecular Distance Geometry Problem with Sparse Distance data*, Journal of Global Optimization, vol. 26, pp. 321–333, 2003.

[118] D. Wu and Z. Wu, *An Updated Geometric Build-up Algorithm for Solving the Molecular Distance Geometry Problems with Sparse Distance data*, Journal of Global Optimization, vol. 37, pp. 661–673, 2007.

[119] R. dos Santos Carvalho, C. Lavor, and F. Protti, *Extending the geometric build-up algorithm for the molecular distance geometry problem*, Information Processing Letters, vol. 108, no. 4, pp. 234–237, 2008.

[120] Cplex IBM ILOG, *V12. 1: User's Manual for CPLEX*, International Business Machines Corporation, vol. 46, no. 53, p. 157, 2009.

[121] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical analysis*. 2015.

[122] C. Lavor, J. Lee, A. Lee-St.John, L. Liberti, A. Mucherino, and M. Sviridenko, *Discretization orders for distance geometry problems*, Optimization Letters, vol. 6, pp. 783–796, 2012.

[123] A. Cassioli, O. Günlük, C. Lavor, and L. Liberti, *Discretization vertex orders in distance geometry*, Discrete Applied Mathematics, vol. 197, pp. 27–41, 2015.

[124] J. Omer and A. Mucherino, *The referenced vertex ordering problem: theory, applications, and solution methods*, Open Journal of Mathematical Optimization, vol. 2, pp. 1–29, 2021.

[125] L. Liberti, B. Masson, J. Lee, C. Lavor, and A. Mucherino, *On the number of realizations of certain Henneberg graphs arising in protein conformation*, Discrete Applied Mathematics, vol. 165, pp. 213–232, 2014.

[126] A. Mucherino, C. Lavor, T. Malliavin, L. Liberti, M. Nilges, and N. Maculan, *Influence of pruning devices on the solution of molecular distance geometry problems*, Experimental Algorithms: 10th International Symposium, SEA 2011, Kolimpari, Chania, Crete, Greece, May 5-7, 2011. Proceedings 10, pp. 206–217, 2011.

[127] B. Worley *et al.*, *Tuning interval Branch-and-Prune for protein structure determination*, Journal of Global Optimization, vol. 72, pp. 109–127, 2018.

[128] T. E. Malliavin, A. Mucherino, C. Lavor, and L. Liberti, *Systematic exploration of protein conformational space using a distance geometry approach*, Journal of Chemical Information and Modeling, vol. 59, no. 10, pp. 4486–4503, 2019.

[129] C. Lavor, R. Alves, W. Figueiredo, A. Petraglia, and N. Maculan, *Clifford algebra and the discretizable molecular distance geometry problem*, Advances in Applied Clifford Algebras, vol. 25, pp. 925–942, 2015.

[130] D. S. Gonçalves and A. Mucherino, *Discretization orders and efficient computation of cartesian coordinates for distance geometry*, Optimization Letters, vol. 8, pp. 2111–2125, 2014.

[131] I. Coope, *Reliable computation of the points of intersection of n spheres in $R^n$*, ANZIAM Journal, vol. 42, pp. C461–C477, 2000.

[132] J. Parsons, J. B. Holmes, J. M. Rojas, J. Tsai, and C. E. Strauss, *Practical conversion from torsion space to cartesian space for in silico protein synthesis*, Journal of computational chemistry, vol. 26, no. 10, pp. 1063–1068, 2005.

[133] J. K. Leman *et al.*, *Macromolecular modeling and design in rosetta: recent methods and frameworks*, Nature methods, vol. 17, no. 7, pp. 665–680, 2020.

[134] S. B. Hengeveld, M. Merabti, F. Pascale, and T. E. Malliavin, *A Study on the Covalent Geometry of Proteins and Its Impact on Distance Geometry*, International Conference on Geometric Science of Information, pp. 520–530, 2023.

[135] Z. Wang *et al.*, *Total body protein: a new cellular level mass and distribution prediction model*, The American Journal of Clinical Nutrition, vol. 78, no. 5, pp. 979–984, 2003.

[136] R. Dosi, A. Carusone, A. Chambery, V. Severino, A. Parente, and A. Di Maro, *Rapid primary structure determination of myoglobins by a complementary approach based on mass spectrometry and edman degradation*, Food chemistry, vol. 133, no. 4, pp. 1646–1652, 2012.

[137] J. B. Smith, *Peptide sequencing by edman degradation*, eLS, 2001.

[138] H. D. Niall, *Automated edman degradation: the protein sequenator*, Methods in enzymology, vol. 27, pp. 942–1010, 1973.

[139] K. Biemann, *Contributions of mass spectrometry to peptide and protein structure*, Biomedical & environmental mass spectrometry, vol. 16, no. 1-12, pp. 99–111, 1988.

[140] R. Zubarev, *Protein primary structure using orthogonal fragmentation techniques in fourier transform mass spectrometry*, Expert review of proteomics, vol. 3, no. 2, pp. 251–261, 2006.

[141] B. T. Chait, *Chemistry: mass spectrometry: bottom-up or*, Neuroscience, vol. 123, p. 931, 2004.

[142] J. A. Maier, C. Martinez, K. Kasavajhala, L. Wickstrom, K. E. Hauser, and C. Simmerling, *Ff14sb: improving the accuracy of protein side chain and backbone parameters from ff99sb*, Journal of Chemical Theory and Computation, vol. 11, pp. 3696–3713, 2015.

[143] A. Liwo, J. Lee, D. R. Ripoll, J. Pillardy, and H. A. Scheraga, *Protein structure prediction by global optimization of a potential energy function*, Proceedings of the National Academy of Sciences, vol. 96, no. 10, pp. 5482–5485, 1999.

[144] M. J. Dudek, K. Ramnarayan, and J. W. Ponder, *Protein structure prediction using a combination of sequence homology and global energy minimization: ii. energy functions*, Journal of Computational Chemistry, vol. 19, no. 5, pp. 548–573, 1998.

[145] B. Matthews, *X-ray crystallographic studies of proteins*, Annual review of physical chemistry, vol. 27, no. 1, pp. 493–493, 1976.

[146] M. Woolfson, *An Introduction to X-ray Crystallography* (An Introduction to X-ray Crystallography). 1997.

[147] M. Smyth and J. Martin, *X Ray crystallography*, Molecular Pathology, vol. 53, no. 1, p. 8, 2000.

[148] A. Ilari and C. Savino, *Protein structure determination by X-ray crystallography*, Bioinformatics: Data, Sequence Analysis and Evolution, pp. 63–87, 2008.

[149] M. Benvenuti and S. Mangani, *Crystallization of soluble proteins in vapor diffusion for x-ray crystallography*, Nature protocols, vol. 2, no. 7, pp. 1633–1651, 2007.

[150] S. H. Chung, D. L. Ma, and R. D. Braatz, *Optimal seeding in batch crystallization*, The Canadian journal of chemical engineering, vol. 77, no. 3, pp. 590–596, 1999.

[151] A. Srivastava, T. Nagai, A. Srivastava, O. Miyashita, and F. Tama, *Role of computational methods in going beyond x-ray crystallography to explore protein structure and dynamics*, International journal of molecular sciences, vol. 19, no. 11, p. 3401, 2018.

[152] Y. Cheng, N. Grigorieff, P. A. Penczek, and T. Walz, *A primer to single-particle cryo-electron microscopy*, Cell, vol. 161, no. 3, pp. 438–449, 2015.

[153] K. R. Vinothkumar and R. Henderson, *Single particle electron cryomicroscopy: trends, issues and future perspective*, Quarterly reviews of biophysics, vol. 49, e13, 2016.

[154] D. J. Mills, *Setting up and operating a cryo-em laboratory*, Quarterly Reviews of Biophysics, vol. 54, e2, 2021.

[155] E. Palovcak *et al.*, *A simple and robust procedure for preparing graphene-oxide cryo-EM grids*, Journal of structural biology, vol. 204, no. 1, pp. 80–84, 2018.

[156] G. Weissenberger, R. J. Henderikx, and P. J. Peters, *Understanding the invisible hands of sample preparation for cryo-EM*, Nature Methods, vol. 18, no. 5, pp. 463–471, 2021.

[157] X. Zhang *et al.*, *Near-atomic resolution using electron cryomicroscopy and single-particle reconstruction*, Proceedings of the National Academy of Sciences, vol. 105, no. 6, pp. 1867–1872, 2008.

[158] A. Punjani, H. Zhang, and D. J. Fleet, *Non-uniform refinement: adaptive regularization improves single-particle cryo-em reconstruction*, Nature methods, vol. 17, no. 12, pp. 1214–1221, 2020.

[159] A. Ben-Shem, N. Garreau de Loubresse, S. Melnikov, L. Jenner, G. Yusupova, and M. Yusupov, *The structure of the eukaryotic ribosome at 3.0 Å resolution*, Science, vol. 334, no. 6062, pp. 1524–1529, 2011.

[160] S. Safarian *et al.*, *Active site rearrangement and structural divergence in prokaryotic respiratory oxidases*, Science, vol. 366, no. 6461, pp. 100–104, 2019.

[161] E. D'Imprima and W. Kühlbrandt, *Current limitations to high-resolution structure determination by single-particle cryoem*, Quarterly Reviews of Biophysics, vol. 54, e4, 2021.

[162] P. Koukos and A. Bonvin, *Integrative modelling of biomolecular complexes*, Journal of molecular biology, vol. 432, no. 9, pp. 2861–2881, 2020.

[163] R. Puthenveetil and O. Vinogradova, *Solution nmr: a powerful tool for structural and functional studies of membrane proteins in reconstituted environments*, Journal of Biological Chemistry, vol. 294, no. 44, pp. 15 914–15 931, 2019.

[164] R. R. Ernst, G. Bodenhausen, and A. Wokaun, *Principles of nuclear magnetic resonance in one and two dimensions*, 1987.

[165] D. Neuhaus, *Nuclear overhauser effect*, eMagRes, 2007.

[166] P. J. Van Laarhoven, E. H. Aarts, P. J. van Laarhoven, and E. H. Aarts, *Simulated annealing*. 1987.

[167] D. Bertsimas and J. Tsitsiklis, *Simulated annealing*, Statistical science, vol. 8, no. 1, pp. 10–15, 1993.

[168] O. Sannikov, E. Ye, B. M. Pinto, P. Saunders, and N. Merbouh, *Introducing complex nmr mixtures at the undergraduate level: isomerization, separation and analysis of the diels-alder adducts from the reaction of methylcyclopentadiene and maleic anhydride (part ii)*, Journal of Laboratory Chemical Education, vol. 8, no. 3, pp. 39–80, 2020.

[169] J. Jumper *et al.*, *Highly accurate protein structure prediction with alphafold*, Nature, vol. 596, no. 7873, pp. 583–589, 2021.

[170] M. AlQuraishi, *Machine learning in protein structure prediction*, Current opinion in chemical biology, vol. 65, pp. 1–8, 2021.

[171] *Algorithms for Structure Comparison and Analysis: Homology Modelling of Proteins*, Encyclopedia of Bioinformatics and Computational Biology, S. Ranganathan, M. Gribskov, K. Nakai, and C. Schönbach, Eds., pp. 38–61, 2019.

[172] D. T. Jones, T. Singh, T. Kosciolek, and S. Tetchner, *Metapsicov: combining coevolution methods for accurate prediction of contacts and long range hydrogen bonding in proteins*, Bioinformatics, vol. 31, no. 7, pp. 999–1006, 2015.

[173] S. Wang, S. Sun, Z. Li, R. Zhang, and J. Xu, *Accurate de novo prediction of protein contact map by ultra-deep learning model*, PLoS computational biology, vol. 13, no. 1, e1005324, 2017.

[174] V. Golkov *et al.*, *Protein contact prediction from amino acid co-evolution using convolutional networks for graph-valued images*, Advances in Neural Information Processing Systems, vol. 29, 2016.

[175] A. W. Senior *et al.*, *Improved protein structure prediction using potentials from deep learning*, Nature, vol. 577, no. 7792, pp. 706–710, 2020.

[176] J. Xu, *Distance-based protein folding powered by deep learning*, Proceedings of the National Academy of Sciences, vol. 116, no. 34, pp. 16 856–16 865, 2019.

[177] M. AlQuraishi, *End-to-end differentiable learning of protein structure*, Cell systems, vol. 8, no. 4, pp. 292–301, 2019.

[178] A. v. Bondi, *Van der waals volumes and radii*, The Journal of physical chemistry, vol. 68, no. 3, pp. 441–451, 1964.

[179] B. Brooks, R. Bruccoleri, B. Olafson, D. States, S. Swaminathan, and M. Karplus, *Charmm: a program for macromolecular energy, minimization, and dynamics calculations*, Journal of Computational Chemistry, vol. 4, pp. 187–217, 2004.

[180] R. Engh and R. Huber, *Accurate bond and angle parameters for X-ray protein structure refinement*, Acta Crystallogr A, vol. 47, pp. 392–400, 1991.

[181] S. B. Hengeveld, T. E. Malliavin, L. Liberti, and A. Mucherino, *Collecting Data for Generating Distance Geometry Graphs for Protein Structure Determination*, Proceedings of Roadef23, 2023.

[182] K. Wuthrich, *NMR in Structural Biology: A Collection of Papers by Kurt Wüthrich.* World Scientific, 1995, vol. 5.

[183] Y. Shen and A. Bax, *Protein structural information derived from NMR chemical shift with the neural network program TALOS-N*, Methods Mol Biol, vol. 1260, pp. 17–32, 2015.

[184] L. Zhang, H. Ma, W. Qian, and H. Li, *Protein structure optimization using improved simulated annealing algorithm on a three-dimensional ab off-lattice model*, Computational Biology and Chemistry, vol. 85, p. 107 237, 2020.

[185] C. D. Schwieters, J. J. Kuszewski, N. Tjandra, and G. M. Clore, *The xplor-nih nmr molecular structure determination package*, Journal of magnetic resonance, vol. 160, no. 1, pp. 65–73, 2003.

[186] P. Güntert, *Automated nmr structure calculation with cyana*, Protein NMR techniques, pp. 353–378, 2004.

[187] M. Berjanskii *et al.*, *Genmr: a web server for rapid nmr-based protein structure determination*, Nucleic acids research, vol. 37, no. suppl_2, W670–W677, 2009.

[188] P. Güntert and L. Buchner, *Combined automated noe assignment and structure calculation with cyana*, Journal of biomolecular NMR, vol. 62, pp. 453–471, 2015.

[189] T. Sugiki, N. Kobayashi, and T. Fujiwara, *Modern technologies of solution nuclear magnetic resonance spectroscopy for three-dimensional structure determination of proteins open avenues for life scientists*, Computational and structural biotechnology journal, vol. 15, pp. 328–339, 2017.

[190] H. Berman *et al.*, *The Protein Data Bank*, Nucleic Acids Research, vol. 28, pp. 235–242, 2000.

[191] D. K. Agrafiotis, A. C. Gibbs, F. Zhu, S. Izrailev, and E. Martin, *Conformational sampling of bioactive molecules: a comparative study*, Journal of chemical information and modeling, vol. 47, no. 3, pp. 1067–1086, 2007.

[192] A. Mucherino, L. Liberti, C. Lavor, and N. Maculan, *Comparisons between an exact and a metaheuristic algorithm for the molecular distance geometry problem*, *in Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, 2009, pp. 333–340.

[193] W. Kabsch, *A solution for the best rotation to relate two sets of vectors*, Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography, vol. 32, no. 5, pp. 922–923, 1976.

[194] G. Wang and R. L. Dunbrack, *PISCES: a protein sequence culling server*, Bioinformatics, vol. 19, pp. 1589–1591, 2003.

[195] G. N. Ramachandran, C. Ramakrishnan, and V. Sasisekharan, *Stereochemistry of polypeptide chain configurations*, J Mol Biol., vol. 7, pp. 95–99, 1963.

[196] S. Khalife, T. Malliavin, and L. Liberti, *Secondary structure assignment of proteins in the absence of sequence information*, Bioinformatics Advances, vol. 1, no. 1, vbab038, 2021.

[197] A. Richards and J. P. How, *Aircraft trajectory planning with collision avoidance using mixed integer linear programming*, Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301), vol. 3, pp. 1936–1941, 2002.

[198] T. B. Moeslund, A. Hilton, and V. Krüger, *A survey of advances in vision-based human motion capture and analysis*, vol. 104, no. 2, pp. 90–126, 2006.

[199] M. Meredith, S. Maddock, *et al.*, *Motion capture file formats explained*, Department of Computer Science, University of Sheffield, vol. 211, pp. 241–244, 2001.

[200] M. Antonio, *Introducing the interaction distance in the context of distance geometry for human motions*, Chebyshevskii sbornik, vol. 20, no. 2 (70), pp. 273–283, 2019.

[201] J.-S. Monzani, P. Baerlocher, R. Boulic, and D. Thalmann, *Using an intermediate skeleton and inverse kinematics for motion retargeting*, Computer Graphics Forum, vol. 19, no. 3, pp. 11–19, 2000.

[202] C. Rocha, C. Tonetto, and A. Dias, *A comparison between the denavit–hartenberg and the screw-based methods used in kinematic modeling of robot manipulators*, Robotics and Computer-Integrated Manufacturing, vol. 27, no. 4, pp. 723–728, 2011.

[203] D. Botto and M. Gola, *Solution of the inverse kinematic problem of a robot manipulator with eulerian joints*, IFAC Proceedings Volumes, vol. 27, no. 14, pp. 375–379, 1994.

[204] D. A. Drexler, *Solution of the closed-loop inverse kinematics algorithm using the crank-nicolson method*, 2016 IEEE 14th International Symposium on Applied Machine Intelligence and Informatics (SAMI), pp. 351–356, 2016.

[205] D. Manocha and J. F. Canny, *Efficient inverse kinematics for general 6r manipulators*, IEEE transactions on robotics and automation, vol. 10, no. 5, pp. 648–657, 1994.

[206] F. Caccavale, B. Siciliano, and L. Villani, *The role of euler parameters in robot control*, Asian journal of control, vol. 1, no. 1, pp. 25–34, 1999.

[207] E. G. Hemingway and O. M. O'Reilly, *Perspectives on euler angle singularities, gimbal lock, and the orthogonality of applied forces and applied moments*, Multibody system dynamics, vol. 44, pp. 31–56, 2018.

[208] J. Diebel *et al.*, *Representing attitude: euler angles, unit quaternions, and rotation vectors*, Matrix, vol. 58, no. 15-16, pp. 1–35, 2006.

[209] G. G. Slabaugh, *Computing euler angles from a rotation matrix*, Retrieved on August, vol. 6, no. 2000, pp. 39–63, 1999.

[210] W. Maurel and D. Thalmann, *Human shoulder modeling including scapulo-thoracic constraint and joint sinus cones*, Computers & Graphics, vol. 24, no. 2, pp. 203–218, 2000.

[211] A. Wolodtschenko and T. Forner, *Prehistoric and early historic maps in europe: conception of cd-atlas*, E-perimetron, vol. 2, pp. 114–116, 2007.

[212] C. Rolland-May, *A Valuation Model of Subjective Spaces*, IFAC Proceedings Volumes, vol. 16, no. 13, pp. 375–380, 1983.

[213] S. Raveau, J. C. Muñoz, and L. de Grange, *A topological route choice model for metro*, Transportation Research Part A: Policy and Practice, vol. 45, pp. 138–147, 2011.

[214] S. B. Hengeveld, F. Plastria, A. Mucherino, and D. A. Pelta, *A linear program for points of interest relocation in adaptive maps*, International Conference on Geometric Science of Information, pp. 551–559, 2023.

[215] D. G. Luenberger, Y. Ye, *et al.*, *Linear and nonlinear programming*. 1984, vol. 2.

[216] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali, *Linear programming and network flows*. 2011.

[217] A. Mucherino, *Binmeta: a new java package for meta-heuristic searches*, International Conference on Large-Scale Scientific Computing, pp. 242–249, 2021.

[218] *Hamming distance*, *in Encyclopedia of Biometrics*, S. Z. Li and A. Jain, Eds. 2009, pp. 668–668.

[219] S. B. Hengeveld and A. Mucherino, *Variable Neighborhood Search in Hamming Space*, To appear in Lecture Notes in Computer Science, Proceedings of the 14th International Conference on Large-Scale Scientific Computations (LSSC23), 2023.

[220] D. S. Goncalves, C. Lavor, L. Liberti, and M. Souza, *A new algorithm for the k dmdgp subclass of distance geometry problems with exact distances*, Algorithmica, vol. 83, no. 8, pp. 2400–2426, 2021.

[221] N. T. Shaked, S. Messika, S. Dolev, and J. Rosen, *Optical solution for bounded NP-complete problems*, Applied Optics, vol. 46, no. 5, pp. 711–724, 2007.

[222] M. Oltean and O. Muntean, *Solving the subset-sum problem with a light-based device*, Natural Computing, vol. 8, no. 2, pp. 321–331, 2009.

[223] X.-Y. Xu *et al.*, *A scalable photonic computer solving the subset sum problem*, Science Advances, vol. 6, no. 5, 2020.

[224] T. Haist and W. Osten, *An Optical Solution For The Traveling Salesman Problem*, Optics Express, vol. 15, no. 16, pp. 10 473–10 482, 2007.

COLLEGE DOCTORAL BRETAGNE
MATHS, TELECOMS
INFORMATIQUE, SIGNAL
SYSTEMES, ELECTRONIQUE

Université de Rennes

**Titre :** Méthodes de géométrie d'extension des distances : pas seulement les distances (Une étude de trois applications, avec un focus sur la biologie structurale)

**Mot clés :** Géométrie des distances, Problème de Géométrie des Distances, Détermination de la Structure des Protéines, Repliement des Protéines, Discrétisation, Reciblage des Mouvements, Programmation Linéaire, Cartes Adaptatives

**Résumé :** Dans cette thèse, nous étudions différents aspects du Distance Geometry Problem (DGP). Le DGP est un problème inverse dans lequel un ensemble de distances par paire est inversé pour trouver une structure dans un espace euclidien, étant donné une certaine dimension $K$. Nous nous concentrons principalement sur l'application de la biologie structurelle, où nous pouvons exploiter les distances inter-atomiques pour calculer les structures des protéines. Pour cette application, nous sommes en mesure de discrétiser l'espace de recherche à l'aide de méthodes de branchement et de découpage. Nous étendons la méthode pour utiliser non seulement les informations de distance, mais aussi les angles de torsion. Nous présentons des expériences utilisant des données NMR réelles avec des résultats prometteurs. Ensuite, nous nous penchons sur la géométrie dynamique des distances en nous concentrant sur les mouvements humains, jetant ainsi les bases de futurs projets qui pourraient se concentrer sur la modélisation de la dynamique des protéines. En outre, nous discutons des cartes adaptatives, qui sont un autre exemple d'application de la DG dans laquelle nous pouvons exploiter plus que les informations de distance. Enfin, nous terminons par un bref résumé et une description des travaux en cours.

**Title:** Extending Distance Geometry methods: not only distances (A study of three applications, with a focus on structural biology)

**Keywords:** Distance Geometry, Distance Geometry Problem, Protein Structure Determination, Protein Folding, Discretization, Motion Retargeting, Linear Programming, Adaptive maps .

**Abstract:** In this thesis, we study different aspects of the Distance Geometry Problem (DGP). The DGP is an inverse problem where a set of pairwise distances is inverted to find a structure in a Euclidean space, given a certain dimension $K$. Our main focus lies on the application of structural biology, where we can exploit inter-atomic distances to compute protein structures. For this application, we are able to discretize the search space using branch-and-prune methods. We extend the method to not only use distance information, but also torsion angles. We present experiments using real NMR data with promising results. Next, we look at dynamical Distance Geometry with a main focus on human motions, laying the groundwork for future projects which could focus on modelling protein dynamics. Furthermore, we discuss adaptive maps, which is another example of an application of DG in which we can exploit more than just distance information. Finally, we end with a short summary of the thesis and a description of some ongoing work.