Journal of Artificial Intelligence Research 48 (2013) 953-1000 Submitted 09/13; published 12/13

A Constraint Solver for Flexible Protein Models

Federico Campeotto CAMPE8@NMSU.EDU
Dept. Computer Science, New Mexico State University
Depts. Math. & Computer Science, University of Udine

Alessandro Dal Palù ALESSANDRO.DALPALU@UNIPR.IT
Dept. Math. & Computer Science, University of Parma

Agostino Dovier AGOSTINO.DOVIER@UNIUD.IT
Dept. Math. & Computer Science, University of Udine

Ferdinando Fioretto FFIORETT@CS.NMSU.EDU
Dept. Computer Science, New Mexico State University
Depts. Math. & Computer Science, University of Udine

Enrico Pontelli EPONTELL@CS.NMSU.EDU
Dept. Computer Science, New Mexico State University

Abstract

This paper proposes the formalization and implementation of a novel class of constraints aimed at modeling problems related to placement of *multi-body* systems in the 3-dimensional space. Each multi-body is a system composed of body elements, connected by joint relationships and constrained by geometric properties. The emphasis of this investigation is the use of multi-body systems to model native conformations of protein structures—where each body represents an entity of the protein (e.g., an amino acid, a small peptide) and the geometric constraints are related to the spatial properties of the composing atoms. The paper explores the use of the proposed class of constraints to support a variety of different structural analysis of proteins, such as loop modeling and structure prediction.

The declarative nature of a constraint-based encoding provides elaboration tolerance and the ability to make use of any additional knowledge in the analysis studies. The filtering capabilities of the proposed constraints also allow to control the number of representative solutions that are withdrawn from the conformational space of the protein, by means of criteria driven by uniform distribution sampling principles. In this scenario it is possible to select the desired degree of precision and/or number of solutions. The filtering component automatically excludes configurations that violate the spatial and geometric properties of the composing multi-body system. The paper illustrates the implementation of a constraint solver based on the multi-body perspective and its empirical evaluation on protein structure analysis problems.

1. Introduction

Constraint Programming (CP) is a declarative programming methodology that has gained a predominant role in addressing large scale combinatorial and optimization problems. As a paradigm, CP provides the tools necessary to guide the modeling and resolution of search problems—in particular, it offers declarative problem modeling (in terms of variables and constraints), the ability to rapidly propagate the effects of search decisions, and flexible and efficient procedures to explore the search space of possible solutions. The field of CP

©2013 AI Access Foundation. All rights reserved.

has its roots on the seminal work by Sutherland (1963) in the Sketchpad system, and the successive efforts in systems like CONSTRAINTS (Sussmann & Steele, 1980) and ThingLab (Borning, 1981). Over the years, CP has become a paradigm of choice to address hard search problems, drawing and integrating ideas from diverse domains, like Artificial Intelligence and Operations Research (Rossi, van Beek, & Walsh, 2006). The declarative nature of CP enables fast and natural modeling of problems, facilitating not only development, but the rapid exploration of different models and resolution techniques (e.g., modeling choices, search heuristics).

In recent years, several research groups have started appreciating the potential of constraint programming within the realm of *Bioinformatics*. The field of Bioinformatics presents a number of open research problems that are grounded in critical exploration of combinatorial search space, highly suitable to be manipulated through constraint-based search. Constraint methodologies have been applied to analyze DNA sequences for instance, to locate Cis-regulatory elements (Guns, Sun, Marchal, & Nijssen, 2010), to DNA restriction maps construction (Yap & Chuan, 1993), and to pair-wise and multiple sequence alignment (Yang, 1998; Yap, 2001; Tsai, Huang, Yu, & Lu, 2004). Constraint methodologies have been applied to biological networks (Corblin, Trilling, & Fanchon, 2005; Larhlimi & Bockmayr, 2009; Ray, Soh, & Inoue, 2010; Gay, Fages, Martinez, & Soliman, 2011; Gebser, Schaub, Thiele, & Veber, 2011) and to other biological inference problems, such as Haplotype inference (Graca, Marques-Silva, Lynce, & Oliveira, 2011; Erdem & Ture, 2008), and phylogenetic inference (Erdem, 2011).

A particular area of Bioinformatics that has witnessed an extensive use of CP techniques is the domain of *structural biology*—i.e., the branch of molecular biology and biochemistry that deals with the molecular structure of nucleic acids and proteins, and how the structure affects behavior and functions. Constraint Programming has progressively gained a pivotal role in providing effective ways to explore the space of conformations of macromolecules, to address problems like secondary and tertiary structure prediction, flexibility, motif discovery, docking (Backofen, Will, & Bornberg-Bauer, 1999; Krippahl & Barahona, 2002; Thebault, de Givry, Schiex, & Gaspin, 2005; Dal Palù, Dovier, & Pontelli, 2007; Mann & Dal Palù, 2010; Shih & Hwang, 2011; Krippahl & Barahona, 2005; Dal Palù, Spyrakis, & Cozzini, 2012b; Chelvanayagam, Knecht, Jenny, Benner, & Gonnet, 1998; Yue & Dill, 2000). Two comprehensive surveys on the use of constraint-based methods in structural Bioinformatics have been recently proposed (Dal Palù, Dovier, Fogolari, & Pontelli, 2012a; Barahona & Krippahl, 2008).

Our focus in this work is on the use of constraint-based technology to support structural studies of *proteins*. Proteins are macromolecules of fundamental importance in the way they regulate vital functions in all biological processes. Their structural properties are critical in determining the biological functions of proteins (Skolnick, Fetrow, & Kolinski, 2000; Baker & Sali, 2001) and in investigating protein-protein interactions, which are central to virtually all cellular processes (Alberts, Johnson, Lewis, Raff, Roberts, & Walter, 2007). We refer to the *Protein Structure Prediction (PSP)* problem as the problem of determining the tertiary structure of a protein from knowledge of its primary structure and/or from knowledge of other structures (e.g., secondary structure components, templates from homologous proteins). The PSP problem is also often broken down to specialized classes of problems related to specific aspects of the tertiary structure of a protein, such as side-chain geometry

prediction (Dunbrack, 2002), loop modeling prediction (Go & Scheraga, 1970; Xiang, Soto, & Honig, 2002; Rufino, Donate, Canard, & Blundell, 1997; Soto, Fasnacht, Zhu, Forrest, & Honig, 2008), and protein flexibility investigation (Bennett & Huber, 1984).

All these classes of problems share common roots—the need to track the possible conformations of chains of amino acids. The variations of the problem relate to factors like the length of the chain being considered (from short peptides in the case of loop modeling to entire proteins in the general PSP case) and the diverse criteria employed in the selection of the solutions, as, for instance, the lowest basin of the effective energy surface, composed by the intra-molecular energy of the protein plus the solvation free energy (Karplus & Shakhnovich, 1992; Lazaridis, Archontis, & Karplus, 1995).

Modeling the variability of a protein chain involves many degrees of freedom which are needed to represent different protein conformations. Tracking this variability requires the exploration of a vast conformational space. Model simplifications can be adopted to reduce such computational cost, for instance backbone-only models represent only the backbone of proteins, the side-chain representation could be simplified to a single central point (centroid) describing its center of mass, or one can adopt approximated representation of the space though lattice models.

Nevertheless, even under strong simplifications, the search space remains intractable and prevents the use of brute-force search methods in the space of possible conformations (Crescenzi, Goldman, Papadimitriou, Piccolboni, & Yannakakis, 1998).

Constraint programming methodologies have found natural use in addressing PSP and related problems—where structural and chemical properties have been modeled in terms of constraints over spatial positions of atoms, transforming the search of conformations into a constraint satisfaction/optimization problem. The proposed approaches range from pure ab initio methods (Backofen et al., 1999; Dal Palù et al., 2007) to methods based on NMR data (Krippahl & Barahona, 1999) to methods based on fragments assembly (Dal Palù, Dovier, Fogolari, & Pontelli, 2010). In spite of all these efforts, the design of effective approaches to filter the space of conformations and lead to a feasible search remains a challenging and open problem.

In this work we present a constraint solver targeted at modeling a general class of protein structure studies. In particular our solution is suitable to address protein structure analysis study, requiring the generation of a set of unbiased sampled diverse conformations which satisfy certain given restraints. One of the unique features of the solution presented in this work is its capability to generate a uniformly distributed sampling of target protein regions among a given portion of Cartesian space and with selected granularity—accounting both for spatial and rotational properties.

We abstract the problem as a general *multi-body* system, where each composing body is constrained by means of geometric properties and it is related to other bodies through joint relationships. Each body can represent an entity in the protein, such as an individual amino acid or a small peptide (e.g., a protein fragment). Bodies relate to the spatial positions and organization of individual atoms composing it.

The view of the exploration of protein structures as multi-body systems suggests a number of different constraints, that can be used to model different classes of structural studies and applied to filter infeasible (or unlikely) conformations. We propose an investigation of several classes of constraints, in terms of both their theoretical properties and practical use

for filtering. Particular emphasis is given to the *Joined-Multibody* (JM) constraint, whose satisfaction we prove to be NP-complete. Realistic protein models require the assembly of hundreds of different body versions, making the problem intractable. We study an efficient approximated propagator, called *JM filtering* (JMf), that allows us to efficiently compute classes of solutions, partitioned by structural similarity and controlled tolerance for error. This perspective is novel and holds strong potential. The structural problems we are investigating are computationally intractable; the use of global constraints specifically designed to meet their needs enables a more effective exploration of the search space and a greater potential for effective approximations.

The multi-body model provides an interesting perspective in exploring the space of conformations—while the actual search operates on discrete sets of alternatives (e.g., sets of fragments), the filtering process avails of reasoning processes that operates with continuous domain; this allows the propagation and filtering to be effective.

The proposed multi-body constraints and filtering techniques constitute the core of the resolution engine of *FIASCO (Fragment-based Interactive Assembly for protein Structure prediction with Constraints)*, an efficient C++-based constraint solver. We demonstrate the flexibility and efficiency of FIASCO by using its engine to model and solve a class of problems derived from loop modeling instances. Throughout the paper we show the ability of FIASCO of providing a uniform and efficient modeling platform for studying different structural properties (that have been, so far, addressed only using significantly distinct methods and tools). The declarative nature of constraint-based methods supports a level of elaboration tolerance that is not offered by other frameworks for protein structure prediction, facilitating the integration of additional knowledge in guiding the studies (e.g., availability of information about secondary structure elements).

The rest of the paper is organized as follows. In Section 2, we provide a high-level background on the biological and chemical properties of proteins and review the most commonly used approaches to address structural studies. In Section 3, we develop the constraint framework for dealing with fragments and multi-body structures. Section 4 describes the implementation of the constraints and their propagation schemes in the FIASCO system. In Section 5 we report the experimental results from using FIASCO on a collection of benchmarks on loop modeling. Section 6 provides some concluding remarks.

A preliminary version of the research pursued in this paper was presented (Campeotto, Dal Palù, Dovier, Fioretto, & Pontelli, 2012). While the work of Campeotto et al. focused on one new class of constraints targeting the problem of loop closure, the work presented in this paper provides a comprehensive constraint system, focused on modeling structural protein properties and investigating different types of problems (e.g., structure prediction, studies of flexibility). The present manuscript includes also a more precise and detailed formalization and a more extensive experimentation and comparison.

2. Background, General Context, And Related Work

In this section we will briefly review some basic Biology notions, introduce the problems we are tackling in this paper and refer to a selection of the related literature.

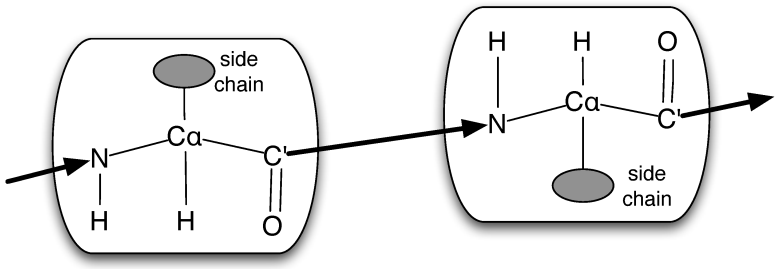

Figure 1: A schematic sequence of two amino acids showing the amino acid backbone and their side chains. The arrow from C' to N denotes the peptidic bond.

2.1 General Background

A protein is a molecule made of smaller building blocks, called *amino acids.* One amino acid can be connected to another one by a *peptidic* bond. Several amino acids can be pairwise connected into a linear chain that forms the whole protein. The *backbone* of a protein, as illustrated in Figure 1, is formed by a sequence of N–C_α–C' atoms contained in each amino acid. The backbone is rather flexible and it allows a large degree of freedom to the protein.

Each amino acid is characterized by a variable group of atoms that influences the specific physical and chemical properties. This group, named *side chain*, ranges from 1 to 18 atoms and connects to the C_α atom of each amino acid. There are 20 kinds of amino acids found in common eukaryotic organisms.

Proteins can be made of 10 up to $1,000$ amino acids, while an average globular protein is about 300 amino acids long. Each amino acid contains 7–24 atoms, therefore the number of atoms and arrangements in the space can grow very easily beyond any computational power. Since the beginning of protein simulation studies, different algorithms for exploring the conformations have been devised, such as molecular dynamics, local search, Monte Carlo, genetic algorithms, constraint approaches, as well as different geometric representations (Neumaier, 1997).

In the literature, several geometric models for proteins have been proposed. One choice that influences the quality and the complexity of computational approaches is the *number of points* that describe a single amino acid.

The simplest representation is the one where each amino acid is represented by one point, typically the C_α atom, given its robust geometric property: *the distance between the C_α atoms of two consecutive amino acids is preserved with a low variance (roughly $3.81\mathring{A}$).* Usually, volumetric constraints are enforced over those points, in order to simulate the average occupancy of each amino acid. This representation can be visualized as a chain of beads that can be moved in the space.

More refined representation models store some (or all) the points of the backbone, plus a centroid of mass (CG) that represents the whole side chain that connects to the C_α atom. In these models, each amino acid is described by different C_α–CG distances and CG volumes. The centroid is an approximation of the side-chain flexibility and allows for more refined energetic models, while the number of points to be taken care of is still low. In this paper

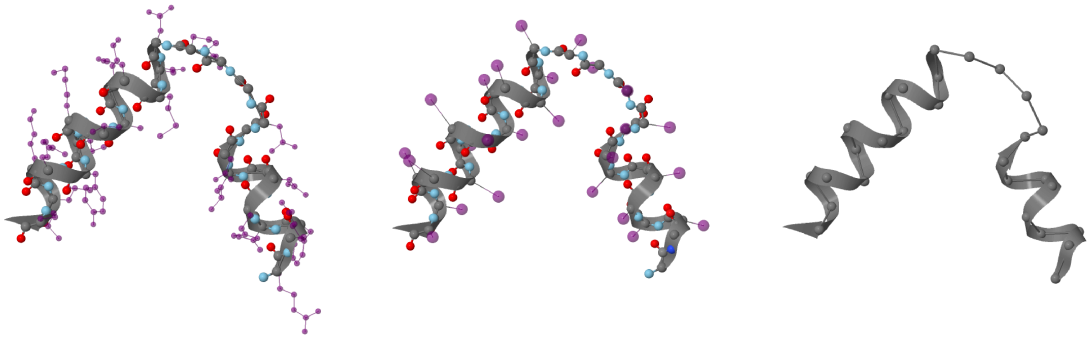

Figure 2: The native structure of "intact influenza virus M1 protein" (indexed as 1EA3 in the PDB) modeled as full atom, with the 5@ model, and with the simple C_α–C_α model (from left to right). The secondary structures (α-helices) are emphasized.

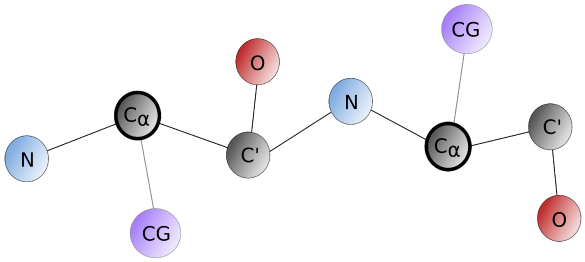

Figure 3: Amino acid concatenation in the 5@ model

we use a particular case of these simplified models, the "5@" model, described precisely below. This is a particular instance of *coarse-grained* protein models (Clementi, 2008; Shehu, 2010). At the end of the spectrum, each atom in the amino acid is represented by one point. This representation is the most accurate, and at the same time allows for the most accurate energetic considerations. The drawback is that the computational demand for handling backbone and side-chain flexibility increases significantly.

In Figure 2 we report three representations for the same protein.

In this paper we select the intermediate representation for amino acids where the atoms N, C_α, C' of the backbone and the centroid of the side chain (CG) are accounted for. We also include an oxygen (O) atom attached to the C' atom, because this atom together with the C' and N identifies a triangle that is chemically stable along the backbone and it is used for the assembly of amino acids (see below for a complete formalization). The position of the two H atoms in the backbone can be deduced by the position the other atoms and we will not deal with them explicitly. In conclusion, we deal with *5 at*omic elements per amino acid: the 4 atoms $NC_\alpha C'O$ and the centroid CG. We briefly refer to this representation as to the "5@" model. Figure 3 illustrates how these atoms are involved in the concatenation of two consecutive amino acids. Inter-atomic distances between consecutive atoms are fixed—due to their chemical bonds; thus, the differences between these structures are identified by the differences between the angles involved. It is common to find substructures of a

protein where consecutive amino acids are arranged according to repeated and characteristic patterns. This property is found in almost every protein; we refer to these typical patterns as *secondary structure elements*. The most common examples are α-helices and β-sheets (see Figure 2).

2.2 Context Of The Proposed Work

In this paper we present a tool for assembling and reasoning about amino acids in the space. As in other similar approaches (e.g, Simons, Kooperberg, Huang, & Baker, 1997), the system relies on a set of admissible elementary shapes (or *fragments*) that represents the spatial dictionary of arrangements for every part of a protein.

Each element of the dictionary is general enough to describe the specific atomic structure of either a single amino acid or a longer sequence (even hundreds of amino acids long). For each amino acid sequence, several alternative arrangements are expected to populate the database, so that to offer various hypothesis about the local shape of the sequence. The protein is partitioned into contiguous fragments that can be arranged according to one of the possible shapes recorded in the database.

A sequence of amino acids is free to rotate its bonds in the space (typically two degrees of freedom along the backbone and several others along the side chain); however, due to chemical properties and physical occupancy that are specific to the types of amino acids involved and the surrounding environment, some arrangements are impossible and/or unlikely. The core assumption in assembling approaches is to rely on a statistical database of arrangements to describe local and feasible behavior, in order to direct the search to candidates that have high probability and are energetically favorable. The presence of multiple candidate fragments for every part of the protein requires a combinatorial search among the possible choices that, once assembled together, leads to alternative putative configurations for the protein. The search process is in charge of verifying the feasibility of each assembly, since the combination of local arrangements could generate a non-feasible global shape, e.g., one that leads to a spatial clash between atoms from different fragments. If one (or more) fragment is described by one single arrangement, that part of the protein is rigidly imposed. This particular degenerate case can be exploited to describe rigid parts of the protein. A specific combination of fragment length and number of instances for each fragment determines the type of protein problem being modeled. We can range from complete backbone flexibility (fragments made of hundreds of choices for each amino acids) to secondary structure - loop models (interleaving of longer fragments modeling helices/β-strands and shorter fragments).

The library of fragments is usually derived from the content of the Protein Data Bank (PDB, `www.pdb.org`) that contains more than 96,000 protein structures. The design adopted in our study is parametric on the choice of the library of fragments to use. For example, our experiments use a library of fragments derived from a subset of the PDB known as top-500 (Lovell, Davis, Arendall, de Bakker, Word, Prisant, Richardson, & Richardson, 2003), which contains non redundant proteins and preserves statistical relevance. Alternative libraries of fragments can be obtained through the use of sophisticated protein database search algorithms, such as FREAD (Choi & Deane, 2010). We retrieve information depending on the specific amino acid sequence, since local properties greatly influence the typical

arrangements observed. Moreover, we build libraries for different sequences lengths h, even if for longer sequences the statistical coverage becomes weak. Nevertheless, Micheletti, Seno, and Maritan (2000) conjectured that a relatively small set of fragment shapes (a few dozens) of length 5, is able to describe virtually any protein. Handl, Knowles, Vernon, Baker, and Lovell (2012) demonstrate how the size and the structure of the search space is affected by the choice of the fragment length and how this can be used to optimize the search process. Similar considerations have been explored by others (Hegler, Lätzer, Shehu, Clementi, & Wolynes, 2009). Recent work show how to efficiently build such dictionaries (Fogolari, Corazza, Viglino, & Esposito, 2012). These models can be easily accommodated into our framework.

Each considered sequence is associated to several configurations of 5@ models, placed according to a standardized coordinate system. In this activity, we also consider the $C'O$ group of the preceding amino acid and the N atom of the following amino acid. This extra information is needed for fragments combination, assuming that the fragment will be connected by two peptidic bonds. Therefore, for a specific sequence, we store all the occurrences of

$$C'O \underbrace{NC_\alpha C'O}_{h \text{ times}} N$$

and relative positions. In order to reduce the impact of the specific properties of the database used, we cluster this set in such a way that if two fragments have a RMSD[1] less than a given threshold, just one of them is stored. For example, for length $h = 1$ and a RMSD threshold of .2Å, we can derive a fragment database of roughly 90 fragments per amino acid.

The CG information is added later using statistical considerations about side-chain mobility, that are not accounted for during the clustering described above (Fogolari, Esposito, Viglino, & Cattarinussi, 1996).

2.3 Protein Structure Prediction

In the protein structure prediction problem, the sequence of amino acids composing a protein (known as the *primary structure*) is given as input; the task is to predict the three dimensional (3D) shape (known as the *native conformation* or *tertiary structure*) of the protein under standard conditions.

The common assumption, based on Anfinsen's work (1973), is that the 3D structure which minimizes some given energy function modeling the atomic force fields, is the candidate that best approximates the functional state of a protein. In such setting, the choice of the number of atoms used to represent each amino acid controls the quality and the computational complexity.

Moreover, the spatial domains where the protein's "points" (e.g., atoms, centroids) can be placed have an impact on the type of algorithms and search that can be performed. The domain can be either *continuous*, often represented by floating point coordinates, or *discrete*, often derived from a discretization of the space based on a crystal lattice structure.

1. The Root Mean Square Deviation captures the overall similarity in space of corresponding atoms, by performing an optimal roto-translation to best overlap the two structures.

Once the geometric model has been determined, it is necessary to introduce an energy function, mostly based on the atoms considered and their distances. In the structure prediction problem, the energy function is used to assign a score to each geometrically feasible candidate; the candidate with the optimal score represents the solution of the prediction problem.

Let us briefly review some popular approaches to this problem, with a particular emphasis on solutions that rely on constraint programming technology.

The natural approach of investigating protein conformations through simulations of physical movements of atoms and molecules is, unfortunately, beyond the current computational capabilities (Jauch, Yeo, Kolatkar, & Clarke, 2007; Ben-David, Noivirt-Brik, Paz, Prilusky, Sussman, & Levy, 2009; Kinch, Yong Shi, Cong, Cheng, Liao, & Grishin, 2011). This has originated a variety of alternative approaches, many based on *comparative modeling*—i.e., small structures from related protein family members are used as templates to model the global structure of the protein of interest (Jones, 2006; Fujitsuka, Chikenji, & Takada, 2006; Simons et al., 1997; Lee, Kim, Joo, Kim, & Lee, 2004; Karplus, Karchin, Draper, Casper, Mandel-Gutfreund, Diekhans, & Source., 2003). In these methods, often referred to as *fragments assembly*, a protein structure is assembled using small protein subunits as templates that present relevant sequence similarities (*homologous affinity*) w.r.t. the target sequence.

In the literature, *Constraint Programming (CP)* techniques have shown their potential: the structural variability of a protein can be modeled as constraints, and *constraint solving* is performed in order to deduce the optimal structure (Backofen & Will, 2006; Barahona & Krippahl, 2008; Dal Palù, Dovier, & Fogolari, 2004; Dal Palù et al., 2010). CP has been used to provide approximated solutions for *ab-initio* lattice-based modeling of protein structures, by using local search and large neighboring search (Shmygelska & Hoos, 2005; Dotú, Cebrián, Van Hentenryck, & Clote, 2011); exact resolution of the problem on lattice spaces using CP, along with with clever symmetry breaking techniques, has also been investigated (Backofen & Will, 2006). These approaches solve a constraint optimization problem based on a simple energy function (HP). A more precise energy function has been used by Dal Palù et al. (2004, 2007), where information on secondary structures (i.e., α-helices, β-sheets) is also taken into consideration. Due to the approximation errors introduced by lattice discretization, these approaches do not scale to medium-size proteins. Off-lattice models, based on the idea of fragment assembly, and implemented using Constraint Logic Programming over Finite Domains, have been presented (Dal Palù et al., 2010; Dal Palù, Dovier, Fogolari, & Pontelli, 2011), and applied not only to structure prediction but also to other structural analysis problems. For instance, Dal Palù et al. (2012b) use this approach to generate sets of feasible conformations for studies of protein flexibility. The use of CP to analyze NMR data and the related problem of protein docking has also been investigated (Barahona & Krippahl, 2008).

In the context of *ab-initio* prediction, a recent work (Olson, Molloy, & Shehu, 2011) has shown that increasing the complexity of the conformational search space—by using a more refined fragment library—in combination with a sampling strategy, enhances the generation of near-native structure sets. The work of Shehu (2009) and Molloy, Saleh, and Shehu (2013) illustrates various enhancement the fragment-based assembly process leading to faster computations and an improved sampling of the conformation space—e.g., using

tree-based methods inspired from motion planning to guarantee progress towards minimal energy conformations while maintaining geometrically separate conformations. In terms of energy landscape, the native state has generally lower free energy than non-native structures, but it is extremely difficult to locate. Hence, a targeted conformational sampling may aid protein structure prediction in that different near-native structure can be used to guide the search; several schemes based on Monte Carlo movements in sampling conformation space through fragments assembly have been proposed (Shmygelska & Levitt, 2009; Xu & Zhang, 2012; Debartolo, Hocky, Wilde, Xu, Freed, & Sosnick, 2010). Methods based on non-uniform probabilistic mass functions (derived from previously generated decoys) have been proposed to aid in this problem (Simoncini, Berenger, Shrestha, & Zhang, 2012). Sampling, however, remains a great challenge for protein with complex topologies and/or large sizes (Kim, Blum, Bradley, & Baker, 2009; Shmygelska & Levitt, 2009).

It is widely accepted that proteins, in their native state, should be considered as dynamic entities instead of steady rigid structures. Indeed, in recent years the research focus has shifted towards prediction schemes that take into account the non-static nature of proteins, supported by recent observations based on magnetic resonance techniques. Processes such as enzyme catalysis, protein transport and antigen recognition rely on the ability of proteins to change conformation according to the required conditions. This dynamic nature can be visualized as a set of different structures that coexist at the same time. The generation of such sets that capture non-redundant structures (in pure geometric terms) is a great challenge (Kim et al., 2009). Robotics and inverse kinematics methods have been extensively explored both in sampling proteins' conformational space (Zhang & Kavraki, 2002; Cortes & Al-Bluwi, 2012) and for molecular simulations (Al-Bluwi, Simeon, & Cortes, 2012; Moll, Schwarz, & Kavraki, 2007; Noonan, O'Brien, & Snoeyink, 2005; Kirillova, Cortes, Stefaniu, & Simeon, 2008).

A motivation for our work is to provide the ability of generating a protein set that contains optimal and sub-optimal candidates, in order to capture dynamic information about the behavior of a protein. A desirable property is that the conformations returned in the pool are sufficiently diverse and uniformly distributed in the 3D space.

2.4 Protein Loop Modeling

The protein loop modeling problem is a restricted version of the structure prediction problem. We will use this problem as a working example in the remaining part of the paper. In this context, the protein structure is already partially defined, e.g., a large number of atoms are already placed in the space. Usually, this common scenario derives from an X-ray crystallography analysis, where the spatial resolution of atoms degenerates in presence of some regions of the protein that are exposed on the surface and presents an increased instability. Since a crystal contains several copies of a protein in order to perform the measurement, such regions appear as more fuzzy, and therefore the placement of atoms in these regions may be ambiguous. Usually, these regions, referred to as *loops*, are not involved in secondary structures, which are instead more stable. When dealing with homology modeling, the same protein found in another organism, typically shows some variations in the sequence due to evolution, especially in the loop regions, since they are less essential for protein stability and functionality. Starting from an homologous protein structure, usually

loops need to be recomputed with a specialized loop modeling approach and the use of minimization techniques.

The length of a loop is typically in the range of 2 to 20 amino acids; nevertheless, compared to secondary structures, the flexibility of loops produces very large, physically consistent, conformation search spaces. Constraints on the mutual positions and orientations (dihedral angles) of the loop atoms can be deduced and used to simplify the search. Such restrictions are defined as the *loop closure* constraints. In Figure 2, we have a (simple) possible scenario where two macro-structures (two helices) are connected by a loop. In this setting, we can assume to know the position of the two helices, while the loop atoms are to be determined.

A procedure for protein loop modeling typically consists of 3 phases: *sampling*, *filtering*, and *ranking* (Jamroz & Kolinski, 2010). Sampling is commonly based on a loop candidate generation, using dihedral angles sampled from structural databases (Felts, Gallicchio, Chekmarev, Paris, Friesner, & Levy, 2008), and subsequent candidate modification in order to satisfy the loop closure constraints. These conformations are checked w.r.t. the loop constraints and the geometries from the rest of the structure, and the loops that are detected as physically infeasible, e.g., causing steric clashes, are discarded by a filtering procedure.

Popular methods used for loop modeling include the *Cyclic Coordinate Descent (CCD)* method (Canutescu & Dunbrack, 2003), the algorithms based on inverse kinematics (Kolodny, Guibas, Levitt, & Koehl, 2005; Shehu & Kavraki, 2012), the *Self-Organizing (SOS)* algorithm (Liu, Zhu, Rassokhin, & Agrafiotis, 2009), which can simultaneously satisfy loop closure and steric clash restrictions by iteratively superimposing small fragments (amide and C_α) and adjusting distances between atoms, and the *Wriggling* method (Cahill, Cahill, & Cahill, 2003), that employs suitably designed Monte Carlo local moves to satisfy the loop closure constraints. Multi-method approaches have also been proposed—e.g., Lee, Lee, Park, Coutsias, and Seok (2010) propose a loop sampling method which combines fragment assembly and analytical loop closure, based on a set of torsion angles satisfying the imposed constraints. *Ab initio* methods (Rapp & Friesner, 1999; Fiser, Do, & Sali, 2000; Jacobson, Pincus, Rapp, Day, Honig, Shaw, & Friesner, 2004; Spassov, Flook, & Yan, 2008; Deane & Blundell, 2001; Felts et al., 2008; Xiang et al., 2002) and methods based on templates extracted from structural databases (Choi & Deane, 2010) have been explored.

Finally, a ranking step—e.g., based on statistical potential energy, like in DOPE (Shen & Sali, 2006), DFIRE (Zhou & Zhou, 2002), or the one proposed in Fogolari et al. (2007), is used to select the best loop candidates.

The sampling and filtering procedures should work together and direct the search towards structurally diverse and admissible loop conformations, in order to maximize the probability of including a candidate close to the native one and to reduce the time needed to analyze the candidates. Our work is motivated by the need of controlling the properties of the resulting set of candidates. In particular, we model structural diversity both in distance and orientation of the backbone and make the sampling phase guided by the loop constraints.

Fragment-based assembly methods have also been investigated in the context of loop modeling (Lee et al., 2010; Zhang & Hauser, 2013). Shehu and Kavraki (2012) review in great detail loop modeling techniques.

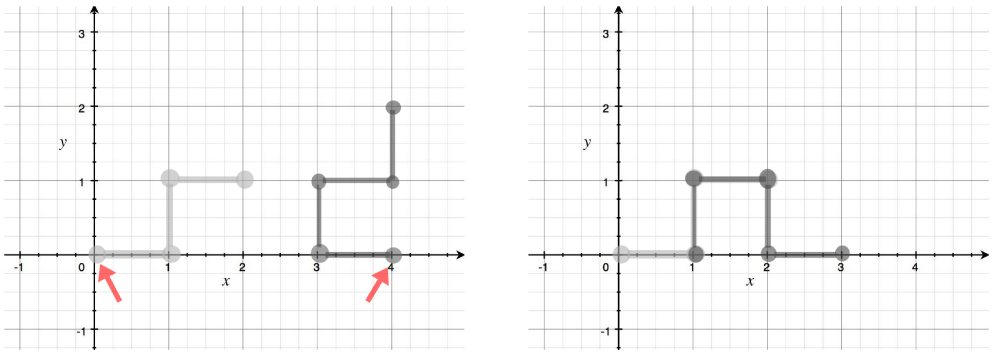

Figure 4: On the left: two fragments B_1 (light grey) and B_2 (dark grey) such that $\mathsf{points}(B_1) = ((0,0),(1,0),(1,1),(2,1))$ and $\mathsf{points}(B_2) = ((4,0),(3,0),(3,1),(4,1),(4,2))$. The arrows address their initial points. On the right: observe that by rotating B_2 of 90 degrees and then translating it by -3 units on the x-axis, the last three points of B_1 ($\mathsf{last}(B_1)$) and the first three points of B_2 ($\mathsf{first}(B_2)$) perfectly overlap. Thus, $\mathsf{end}(B_1) \frown \mathsf{front}(B_2)$.

3. Constraint Solving With 3D Fragments

We assume the reader to have familiarity with the basic principles of constraint programming and constraint satisfaction problems (CSP); the reader is referred, e.g., to the Handbook of Constraint Programming (Rossi et al., 2006). In this Section, we introduce the formalization of an effective solution to tackle practical applications concerning with the placement of 3D fragments. Such applications are described as combinatorial problems, modeled as a set of *variables*, representing the entities the problem deals with, and a set of *constraints*, representing the relationships among the entities. In the context of a constraint programming system, variables and constraints are adopted to provide a *solution* for the CSP, that is, an assignment to the variables that satisfies all the constraints. We extend this concept by enabling the constraint solver to find a *representative solution* for the CSP that satisfies some additional properties expressed among the variables of the whole solution set.

3.1 Some Terminology

A *fragment* B is composed of an ordered list of at least three (distinct) 3D points, denoted by $\mathsf{points}(B)$. The number of points of a fragment is referred to as its *length*. The front- and end-anchors of a fragment B, denoted by $\mathsf{front}(B)$ and $\mathsf{end}(B)$, are the two lists containing the first three and the last three points of $\mathsf{points}(B)$. With $B(i)$ we denote the i-th point of the fragment B. For two ordered lists of points $\vec{p}$ and $\vec{q}$, we write $\vec{p} \frown \vec{q}$ if they can be perfectly overlapped by a rigid coordinate translation and/or rotation (briefly, a *roto-translation*)—see Figure 4 (let us assume the z coordinate is 0 for all points and omitted for simplicity).

A non-empty set of fragments with the same length is called a *body*. A body can be used to model a set of possible shapes for a sequence of points. We say that a body has *length* k if each fragment it contains has length k.

A *multi-body* is a sequence $S_1, \ldots, S_n$ of bodies.

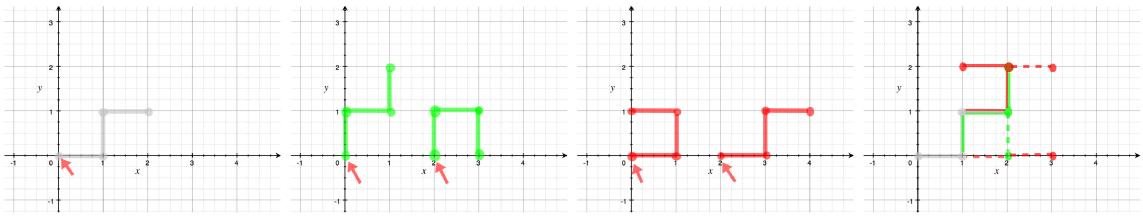

Figure 5: From left to right: the body S_1 composed by an unique fragment, and the bodies S_2 and S_3 composed by two fragments each. Arrows address the initial points of fragments. All the three bodies have length 4. $\vec{S} = S_1, S_2, S_3$ constitutes a multi-body. In the rightmost figure we report the spatial shapes associated to the four rigid bodies that can be obtained from the multi-body $\vec{S}$. One of them is identified by full lines, the other three by dashed lines. Observe that the rigid body identified by $((0,0),(1,0),(1,1),(2,1),(2,0),(3,0))$ can be obtained by a rotation of 180 degrees of the fragment $((2,0),(3,0),(3,1),(4,1))$ of S_2 on the x axis (flipping) and by a translation of -1 units on x and of $+1$ units on y. Observe moreover that the rigid body identified by $((0,0),(1,0),(1,1),(2,1),(2,0),(1,0))$ contains the same point $(1,0)$ twice.

Given a multibody $\vec{S} = S_1, \ldots, S_n$, a *rigid body* from $\vec{S}$ is a sequence of fragments $B_1, \ldots, B_n$, where $B_i \in S_i$ for $i = 1, \ldots, n$ and $\mathsf{end}(B_i) \frown \mathsf{front}(B_{i+1})$, for all $i = 1, \ldots, n-1$. A rigid body is uniquely identified by the sequence $B_1, \ldots, B_n$; however, when consecutive fragments are overlapped, the rigid body can be alternatively identified by a list of points that form a spatial shape. In Figure 5 we report examples of bodies, multi-bodies, and rigid bodies. As in the previous example, we assume that the z coordinate is 0 for all points.

Remark 3.1 (Working Example) *These concepts are related to the loop-modeling problem. Points are atoms. A fragment is a spatial shape of some atoms. If the last three atoms of one fragment overlap with the first three atoms of another fragment, we can join them. A body is a set of admissible shapes for a given list of atoms. A multi-body $S_1, \ldots, S_n$ is a sequence of these elements, corresponding to a sequence of atoms (of amino acids). The idea is that the last three atoms of a body S_i are the same as the first three of the successive body S_{i+1}. A rigid body is a possible complete shape of those atoms, provided the last three atoms of the fragment selected in the set S_i overlap with the first three atoms of the fragment selected in S_{i+1}.*

The overlapping points $\mathsf{end}(B_i)$ and $\mathsf{front}(B_{i+1})$ constitute the i-th *joint* of the rigid body. The number of rigid bodies that can be obtained from a single multi-body $S_1, \ldots, S_n$ is bounded by $\Pi_{i=1}^{n} |S_i|$. Figure 6 provides a schematic general representation of a rigid body.

A rigid body is defined by the overlap of joints, and relies on a chain of relative roto-translations of its fragments. Each points in $\mathsf{points}(B_i)$ is therefore positioned according to the (homogeneous) coordinate system associated to a fragment B_{i-1}. Note that once the reference system for B_1 is defined, the whole rigid body is completely positioned.[2] The

2. With the exception of the case where all points of a joint are collinear. Points $p_1, \ldots, p_n$, with $n \geq 3$ are *collinear* if the points $p_3, p_4, \ldots, p_n$ belongs to the straight line containing the two points p_1 and p_2.

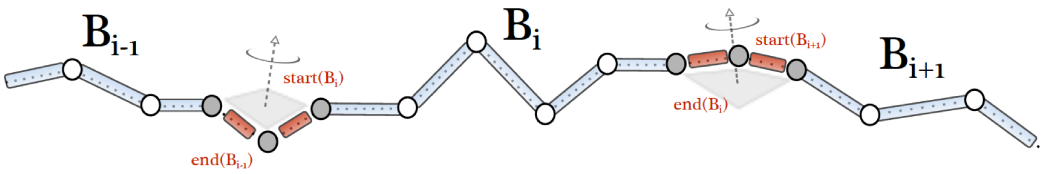

Figure 6: A schematic representation of a rigid body. The joints connecting two adjacent fragments are emphasized. The points in $\mathsf{points}(B)$ of each fragment are represented by circles. Each fragment extends from the first point of a joint to the last point of the successive joint.

relative positions of two consecutive fragments B_{i-1} and B_i of a rigid body ($2 \leq i \leq n$) can be defined by a transformation matrix $T_i \in \mathbb{R}^{4\times4}$. Each matrix depends on the standard Denavit-Hartenberg parameters (Hartenberg & Denavit, 1995) obtained from the start and end of the fragments—the reader is referred to the work of LaValle (2006) for details. We denote the product $T_1 \cdot T_2 \cdot \ldots \cdot T_i \cdot (x, y, z, 1)^T$ by $\mathcal{T}_i(x, y, z)$.

Let us analyze the first matrix T_1. The fragment B_1 can be forced to start in a given point and oriented in a given way; in this case the matrix T_1 defines the roto-translation of B_1 fulfilling these constraints. In the absence of such constraints, we assume that B_1 is *normalized* by T_1—i.e.,its first point is $(0, 0, 0)$, the second point is aligned along the z axis and the third belongs to the plane formed by the x and z axes. This orientation is referred as the reference system Γ_0.

For $i = 1, \ldots, n$, the coordinate system conversion (x', y', z'), for a point $(x, y, z) \in \mathsf{points}(B_i)$ into the coordinate system of B_1, is obtained by:

$$(x', y', z', 1)^T = T_1 \cdot T_2 \cdot \ldots \cdot T_i \cdot (x, y, z, 1)^T = \mathcal{T}_i(x, y, z) \tag{1}$$

Homogeneous transformations are such that the last value of a tuple is always 1.

In the rest of the paper, we focus on the 5@ model; however the proposed formalization and methods can be used also for other models, e.g., the C_α–C_α model. In the latter case, $\mathsf{points}(B_i)$ contains at least 3 amino acids, and the joints are guaranteed to be non-colinear, due to the chemical properties of the backbone. When combining C_α fragments, the specific rotational angles of the full-atom backbone are lost and a more imprecise multi-body assembly is produced.

A *fragment* is a body associated to a sequence of amino acids. A fragment for a sequence of $h \geq 1$ amino acids is described by a body of length $4h + 3$, modeling the concatenation of the atoms represented by the regular expression: $C'O(NC_\alpha C'O)^h N$. In such representation the first and last sequence of $C'ON$ atoms coincide with the *front-* and *end-anchor*, respectively, and are employed during the process of assembling consecutive fragments (i.e., they are used in the roto-translation).

A discretized $\mathbb{R}^3$ space can be represented as a regular lattice, composed of cubic cells with side length equal to a given parameter k. Each cell is referred to as a *3D voxel* (or, simply, *voxel*); we assume that each voxel receives a unique identifier. We denote with $\mathtt{voxel}(p, k)$ the identifier of the voxel that contains the 3D point p in the context of a discretization of the space using cubes with side length equal to k. This spatial quantization allows an efficient treatment of the approximated propagation required by some of the geometric constraints introduced in the following sections.

3.2 Variables And Domains

Let us now define the variables adopted to describe the entities of a problem with fragments. The *domain* of a variable V is the set of allowable values for V, and it will be denoted by D^V. To deal with fragments placements in the 3D space we adopt two distinct types of variables:

Finite Domain Variables (FDVs): The domain of a finite domain variable is a finite set of non negative integer numbers.

Point Variables (PVs): These variables will assume the coordinates of a 3D point in $\mathbb{R}^3$. Their domains are, initially, 3D boxes identified by two opposite vertices $\langle \min, \max \rangle$, as done in the discrete solver COLA (Dal Palù, Dovier, & Pontelli, 2005, 2007).

Remark 3.2 (Working Example) *Following Remark 3.1, FDVs are the identifiers of the various fragments in a body, while PVs are used to represent the 3D coordinates assigned to the various structural points (e.g., atoms, centroids) of interest for each molecule being considered. Clearly, the values of PVs will depend deterministically on the values of FDVs (and vice-versa).*

A variable is *assigned* if its domain contains a unique value; in the case of point variables, this happens if $\mathsf{D}^V = \langle \min, \max \rangle$ and $\min = \max$.

3.3 Constraints

In this section, we formalize the constraints that define the fragments placement, that can be used to describe Protein Structure problems in the context of fragment assembly.

3.3.1 Distance Constraints

Distance constraints model spatial properties of point variables operating in the 3D space. Point variables P and Q can be related by a distance constraint of the form

$$\|P - Q\| \;\; op \;\; d \tag{2}$$

where $\|\cdot\|$ is the Euclidean norm, $d \in \mathbb{R}^+$ and op is $\leq$ or $\geq$.

The built-in global constraint `alldistant` associates a minimal radius d_i to each point variable P_i $(i = 1, \ldots, n)$ and ensures that spheres surrounding each pair of point variables do not intersect:

$$\texttt{alldistant}(P_1, \ldots, P_n, d_1, \ldots, d_n), \tag{3}$$

This constraint is equivalent to the constraints $\|P_i - P_j\| \geq d_i + d_j$ for all $i, j \in \{1, \ldots, n\}, i < j$. It is used to avoid steric clashes among different atoms (and centroids), which have different volumes. Checking consistency of the `alldistant` constraint (given the domains of the variables P_i) is NP-complete (Dal Palù, Dovier, & Pontelli, 2010)—the proof is based on an encoding of the bin-packing problem using the `alldistant` constraint, and holds true even in this particular setting, where the point variables have intervals of $\mathbb{R}^3$ as domains.

Remark 3.3 (Working Example) *The* `alldistant` *constraint is introduced to avoid clashes when a rigid body is obtained from the multi-body $S_1, \ldots, S_n$. The distance constraints are*

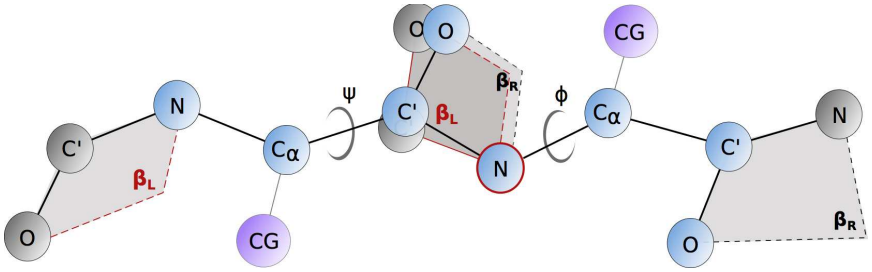

Figure 7: Fragments are assembled by overlapping the plane β_R, described by the rightmost C', O, N atoms of the first fragment (left), with the plane β_L, described by the leftmost C', O, N atoms of the second fragment (right), on the common nitrogen atom

useful when some extra information is known (e.g., one might have inferred by biological arguments that a pair of amino acid should stay within a certain distance).

3.3.2 Fragment Constraint

Fragment constraints relate finite domain variables and point variables. Let us assume we have a database F of fragments, where $F[i]$ represents the i-th fragment in the database. Thus, given an FDV variable V, $F[V]$ denotes the fragment indexed by V when V is instantiated. The fragments are stored in F as an ordered list of 3D points.

Given a list of point variables $\vec{P}$, the constraint:

$$\texttt{fragment}(V, \vec{P}, F) \tag{4}$$

states that there exists a roto-translation $\texttt{Rot}$ such that $\vec{P} = \texttt{Rot} \cdot F[V]$—namely, if $V = i$ then the list of points $\vec{P}$ should take the form of the fragment $F[i]$. For simplicity, we will omit the database F when clear from the context. Intuitively, these constraints ensure that any fragment choice will reproduce the correct shape for the associated 3D point, regardless of the space orientation of the fragment. The orientation is determined by the joined multi-body constraint presented in a following section.

3.3.3 Centroid Constraint

The centroid constraint enforces a relation among four PVs. Intuitively, the first three of them are associated to the atoms N, C_α, C' of an amino acid and the fourth is related to the centroid CG. The constraint is parametric w.r.t. the type a of an amino acid and deterministically establishes the position of CG depending on the position of the other points:

$$\texttt{centroid}(P_N, P_{C_\alpha}, P_{C'}, P_{\mathsf{CG}}, a) \tag{5}$$

In Figure 7 the centroids are displayed along the backbone as purple circles and labeled "CG." This constraint can be used when the database of fragment contains only full backbone information. The centroid information is used in place of the missing full-atom side chain. The side-chain centroid is computed by taking into account the average C_α-side-chain center of mass distance, the average bend angle formed by the side-chain center-of-mass-C_α-C', and the torsional angle formed by the N-C_α-C'-side-center of mass (Fogolari et al.,

1996). This abstraction allows us to reduce the number of fragments to consider, removing fragments that would geometrically conflict with the position of the CG. Consider that a single side chain may have up to 100 main configurations (rotamers).

3.3.4 Table Constraint

This constraint is used to restrict the assignments of a set of FDVs (representing fragments) to specific tuples of choices. This is useful when modeling a specific local and collaborative behavior that involves more than one fragment; for example, this happens when modeling a secondary structure multiple arrangements of underlying amino acids and/or when specific approximation strategies are employed.

Let F be a set of k-tuples of integer values and $\vec{V}$ a k-tuple of FDVs. A table (or combinatorial) constraint, of the form

$$\texttt{table}(\vec{V}, F) \tag{6}$$

requires that the list of variables $\vec{V}$ assumes values restricted to the tuples listed in F, i.e., there exists $t \in F$ such that $\vec{V}[i] = t[i]$, with i in $0, \ldots, k-1$.

Remark 3.4 (Working Example) *Going back to the loop-modeling problem, the role of the* `fragment` *constraint is evident: it relates the (IDs of the) selected fragments of a multi-body with the 3D positions of the various atoms involved. The* `centroid` *constraint is instead introduced to add the position of the centroid that represents the side chain in the 5@ representation.* `table` *constraint is a common constraint in constraint languages and it is useful when some info on consecutive fragments in a rigid body is known due to external knowledge.*

3.3.5 Joined Multibody Constraint

The Joined Multibody (JM) constraint enforces a relation over a list of FDVs encoding a multibody. It limits the spatial domains of the various fragments composing the multibody in order to retain those fragments that assemble properly and that do not compenetrate. The *joined-multibody (JM) constraint* is described by a tuple: $J = \langle \vec{S}, \vec{V}, \vec{\mathcal{A}}, \vec{\mathcal{E}}, \delta \rangle$, where:

- $\vec{S} = S_1, \ldots, S_n$ is a multi-body. Let $\mathcal{B} = \{B_1, \ldots, B_k\}$ be the set of all fragments in $\vec{S}$, i.e., $\mathcal{B} = \bigcup_{i=1}^{n} S_i$.
- $\vec{V} = V_1, \ldots, V_n$ is a list of FDVs, with domains $D^{V_i} = \{j \,:\, B_j \in S_i\}$.
- $\vec{\mathcal{A}} = \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$, and $\vec{\mathcal{E}} = \mathcal{E}_1, \ldots, \mathcal{E}_{3n}$ are lists of sets of 3D points such that:
  - $\mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{A}_3$ is the set of admissible points for $\mathsf{front}(B)$, with $B \in S_1$;
  - $\mathcal{E}_{3i-2} \times \mathcal{E}_{3i-1} \times \mathcal{E}_{3i}$ is the set of admissible points for $\mathsf{end}(B)$, with $B \in S_i$, $i = 1, \ldots, n$;
- δ is a constant, used to express a minimal distance constraint between different point.

A *solution* for the JM constraint J is an assignment $\sigma : \vec{V} \longrightarrow \{1, \ldots, |\mathcal{B}|\}$ s.t. there exist matrices $T_1, \ldots, T_n$ (used in $\mathcal{T}$) with the following properties:

Domain: For all $i = 1, \ldots, n$, $\sigma(V_i) \in \mathsf{D}^{V_i}$.

Joint: For all $i = 1, \ldots, n-1$, let $(a^1, a^2, a^3) = \mathsf{end}(B_{\sigma(V_i)})$ and $(b^1, b^2, b^3) = \mathsf{front}(B_{\sigma(V_{i+1})})$, then it holds that (for $j = 1, 2, 3$):

$$\mathcal{T}_i(a_x^j, a_y^j, a_z^j) = \mathcal{T}_{i+1}(b_x^j, b_y^j, b_z^j)$$

Spatial Domain: Let $(a^1, a^2, a^3) = \mathsf{front}(B_{\sigma(V_1)})$, then $T_1 \cdot a^j \in \mathcal{A}_j \times \{1\}$.[3] For all $i = 1, \ldots, n$, let $(e^1, e^2, e^3) = \mathsf{end}(B_{\sigma(V_i)})$ then

$$\mathcal{T}_i(e_x^j, e_y^j, e_z^j) \in \mathcal{E}_{3(i-1)+j} \times \{1\}$$

where $1 \leq j \leq 3$ and $T_2, \ldots, T_i$ (in $\mathcal{T}_i$) are the matrices that overlap $\mathsf{end}(B_{\sigma(V_{i-1})})$ and $\mathsf{front}(B_{\sigma(V_i)})$

Minimal Distance: For all $j, \ell = 1, \ldots, n$, $j < \ell$, and for all points $a \in \mathsf{points}(B_{\sigma(V_j)})$ and $b \in \mathsf{points}(B_{\sigma(V_\ell)})$, it holds that:[4]

$$\|\mathcal{T}_j(a_x, a_y, a_z) - \mathcal{T}_\ell(b_x, b_y, b_z)\| \geq \delta$$

It has been proved that establishing *consistency*—i.e., existence of a solution—of JM constraints is NP-complete (Campeotto et al., 2012). We have also proved that it remains NP complete even assuming that all all the fragments of the problem have the same three atoms with the same spatial position, and that the same holds for the last three atoms (of course fragments are allowed to contain more than three atoms otherwise the problem is trivial). The proof is reported in `www.cs.nmsu.edu/fiasco/`.

Remark 3.5 (Working Example) *The JM constraint contains exactly all the ingredients needed for modeling a loop problem. We have a multi-body $\vec{S}$, and the corresponding FDs $\vec{V}$, we have a set of possible 3D points where the loop starts $\vec{\mathcal{A}}$ and a set of possible 3D points where the loop ends $\vec{\mathcal{E}}$ and a weak version of the alldistant constraint between pair of atoms that avoid clashes, the solutions are the (non clashing) rigid bodies that starts in $\vec{\mathcal{A}}$ and ends in $\vec{\mathcal{E}}$.*

Let us observe that the JM constraint does not explicitly forbid spatial positions to PVs variables (save for the first three and the last three points of the loop). However, these additional constraints can be explicitly required during domain definition of the PVs variables used for the encoding.

Remark 3.6 *The choice of using three points of overlap resembles the method proposed by Kolodny, Guibas, Levitt, and Koehl (2005). On the other hand, we should observe that it is only a technical exercise to modify the JM constraints and so that they allow a parametric overlap between contiguous fragments.*

4. The FIASCO Constraint Solver

We present the overall structure and implementation of a hybrid constraint solver capable of handling the classes of constraints described in the previous section.

4.1 Constraint Solving

A distinctive feature of FIASCO is the possibility to handle continuous domains at the cost of keeping a discrete library of choices (finite domain variables). The handling of fragments allows us to reason about spatial properties in a more efficient and descriptive way than the pure 3D domain modeling adopted in previous proposals. Moreover, FIASCO allows

3. The product $\times\{1\}$ is necessary as we use homogeneous coordinates.
4. Let us observe that this is a weak form of the `alldistant` constraint where different distances for each point are allowed. It is, in a sense, closer to the `alldifferent` constraint.

the solver to *uniformly* sample the search space by means of a spatial equivalence relation that is used to control the tradeoff between accuracy and efficiency. This is particularly effective when the finite domains are heavily populated, and is a critical component to model real-world problems.

The constraint solver builds on the classical prop-labeling tree exploration where constraint propagation phases are interleaved with non-deterministic branching phases used to explore different value assignments to variables (Apt, 2009). The solver is able to handle both point variables and finite domain variables—this is the reason why we refer to it as an *hybrid* solver. In particular, the assignments to finite domain variables guide the search; their values imply assignments of the point variables, that in turn may propagate and reduce the domains of both point variables and finite domain variables. Moreover, the "propagation" technique implemented for the JM constraint is not a classical filtering technique—it is an approximated technique that we describe later.

The presence of point variables allows, in principle, an infinite number of domain values in $\mathbb{R}^3$. However, we noted that the information carried by assembling fragments (encoded by finite domain variables) is much more informative than any complex and demanding model for 3D continuous space (e.g., Oct-trees, CSG, no-goods). In particular, the direct kinematics encoded by a JM constraint is able to efficiently identify a set of admissible regions of a point variable in a fast, approximated, and controlled way. Therefore, the point variables can be seen as an internal aid to propagation. These variables are updated during the JM propagation phase and can interact with the JM propagator to prune the corresponding fragment variables. Distance constraints on point variables are included in a standard AC3 propagation loop for domains updates.

The other aspect that extends the classical solver structure is the capability of controlling the amount of the search tree to be explored. The search tree contains a large number of branches that are very similar, from the point of view of the geometric distance between corresponding point variables. The goal is to produce a subset of feasible solutions that exhibit significant 3D differences between themselves. This is accomplished by introducing the possibility to explore a subtree of a given depth, by enumerating a specific and limited number of branches, rather than following the standard recursion of propagation and expansion. To achieve this behavior, it is necessary to selectively interfere with the standard recursive call to the solver, and implement a non-deterministic assignment of partial tuples of finite domain variables. This resembles the implementation of a *table* constraint, which is dynamically created during the search. This strategy allows us to significantly reduce the number of branches explored in the subtree, and produces significant results when the selection of the branches is controlled by an adequate partitioning function. In this work, we propose an effective partitioning function based on a measure of 3D similarity for point variables; this is used to direct the search along specific branches of controlled depth that are adequately "separated" by the partitioning function. This is practically realized by introducing a form of look-ahead, controlled by the JM propagator, that returns a set of partial assignments as well as the filtered domains for the finite domain variables.

4.1.1 The Hybrid Solver

Algorithm 1 **search**$(\vec{V}, \vec{P}, \vec{D}, \mathcal{C}, \ell)$

Require: $\vec{V}, \vec{P}, \vec{D}, \mathcal{C}, \ell$
1: **if** $\ell > |\vec{V}|$ **then**
2: **output** $(\vec{P})$
3: **return**
4: **end if**
5: **for each** fragment index f **in** D_{v_ℓ} **do**
6: **if AC-3**$(\mathcal{C} \cup \{v_\ell = f\}, \vec{V}, \vec{P}, \vec{D})$ **then**
7: $T^{n \times m} \leftarrow$ **get_table_from_JM**()
8: **if** $n > 0$ **then**
9: **Non-deterministically select** i **in** $1..n$
10: **for** $j = 1..m$ **do**
11: $\mathcal{C} \leftarrow \mathcal{C} \cup \{v_{\ell+j} = T[i][j]\}$
12: **end for**
13: **search**$(\vec{V}, \vec{P}, \vec{D}, \mathcal{C}, \ell + m)$
14: **else**
15: **search**$(\vec{V}, \vec{P}, \vec{D}, \mathcal{C}, \ell + 1)$
16: **end if**
17: **end if**
18: **end for**

The general structure of the solver is highlighted in Algorithm 1. The solver is designed to process a list $\vec{V} = v_1, \ldots, v_n$ of finite domains variables, together with the domains $D_{v_1}, \ldots, D_{v_n}$ for them. Intuitively, each domain is a set of indices for the set of fragments. Moreover, the solver receives a list $\vec{P} = p_1, \ldots, p_{5n}$ of $5 * n$ point variables, where the variables $p_{4*i}, \ldots, p_{4*i+4}$ are related to the fragment in the domain D_{v_i}. Each point variable p_j has, in turn, a spatial domain D_{p_j}. $\mathcal{C}$ represents the constraints between elements of $\vec{V}$ and $\vec{P}$. Finally, the solver receives also as input the "current" level ℓ in the exploration of the search tree (set to 1 the first time the procedure is called). For the sake of simplicity, the choice of variables to be assigned is based on their ordering in the input list (more sophisticated selection strategies can be easily introduced). When we enter the level ℓ, we assume that the variables $v_1, \ldots, v_{\ell-1}$ have already been assigned.

Let us briefly describe the algorithm. If all the variables in $\vec{V}$ have already been assigned (lines 1–4), then the search algorithm terminates and returns the computed solution, represented by the values assigned to the variables $\vec{P}$. Otherwise, we non-deterministically select a fragment index in the domain of the variable v_ℓ and assign it to the variable. Lines 6–7 indicate the execution of a standard constraint propagation step (using AC-3). If the propagation step fails, then we assume that another non-deterministic choice is made, if possible. Every reference to a non-deterministic choice in the algorithm corresponds to the creation of a choice-point that will be the target of backtracking in case of failure (for simplicity, we assume chronological backtracking). If it succeeds, leading to a possible reduction of the domains $\vec{D}$, then the computation will proceed. A table constraint might be produced during the propagation of the JM constraint in the AC-3 procedure (see below for details). If this is the case (lines 8–9), some (m) variables are non-deterministically assigned with the values in the table (lines 9–12), and the search continues with m less variables to be

assigned (line 13). If this is not the case, then the search will continue with only one less variable (v_ℓ) to be assigned (line 15).

A peculiar feature of our constraint solver (not reported in the abstract algorithm just defined) is that it can be used to avoid the search of solutions "too similar" to each others. Let us assume that the 3D space is partitioned in cubic voxels of size kÅ. Then, given a list of FDVs $\vec{V}$ and a list of PVs $\vec{P}$, the user can state:

$$\texttt{uniqueseq}(\vec{V}, \vec{P}, k) \tag{7}$$

This constraint forces the solver to prune the search tree in the following way. Given a partial assignment σ, let $v \in \vec{V}$ be the variable to be assigned at the next step and $p_1, \ldots, p_h \in \vec{P}$ the PVs to be consequently instantiated. The constraint ensures that for any two assignments σ_1, σ_2 extending σ to $v, p_1, \ldots, p_h$ it holds that there exists at least one $i \in \{1, \ldots, h\}$ such that $\sigma_1(p_i)$ and $\sigma_2(p_i)$ do not belong to the same voxel.

4.2 Constraint Propagation

In this section, we discuss the propagation rules associated to the various constraints introduced in Section 3.3; these are applied within the call to the AC-3 procedure (line 6 of Algorithm 1). The constraint propagation is used to reduce the domain size of the PVs and FDVs, ensuring constraint consistency. AC-3 is a standard implementation of a fixpoint propagation loop (Apt, 2009; Rossi et al., 2006).

4.2.1 Joined Multibody Constraint

The JM constraint is a complex constraint that is triggered when the leftmost points involved in the constraint (anchors) are instantiated. The JM propagation (JMf) is based on the analysis of the distribution in the space of the points involved. The goal of the propagation is to reduce the domains of the FDVs through the identification of those fragments that cannot contribute to the generation of a rigid body that is compatible with the corresponding Point Variable domains. This can be viewed as a form of hyper-arc consistency over a set of fragments. Moreover, due to complexity and precision considerations, this propagator is approximated by the use of a spatial equivalence relation ($\sim$), that identifies classes of tuples of fragments; these classes have the property to be spatially "different" from one another.

This allows a compact handling of the combinatorics of the multi-body, while a controlled error threshold allows us to select the precision of the filtering. The equivalence relation captures those rigid bodies that are geometrically similar, allowing the search to "compact" small differences among them.

The JMf algorithm receives as input a JM-constraint $\langle \vec{S}, \vec{V}, \vec{\mathcal{A}}, \vec{\mathcal{E}}, \delta \rangle$, along with

- A set $\mathcal{G}$ of points that are not available for the placement of bodies, and
- The equivalence relation $\sim$.

For the sake of readability, we assume that the domain information for variables are available. The algorithm builds a table constraint $\texttt{table}\,(\vec{V}, \mathsf{Tab})$. In this process, the algorithm makes use of a function ρ (lines 7 and 8); this function takes as input two lists $\vec{a}$ and $\vec{b}$ of 3D points, and computes the homogeneous transformation to overlap $\vec{b}$ on $\vec{a}$. A call to

Algorithm 2 The JMf algorithm.

Require: $\vec{S}, \vec{V}, \vec{\mathcal{A}}, \vec{\mathcal{E}}, \delta, \mathcal{G}, \sim$
Ensure: Tab
1: $n \leftarrow |\vec{V}|; \mathsf{Tab} = \emptyset$
2: $\mathcal{R}_1 \leftarrow \left\{ (B, T_1) \,\middle|\, B \in S_1, \exists T_1 \left(\begin{array}{l} T_1 \cdot \mathsf{start}(B) \in \mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{A}_3 \ \wedge \\ T_1 \cdot \mathsf{end}(B) \in \mathcal{E}_1 \times \mathcal{E}_2 \times \mathcal{E}_3 \quad \wedge \\ \forall p \in \mathsf{points}(B). \forall q \in \mathcal{G}. \, \|(T_1 \cdot p) - q\| \geq \delta \wedge \\ \forall c \in \mathcal{C} \text{ involving } p. \mathsf{consistent}(c)) \end{array} \right) \right\}$
3: $\mathcal{P}_1 \leftarrow \{T_1 \cdot \mathsf{end}(B) \mid (B, T_1) \in \mathcal{R}_1\}$
4: **for each** $i = 2, \ldots, n$ **do**
5: $\quad \mathcal{P}_i = \emptyset$; $\mathcal{R}_i = \emptyset$;
6: $\quad$ **for each** $E \in \mathcal{P}_{i-1}/\sim$ **do**
7: $\quad\quad \mathcal{R}_i \leftarrow \mathcal{R}_i \cup \left\{ B \in S_i \,\middle|\, \begin{array}{l} T = \rho(E, \mathsf{start}(B)) \ \wedge \ T \neq \texttt{fail} \ \wedge \\ T \cdot \mathsf{end}(B) \in \mathcal{E}_{3i-2} \times \mathcal{E}_{3i-1} \times \mathcal{E}_{3i} \quad \wedge \\ \forall p \in \mathsf{points}(B). \forall q \in \mathcal{G}. \, \|(T \cdot p) - q\| \geq \delta \wedge \\ \forall c \in \mathcal{C} \text{ involving } p. \mathsf{consistent}(c)) \end{array} \right\}$
8: $\quad\quad \mathcal{P}_i \leftarrow \{\rho(E, \mathsf{start}(B)) \cdot \mathsf{end}(B) \mid B \in \mathcal{R}_i\}$
9: $\quad$ **end for**
10: $\quad$ **compute** $\mathcal{P}_i/\sim$ **and filter** $\mathcal{R}_i$ **accordingly**
11: **end for**
12: **for each** representative L of $\mathcal{P}_n/\sim$ **do**
13: $\quad \mathsf{Tab} = \mathsf{Tab} \cup \eta(L)$
14: **end for**

this function will fail if $\vec{a} \not\frown \vec{b}$. For simplicity, the fourth component (always 1) of the homogeneous transformation is not explicitly reported in the algorithm.

For $i = 1, \ldots, n = |\vec{V}|$, the algorithm computes the sets $\mathcal{R}_i$ and $\mathcal{P}_i$, that will respectively contain the fragments from S_i that can still lead to a solution, and the corresponding allowed 3D positions of their end-points. For each fragment $B \in \mathcal{R}_{i+1}$ we denote with $\mathsf{parent}(B)$ the set of fragments $B' \in \mathcal{R}_i$ such that $\mathsf{end}(B') \frown \mathsf{front}(B)$ via ρ. For each fragment B, we denote with $\mathsf{label}(B)$ the corresponding FD value associated.

In computing/updating $\mathcal{R}_i$ and $\mathcal{P}_i$, only fragments that have end-anchors contained in the bounds $\mathcal{E}_{3i-2}, \mathcal{E}_{3i-1}, \mathcal{E}_{3i}$ are kept. Fragments that would cause points to collapse—i.e., due to a distance smaller than δ from previously placed points—are filtered out (lines 2 and 7). Moreover, the spatial positions of the points of the first fragment are validated against $\mathcal{A}$ (line 2); finally, we enforce the consistency check of each constraint $c \in \mathcal{C}$ involving points in $\mathsf{points(B)} \in S_i$ to retain only those points that can potentially reach the admissible positions (lines 2 and 7).

The algorithm performs $|\vec{V}| - 1$ iterations (lines 4–11). First $\mathcal{R}_i$ and $\mathcal{P}_i$ are computed on the basis of the sets of end-anchors of the previous level $\mathcal{P}_{i-1}$ and the starting point of a selected fragment B, filtering out those that are not overlapping and those that lead to wrong portions of space (lines 7–8). The filtering based on $\sim$ is applied (line 10). During this step, the set of triples of 3D points $\mathcal{P}_i$ is clustered using $\sim$. A representative of each equivalence class is chosen (within $\mathcal{P}_i$) and the corresponding fragment in $\mathcal{R}_i$ is identified; all the other (non-identified) fragments are filtered out from $\mathcal{R}_i$. Let us also note that the

filtering based on clustering is not performed for the initial step $\mathcal{P}_1$, as typically this is already captured by the restrictions imposed by $\mathcal{A}$.

Once the fragments reachable at last iteration are determined and their representatives selected, we populate the Tab with the set of tuples associated to each representative L. The function $\eta(L)$ returns the assignments to $\vec{V}$ that allows us to overlap the last point to L.

The JMf algorithm is parametric w.r.t. the clustering relation and the function selecting the representative; they both express the degree of approximation of the rigid bodies to be built. The proposed clustering relation for loop modeling takes into account two factors: *(a)* The positions of the end-anchors in the 3D space and *(b)* The orientation of the plane formed by the fragment's anchor β_L w.r.t. a fixed reference system Γ_0 adopted by FIASCO (c.f. Figure 7). This combination of clusterings allows to capture local geometrical similarities, since both spatial and rotational features are taken into account.

The spatial clustering *(a)* used is the following. Given a set of fragments, three end points $C'ON$ (end anchors) of each cluster are considered, and the centroid of the triangle $C'ON$ is computed. We use three parameters: $k_{min}, k_{max} \in \mathbb{N}$, $k_{min} \leq k_{max}$, and $r \in \mathbb{R}$, $r \geq 0$. We start by selecting a set of k_{min} fragments, pairwise distant at least $2r$. These fragments are selected as representatives of an equivalence class for other fragments that fall within a sphere of radius r centered in the centroid of the representative. This clustering ensures a rather even initial distribution of clusters, however some fragments may not fall within the k_{min} clusters. We allow to create up to $k_{max} - k_{min}$ new clusters, each of them covering a sphere of radius r. Remaining fragments are then assigned to the closest cluster. The employed technique is a variant of the k-means clustering algorithm called *leader clustering algorithm*; it allows a fast implementation and acceptable results.

The orientation clustering *(b)* partitions the fragments according to their relative orientation of planes β_R w.r.t. Γ_0. A plane spatial orientation is described by the Euler angles ϕ, θ, ψ of its frame w.r.t. Γ_0. This algorithm produces a variable number of partitions depending on β. In particular, given a threshold $\beta > 0$ there are $3 \cdot (360/\beta)$ possible partitions describing equal regions on a sphere though the interval $(\phi \pm /\frac{\beta}{2}, \theta \pm /\frac{\beta}{2}, \psi \pm /\frac{\beta}{2})$. Each fragment is allotted to the partition determined by β.

The final cluster is the intersection of the two partitioning algorithms. This defines an equivalence relation $\sim$ depending on k_{min}, k_{max}, r, and β. The representative selection function selects the fragment for each partition according to some preferences (e.g., most frequent fragment, closest to the center, etc.).

Note that for $r = 0$, $\beta = 360$, and k_{max} unbounded, no clustering is performed and this would cause the combinatorial explosion of every possible end-anchor on the whole problem. The spatial error introduced depends on r and β. With $\beta = 360$, the error introduced at each step can be bounded by $2r$ for each dimension. At each iteration the errors are linearly increased, since a new fragment is placed with an initial error gathered from previous iterations, thus resulting in a $2nr$ bound for the last end-anchor. Clearly this bound is very coarse, and on average the experimental results show better performances. Similar considerations can be argued for rotational errors, however the intersection of the two clusterings, provide, in general, a much tighter bound.

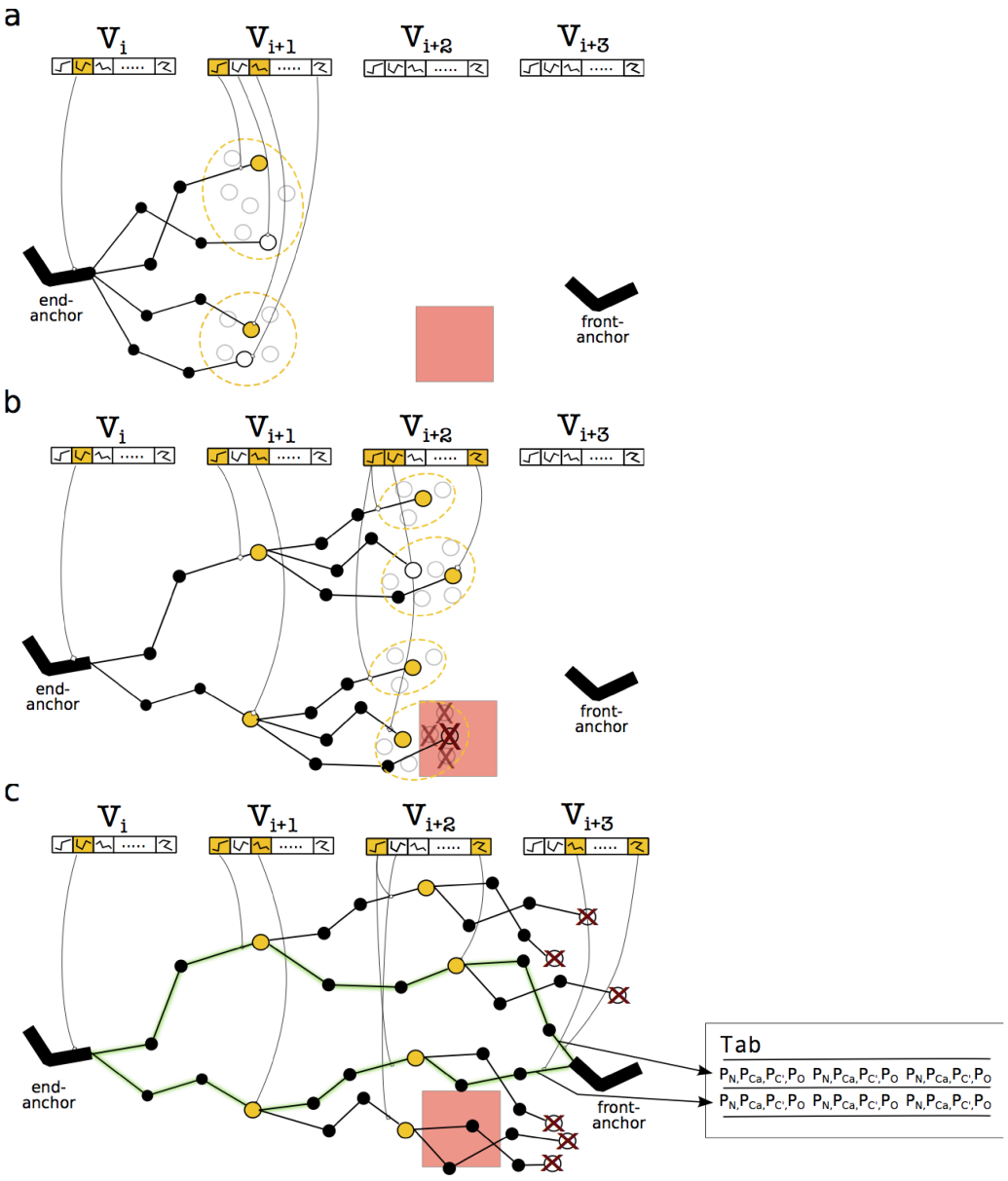

Figure 8: A graphical representation of the propagation of a JM constraint over the variables $V_i, \ldots, V_{i+3}$. **(a)** A simultaneous placement of all the elements in the domain of the variable V_{i+1} is simulated, by overlapping each corresponding fragment with the end-anchor of the fragment associated to the element in the domain of V_i. The set of points $\mathcal{P}_{i+1}$ is computed and clustered using the relation $\sim$ (points within the dotted ellipses). For each cluster one fragment representative is hence chosen (highlighted fragments with filled rightmost circle). The collection of representatives constitutes the set $\mathcal{R}_{i+1}$ **(b)** The previous step is performed again on the basis of the end-anchors related to the fragments representatives chosen in the previous level. The filled box, represents the set of points $\mathcal{G}$ that are not available for the placement of bodies (for instance due to a distance constraint). and the fragment falling in such area are discarded. **(c)** In the last iteration of the JMf algorithm the set of points $\mathcal{P}_{i+3}$ is not clustered, but only those that reach the desired position are retained, for instance the front-anchor associated to the fragment of the next variable, and the sequence of fragments able to lead to such condition (marked by thick lines) are selected to populate the table Tab.

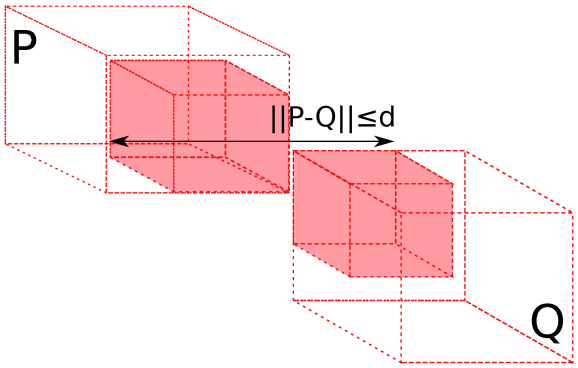

Figure 9: The effect of a distance constraint $||P - Q|| \leq d$ propagation. Empty boxes represent the original PVs domains and the full boxes represent the reduced PVs domains after the effect of constraint propagation.

4.2.2 Distance Constraints

The propagation of the distance constraints is an approximated technique that reduces the size of the box domains. We introduce the following operations over PVs box domains of two variables P and Q that will be used to describe the propagation rule in this and in the following subsections:

$$\begin{aligned} \textbf{Domain intersection:} \quad \mathsf{D}^P \cap \mathsf{D}^Q &= \langle \max(P_{\min}, Q_{\min}),\ \min(P_{\max}, Q_{\max}) \rangle \\ \textbf{Domain union:} \quad \mathsf{D}^P \cup \mathsf{D}^Q &= \langle \min(P_{\min}, Q_{\min}),\ \max(P_{\max}, Q_{\max}) \rangle \\ \textbf{Domain dilatation:} \quad \mathsf{D}^P + d &= \langle P_{\min} - d, P_{\min} + d \rangle \end{aligned}$$

where $\max(P, Q) = (\max(P_x, Q_x), \max(P_y, Q_y), \max(P_z, Q_z))$, (and similarly for min), and $P + d = (P_x + d, P_y + d, P_z + d)$.

Given two point variables P and Q, with domains D^P and D^Q, respectively, the simplification rule for the constraint $||P - Q|| \leq d$ updates the domains as follows:

$$D^P = ((\mathsf{D}^Q + d) \cap \mathsf{D}^P) \qquad D^Q = ((\mathsf{D}^P + d) \cap \mathsf{D}^Q) \tag{8}$$

which ensures that the points in D^P and D^Q are positioned within an approximation of a sphere of radius d. The sphere is approximated by considering the box inscribing it (a cube of side $2d$), as illustrated in Figure 9.

The propagation of the constraint $||P - Q|| \geq d$ is harder as the coarse representation of the box domains adopted in this work to model PVs does not allow the description of more complex polyhedron. We hence apply a simple form of bound consistency described by the following rule:

$$||P - Q|| \geq d : \frac{\left\{(\mathsf{D}^P \cup \mathsf{D}^Q) = \langle l, u \rangle,\ ||u - l|| < d\right\}}{\left\{\mathsf{D}^P = \emptyset, \mathsf{D}^Q = \emptyset\right\}} \tag{9}$$

that establishes unsatisfiability of the constraint.

4.2.3 Fragment Constraint

The propagation a fragment constraints $\texttt{fragment}(V, \vec{P}, T)$ is exploited during the solution search to enforce the assembly process of the fragment $T[V]$ along the point variables $P_1, \ldots, P_n$ of $\vec{P}$. Recall that D^V is the domain of V containing the references $\{j_1, \ldots, j_k\}$ to the database of fragments T.

$$\texttt{fragment}(V, \vec{P}, T) : \frac{\left\{\mathsf{D}^{P_1} = \{p_1\},\, \mathsf{D}^{P_2} = \{p_2\},\, \mathsf{D}^{P_3} = \{p_3\}, \mathsf{D}^V = \{j_1, \ldots, j_k\}\right\}}{\left\{\bigwedge_{i=1}^{n} \mathsf{D}^{P_i} = \mathsf{D}^{P_i} \cap \bigcup_{f=j_1}^{j_k} \{\rho((p_1, p_2, p_3), T[f]) \cdot T[f](i)\}\right\}} \tag{10}$$

where $\rho((p_1, p_2, p_3), T[f])$ is the roto-translation to be applied to overlap the first three points of the fragment $T[f]$ with the start-anchor (p_1, p_2, p_3).

The conjunction in the bottom part of the rule re-evaluates the domains for P_1, P_2, P_3, and it may reduce the singleton domains to empty whenever there is no compatible ρ for the selected fragment.

4.2.4 Centroid Constraint

When the positions of the atoms N, C_α and C' for an amino acids a are determined, the propagation algorithm enforces the value for the PV P_{CG} involved in the centroid constraint.

$$\texttt{centroid}(P_N, P_{C_\alpha}, P_{C'}, P_{\mathsf{CG}}, a) : \frac{\left\{\mathsf{D}^{P_N} = \{p_N\},\, \mathsf{D}^{C_\alpha} = \{p_{C_\alpha}\},\, \mathsf{D}^{P_{C'}} = \{p_{C'}\}\right\}}{\left\{\mathsf{D}^{P_{\mathsf{CG}}} = (\mathsf{D}^{P_{\mathsf{CG}}} \cap \{\texttt{cg}(p_N, p_{C_\alpha}, p_{C'}, a)\})\right\}} \tag{11}$$

where $\texttt{cg}(p_N, p_{C_\alpha}, p_{C'}, a)$ is a support function which returns the center of the mass for the side chain of the amino acid a by considering the points $p_N, p_{C_\alpha}, p_{C'}$, as described in Sect. 3.3.3.

4.2.5 Some Implementation Details

The proposed solver relies on an efficient C++ implementation, and it is carefully designed to allow additional tailored solving capability without the need of reshaping the core structures.

The internal representation of the domains of the finite domain variables can be abstracted by two arrays of the same length of the size of the initial domain. One array points to the values and the other is a Boolean bit-mask that states whether a value is still in the domain. If all flags are set to 0, the current partial assignment cannot be a part of a solution of the overall constraint; if exactly one is set to 1, then the variable is assigned to a value. This representation implies a linear scan of the domains during the propagation but it is justified by the reasonably small size of the domains of the target application (typically less than 100 values). The internal representation of the domains for point variables is simply a pair $\langle \min, \max \rangle$ that uniquely characterizes a 3D box in $\mathbb{R}^3$. Since these variables are used mostly in distance constraints, this representation is expressive enough (*Oct-trees* have been considered but with no significant advantage).

Point Variables propagation has been described above; these variables are instantiated after fragment selection.

For the management of the `uniqueseq` property (7) we implemented a dedicated data structure based on hash tables. Every time a PV is assigned, its value is mapped into a 3D voxel of fixed size. The 3D grid is implemented via a Hash Table with voxel indexes as keys and points contained in such voxels as values. All the operations can be performed in $\tilde{O}(1)$ (amortized complexity).

4.3 One Or More JM Constraints

We briefly describe how we have modeled two problems with FIASCO. The JM constraint is able to model geometrically assembly of fragments and therefore it is used for every protein model. A single JM that covers a protein ensures its flexibility, however for long proteins some computational and precision issues arise. It can be beneficial to model a protein by multiple JM constraints, e.g. $JM(i, j)$ and $JM(j, k)$ so that the amino acids from i to j are covered and the JM constraints overlap on a common amino acid. This practical choice improves the approximate search and allows to increase the number of different solutions produced. In practice, each protein section handled by a JM constraint is potentially combined to the different arrangements for the other sections. Therefore, it is expected that the number of solutions found grows exponentially in the number of JM constraints. The other JM constraint parameters can be used to control clustering precision and number of conformations found.

5. Experimental Results

We report on the experimental results obtained with the FIASCO system (available at `http://www.cs.nmsu.edu/fiasco`). Experiments are conducted on a Linux Intel Core i7 860, 2.5 GHz, memory 8 GB, machine. The solver has been implemented in C++.

The fragment database adopted is the FREAD database which has been shown to be effective in loop structure prediction (Choi & Deane, 2010). For the parameters analysis 5.1.4 we use a database of fragments of length 1. These fragments are classified by their amino acid class and their frequency of occurrence over the whole top-500.

We set the system to model the two applications described below. In particular, in Section 5.1 we analyze the loop modeling scenario and we focus on the performances of `JM` filtering by examining the filtering power and computational costs. Next, we compare the quality of the loop conformations generated, by measuring the RMSD of the proposed loop with respect to the native conformation. We then present some relationships among the JM parameters to control quality and efficiency.

In Section 5.2 we show some examples of ab-initio protein structure prediction and we conclude with a comparison of FIASCO against other constraint solvers, for protein models that can be described by a common subset of constraints.

5.1 Loop Modeling

The loop modeling problem is formalized by the presence of two known (large) fragments that are both fixed in the space. A sequence of amino acids of length n is given for connecting these two parts of the protein. A JM constraint is defined over the sequence, with particular attention to the starting and ending points that are fixed. The start of the first fragment

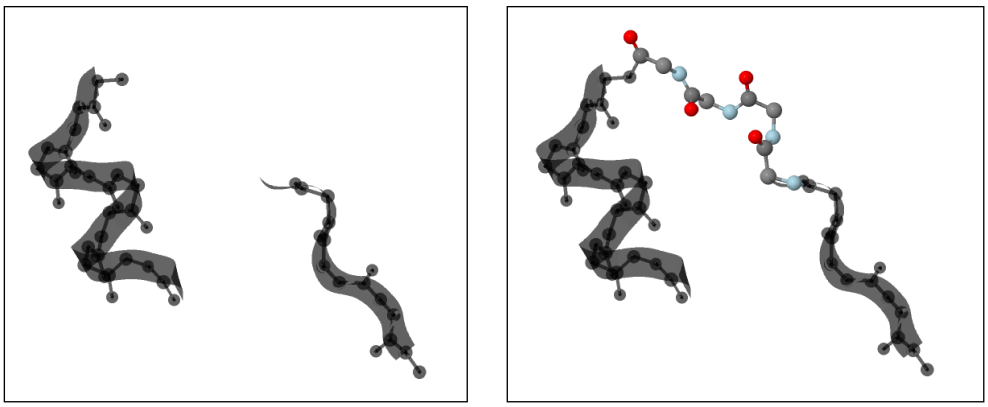

Figure 10: An example of loop computed by our tool

and the end of the last fragment, namely a sequence $C'ON$ (initial points) of coordinates $\vec{a} = (a^1, a^2, a^3)$, and a sequence $C'ON$ (final points) of coordinates $\vec{e} = (e^1, e^2, e^3)$ are known. There is one caveat about the end points: due to the discrete nature of fragment assembly, it is unlikely to exactly reach the final points. We accommodate for some errors, and require that the JM constraint produces results that fall within some threshold from the corresponding final points.

In Figure 10 we show an Example of loop computed by our tool (the parts of the protein to be connected are shown on the left and the connecting loop on the right).

Additional spatial constraints about points (e.g. no-good regions determined by presence of other atoms) are given. The constant δ (now $\delta = 1.5$Å) asserts a minimum distance between pairs of atoms.

5.1.1 Filtered Search Space And Performances

We selected 30 protein targets from a set of non-redundant X-ray crystallography structures as done by Canutescu and Dunbrack (2003). We partitioned the proteins into 3 classes according to their loop region lengths ($n = 4, 8$, and 12). We model a CSP that uses fragment assembly to model the loop, in particular using the JM constraint over the loop region.

To assess the filtering capabilities of FIASCO, we perform an exhaustive search generating all the solution for each of the protein targets. Using a clusterization of 0.2Å, a number of different fragments of length 1 is found for each amino acid (see Fig. 11). The size of the domains for the corresponding FDVs is bound by 100—this is an adequate sampling to describe a reasonable amino acid flexibility. In those cases where the number of fragments exceeds 100, the 100 most frequent ones are kept.

This increases the likelihood of generating a loop structure that is similar to the native one. A loop of length n generates an exponential search space of size bounded by 100^n. The selected variable is the leftmost one. Fragments are selected in decreasing frequency order. We have imposed a JM constraint for every 4 consecutive amino acids. The clustering parameters are set as follows: the k_{min} value is equal to the size of the domains, while we

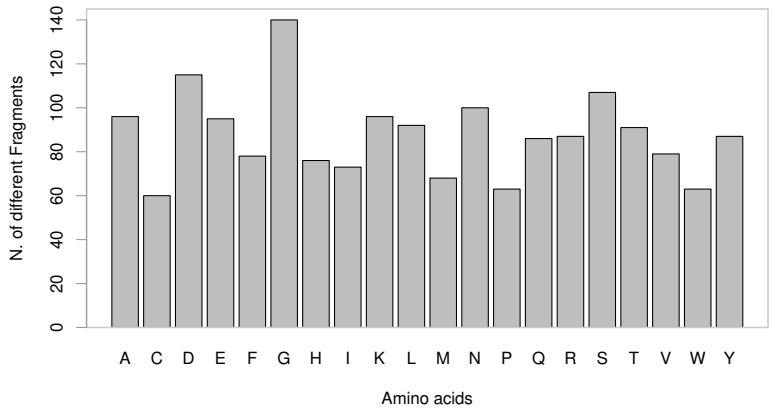

Figure 11: Number of different fragments (after clustering) per amino acid in the dataset

have used different values for k_{max} based on loop lengths. The values for r and β are set to 120 and 0.5 in each setting. A summary of the parameters is listed in Table 1.

In Table 1 we report the average times needed to exhaustively explore the loop search space, and the average number of solutions generated.

n	**JM Parameters**					**Full JM**	
	# JM	k_{min}	k_{max}	β	r	# Solutions	Time (s)
4	1	100	1000	120	0.5	597	3.13
8	2	100	500	120	0.5	98507	10.12
12	3	100	100	120	0.5	328309	28.87

Table 1: Loop Modeling settings and average running times (in seconds) and number of solutions generated.

5.1.2 JM Approximated Propagator Quality

Even if the approximated JM produces a small set of solutions, we show here that this is a good representation of the overall variability of the protein structure. For this test, we compare the solutions by means of RMSD from the original structures. The experiments were carried out with the same 30 protein targets and settings described in Table 1, with the only exception of k_{max} for the loop set of size 12, which was set to 500.

In Figure 12 we show the bar chart for the RMSD of the predictions for each protein loop within the group of targets analyzed. Precisely, in the x-axis there are the 30 (10 for each loop length) protein targets. Each bar reports the best RMSD (dark), the average RMSD (grey), and the worst RMSD (light grey) found. Numbers over the bars represent the number of loops found (multiplied by the factors indicated underneath). The results are biased by the fragment database in use: we excluded from it the fragments that belong

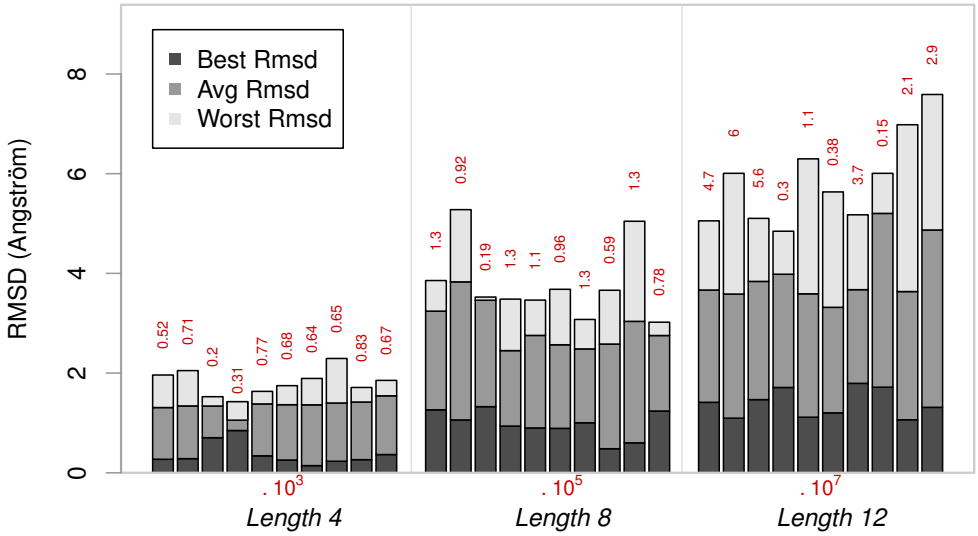

Figure 12: RMSD comparison for each Loop Set (x-axis: the 30 protein targets)

to the deposited protein targets. Therefore, it is not possible to reconstruct the original target loop and none of the searches are expected to reach a RMSD equal to 0.

For loops of length 8 and 12, the exploration of the whole conformational search space using a simple search procedure would result in an excessively long computation time. This enforces the need for a propagator such as JM, as its filtering algorithm successfully removes redundant conformations and it allows us to cover the whole search space in a short period of time.

In Fig. 12 loop predictions are calculated using fragments of length 1. To study how this choice affects both time and accuracy of the sampling we also model the loops of length 12 using fragment of length 3, 6, and 9. Best RMSDs are reported in Figure 13. For these experiments we kept the settings used above ($kmax = 500$). Moreover, each JM constraint is imposed on the fragments in order to cover the whole fragment (e.g, for fragments of length 3 we set a JM constraint every three consecutive amino acids) and we set a time-out of 3600 Seconds.

Notice that increasing the length of the fragments the accuracy decreases due to the reduced size of the domains. Nevertheless, the time is also reduced since the sampling is performed on a smaller search space and the JM constraints cover longer sequences of amino acids. The average times are: 1580.14, 0.98, and 0.74 seconds using fragments of length 3, 6, and 9 respectively.

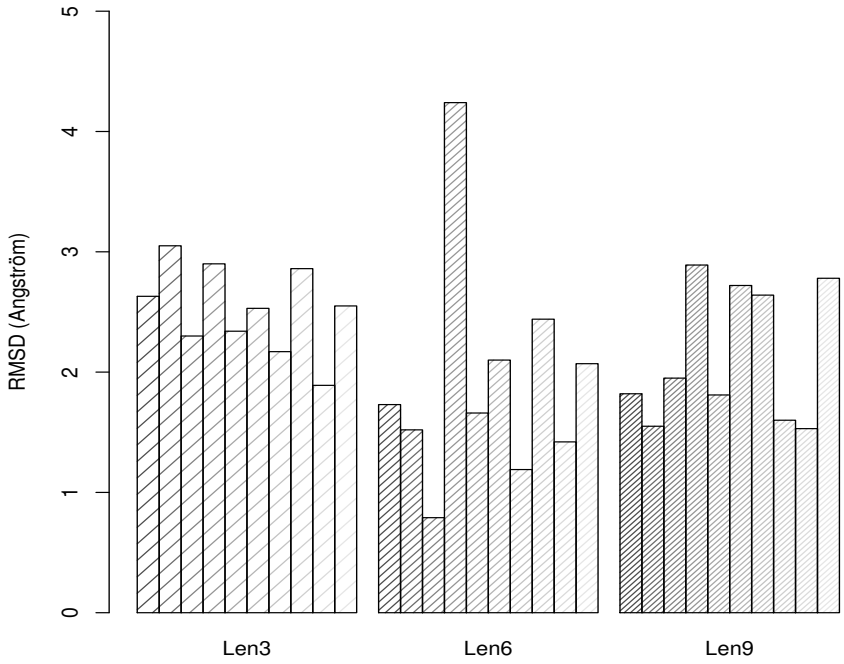

Figure 13: RMSD comparison for loop sampling on loops of length 12 using fragments of length 3, 6, and 9.

5.1.3 Comparison With State-of-the-art Loop Samplers

In this section, we compare our method to three state-of-the-art loop samplers: the *Cyclic Coordinate Descent (CCD)* algorithm (Canutescu & Dunbrack, 2003), the *Self-Organizing algorithm (SOS)* (Liu, Zhu, Rassokhin, & Agrafiotis, 2009), and the *FALCm* method (Lee, Lee, Park, Coutsias, & Seok, 2010).

Table 2 shows the average of the best RMSD for the benchmarks of length 4, 8 and 12 as computed by the four programs. We report the results as given in Table 2 of Canutescu and Dunbrack for the CCD, Table 1 of Liu et al. for SOS, Table II of Lee et al. for the FALCm method, and the RMSD's obtained adopting the settings for JMf that provided the best results in the previous section (see also Subsection 5.1.5). It can be noted that our results are in line with those produced by the other systems.

Loop	**Average (best) RMSD**			
Length	CCD	SOS	FALCm	JMf
4	0.56	**0.20**	0.22	0.27
8	1.59	1.19	**0.72**	0.93
12	3.05	2.25	1.81	**1.58**

Table 2: Comparison of loop sampling methods

The execution time we reported appear to be very competitive (e.g., if we considered the results reported in Soto et al., 2008).

5.1.4 JM Parameters Analysis

In this section, we analyze the impact of the `JM` parameters on the quality of the best solutions found and on the execution times. In particular, the aim of these experiments is to shed light on the relationship between the `JM` constraint settings and the results.

In Figure 14, we analyze the impact of the k_{max} on the execution times (left) and on the precision (right) of the filtering of the JM constraint. From top to bottom, we use $\beta = 60, 120, 360$. The tests are performed over the protein loops of length 4 (see section above), adopting as cluster parameters, r in $\{0.5, 1.0, 3.0, 5.0\}$, and $k_{min} = 100$. Each dot in the plots represents the average of the best RMSD found by each predictions (left) and the average execution time (right). The RMSD values tend to decrease for smaller clustering parameters r and β and as the number of clusters increases, while the filtering time increases as k_{max} increases.

In Figure 15 we study the relation between the RMSD and both the number of JMs that cover a given target loop or protein and the Voxel-side parameter. For these experiments we used the values $\{100, 250, 500, 800, 1000\}$ for the k_{max}, we set $r = 1$, $\beta = 120$, and we averaged the RMSDs values on the resulting sample set of structures. The relation between the RMSD and number of JM as well as the average and worst computational times are shown in Fig. 15 left. Here we use a medium-length loop taken from the protein 1XPC (res. 216-230) and we vary the number of JMs that cover the loop (the side of the voxel has been set to $3\mathring{A}$). From the figure we observe that increasing the number of JMs (i.e. covering less amino acids with a single JM) the RMSD decreases but the computational cost is higher. Notice that the best RMSD is given when the loop is covered by 4 JM

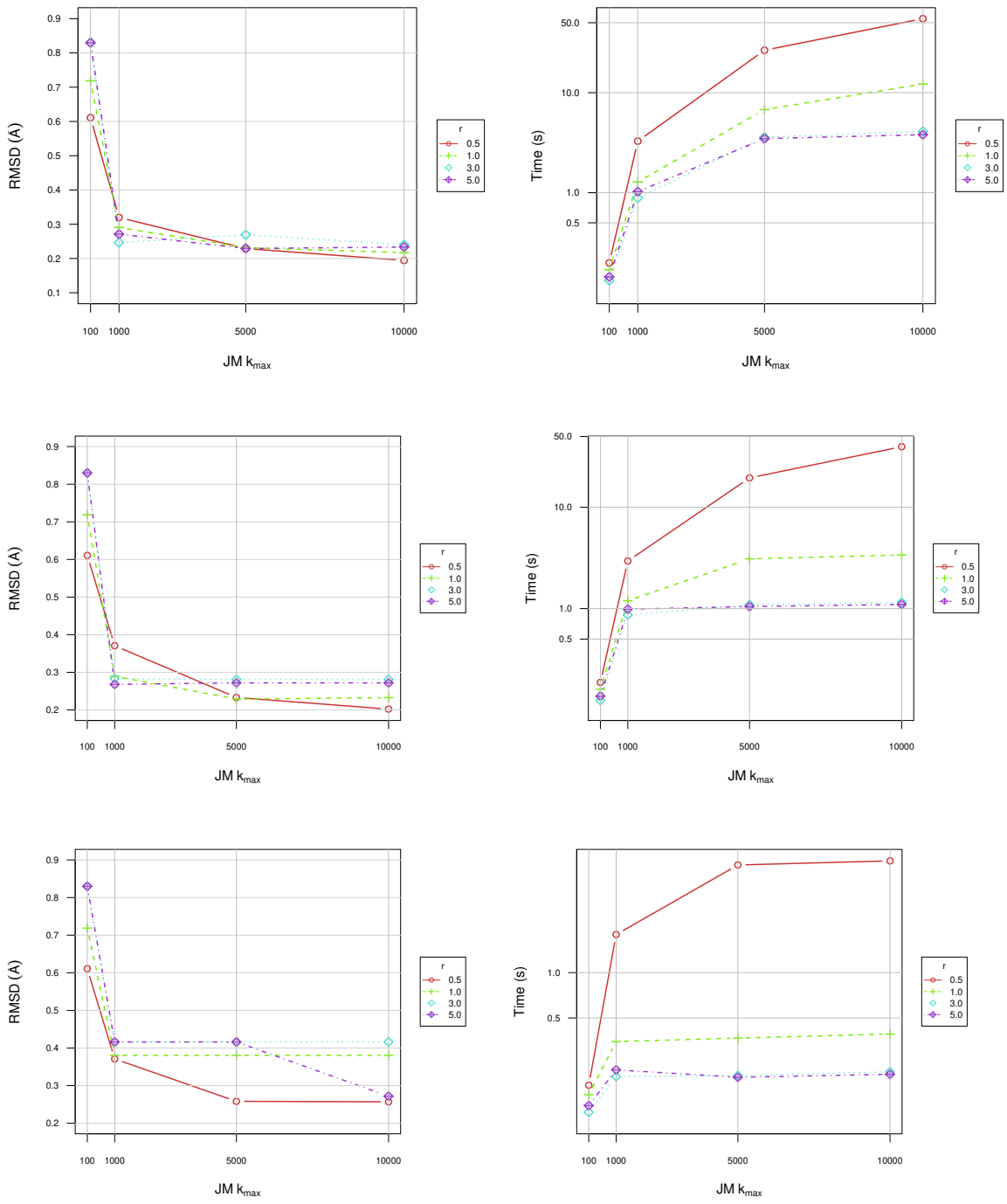

Figure 14: Comparison of the best RMSD values and execution times at varying of the k_{max} clustering parameter for $\beta = 60\,(top), 120\,(center), 360\,(bottom)$

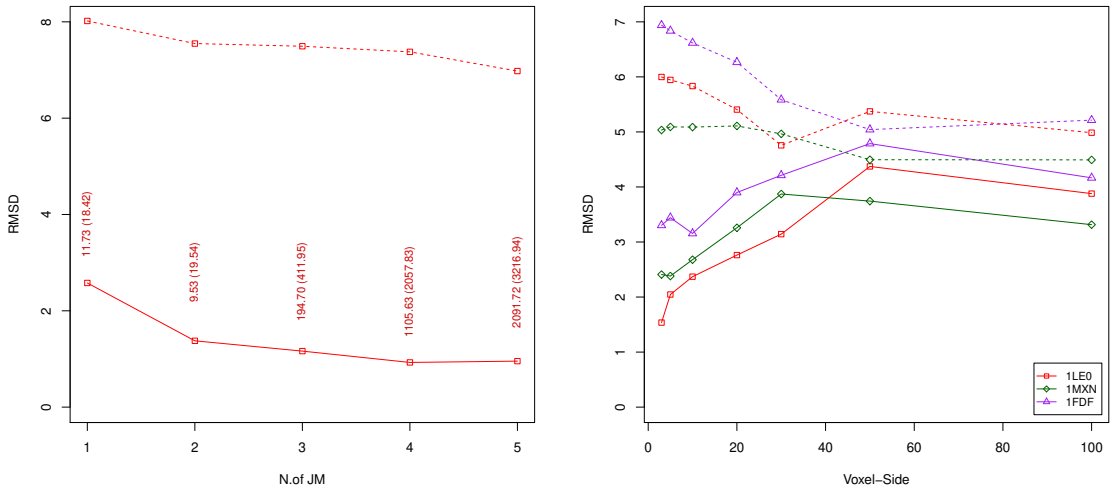

Figure 15: Left: RMSD (best and average) and Time (average and worst) values increasing the number of JM constraints that completely cover a target loop of length 15. Right: Average (dotted line) and best (solid line) RMSD for the targets 1LE0 of length 12 (top), 1MXN of length 16 (medium), and 1FDF of length 25 (low). The JM-*Voxel-side* parameter for the voxels of the clustering varies from 3 to 100. The JM k_{max} parameter varies from 100 to 1000. The targets are completely covered by multiple JM-constraints.

constraint (i.e., a JM constraint each four consecutive amino acids). As a rule of thumb we suggest to use a JM constraint to cover from 3 to 4 consecutive amino acids since this setting produces the best results within an acceptable time. In Fig. 15 right we report the best RMSD (solid line) and the average RMSD (dotted line) of the structures found using multiple `JM` constraints that cover sequences of 4 consecutive amino acids through the whole target proteins. Namely, if the protein target has length n, we set the `JM` constraints from i to $i+3$, where $i = 3 \cdot j, 0 \leq j < n/3$. For these experiments, we considered three proteins of relatively short length, in order to obtain a complete exploration of the search space in reasonable computational time: 1LE0 (length 12), 1MXN (length 16), and 1FDF (length 24). Moreover we used the values $\{3, 5, 10, 20, 30, 50, 100\}$ for the side of the voxels used for the clustering.

From the Figure 15 we observe that the voxel size (enabled by the `uniqueseq`) has an impact on the clustering for values lower than $30\mathring{A}$ (recall that these proteins have a diameter less than $30\mathring{A}$). For voxel sides lower than $3\mathring{A}$ we observe no substantial improvement in terms of quality, while the time required by the solver to compute the solutions increases exponentially.

5.1.5 Results Summary And Default Parameters

We now provide some guidelines that may be helpful to tune the JM parameters for a given protein modeling problem. We suggest several levels of parametrization that might be used according to the user needs with respect to running time or prediction accuracy. We stress

that these are merely guidelines, outlined from our empirical evaluations, and that several tests should be done to establish the desired tuning.

We suggest to set a JM to model a sequence of at least 3 amino acids and in general not longer than 8, to payoff the computational load of the JM clustering. The default choice for k_{min} is set to be the average size of the variable domains involved in a JM constraint, while we suggest to set k_{max} to be at least as k_{min} and not greater than 10000. The latter, together with the number of consecutive JM constraints, will have the greatest impact on the computational cost and prediction accuracy. Computational costs will grow as the number of consecutive JM increases, and at the same time it will also produce in general higher accuracy. The same trend is exhibited by the growing k_{max} parameter. Table 3 illustrates five basic settings that could be used incrementally to establish a trade off between running times and prediction accuracy. The first level (Lev. 1) is associated to faster computational times and lower accuracy while the last one (Lev. 5) is the slowest but also the most accurate. The second column of the table indicates the length of the amino acid sequence modeled by a single JM.

Lev.	$n.JM$	k_{min}	k_{max}	β	r	Speed	Accuracy
1	8	$\|D\|$	500	120	5	* * * *	*
2	8	$\|D\|$	1000	120	3	* * *	* *
3	6	$\|D\|$	100	120	3	* * *	* *
4	4	$\|D\|$	500	120	3	* *	* * *
5	4	$\|D\|$	1000	120	1	*	* * * *

Table 3: JM default parameters

5.2 An Application In Protein Structure Prediction

In the protein structure prediction problem, we model a generic backbone through multiple JM constraints. In principle, an unique JM constraint can model the whole problem. As in the previous cases, we split it into smaller parts, moreover, the presence of secondary structure is a valid help in the placement of JM constraints that can handle loops between each consecutive pair. A simple search can generate a pool of conformations, then energy scoring can select the best candidate. We have used a statistical energy function developed for the 5@ model, but any other energy function can be used instead.

In this section, we study the applicability of FIASCO to the protein structure prediction problem. In particular, we consider prediction problems where the secondary structure elements of the protein are given. Furthermore, in order to assess the potential structure, we introduce an energy function—the same that we have adopted in previous studies, and more precisely described in `http://www.cs.nmsu.edu/fiasco`.

For the modeling, we have used the information about the location and the type of the secondary structure elements on the primary sequence provided directly by the Protein Data Bank. We have imposed a sequence of JM constraints between every consecutive pair of secondary structure elements. The number of consecutive JM constraints varied according to the length of the unstructured sequence being modeled, covering at most 5 amino acids with a single JM constraint. In addition one JM constraint was imposed from the first amino acid to the beginning of the first secondary structure element and another

from the end of the last secondary structure element and the last amino acid (the tails of the protein). The domains for the initial and end points of the JM constraints are the set of all admissible points (a box large enough to contain the protein). In the search phase, the "first" secondary structure is deterministically set in the space. Then the labeling proceeds with the JM constraint attached to it leading to the next secondary structure and so on. Tails are instantiated at the end.

The propagation of the constraints generates a set of admissible structures, that represents the possible folds of the target protein. From this set, we select the structure with minimum energy; we extract also the structure with minimum RMSD, in order to evaluate the quality of the energy function. For these tests we adopt the FREAD database. Table 4 reports the best energy values found by FIASCO. In the RMSD columns is reported the corresponding RMSD associated to the conformation with best energy found by the solver. The $\#JM$ column reports the total number of JM used to model each protein, together with the maximum number of consecutive JM adopted to model a contiguous sequence of amino acids (within parentheses).

Protein ID	Len.	# JM	Energy	RMSD	Time (Min.)
1ZDD	35	4(2)	−100513	2.05	11.42
2GP8	40	4(2)	−138110	6.28	8.55
2K9D	44	5(2)	−204693	2.52	2.69
1ENH	54	4(1)	−309896	8.21	31.67
2IGD	60	7(2)	−295882	10.50	26.47
1SN1	63	7(3)	−358874	5.55	14.82
1AIL	69	4(1)	−411077	4.59	4.46
1B4R	79	11(2)	−313590	6.11	8.41
1JHG	100	7(1)	−572950	4.51	4.50

Table 4: Ab initio prediction with FIASCO.

The results show that the quality of the predictions computed by FIASCO (6.3 as average RMSD) is competitive (and, as shown in the following section, at par or better than what produced by other methods). The results are particularly encouraging for proteins of longer length, where the sampling of the search space aids in development of admissible structures. The time required by FIASCO to completely explore the search space depends strongly on the type and the mutual arrangement of secondary structure elements of the target. For example, the protein *2K9D* and the protein *1ENH* have the same length, but FIASCO is significantly faster on the first protein than on the second one. The same observation can be made for the proteins *2IGD* and *1SN1*. The results reported in Table 4 are promising and they suggest that this is a feasible approach to solve the ab initio prediction problem. As a future work, we will explore the integration of local search techniques (e.g., large-neighboring search), in order to sample the search space and to further decrease the time needed to explore it.

5.3 A Comparison of FIASCO with State-of-the-Art Constraint Solvers

In this section, we motivate our choice of designing an ad-hoc solver instead of using a general-purpose constraint solver. In particular we provide a comparison between FIASCO and state-of-the-art constraint solving. The results justify the choice of implementing a new solver from scratch instead of using an available constraint programming library or a constraint programming language. The solver chosen for this comparison is Gecode (Gecode Team, 2013), a very efficient solver and the winner of the most recent MiniZinc challenges (Stuckey, Becket, & Fischer, 2010).

Gecode has recently introduced (in version 4.0) the handling of floating point variables. Nevertheless, since Gecode is the fastest solver for FD variables, we have first encoded the PSP by discretizing fragments and positions. In particular, we multiplied each real number by a scaling factor (100) to obtain integer values. Each spatial position is encoded by a triple of variables, representing the coordinates of the point. Each operation (e.g., multiplications) applied to such variables requires re-scaling of the result; this unfortunately leads to ineffective propagation. This is particularly evident when dealing with distance constraints, that require the implementation of Euclidean distance between pairs of triples of variables.

In order to understand the solvers capabilities to propagate constraints on the placement of overlapping fragment we implemented three versions of the code, that considered a different number of constraints, precisely:

1. An implementation that uses only JM constraint (JM only)
2. An implementation that adds the `alldistant` constraint and
3. An implementation that adds the `alldistant` and `centroid` constraints

In all cases we use a complete search (in particular, the clustering and tabling constraints of lines 10 and 12–14 of Algorithm 2 are disabled).

In Table 5, we report the execution times required by FIASCO and by Gecode (with the same considered constraints) to determine an increasing number of solutions, from $1,000$ to $1,000,000$. These solutions are computed for the target protein 1ZDD which has length 35. Table 5 shows that the execution time of both solvers increases proportionally with the number of solutions found. However, FIASCO is one order of magnitude faster than Gecode in the unconstrained case, and two orders of magnitude faster in the other cases. The main reason is that FIASCO is specifically developed to handle the finite domains and 3D point variables, while these are approximated by FD variables in Gecode. Constraints on these approximations propagate poorly and slowly. Moreover, the approximation of fragments using finite domain variables introduces approximation errors, that grow during the search phase (and consequently, less solutions are returned in the constrained cases). These errors may result in final structures that are relatively imprecise when the coordinates of the atoms are converted back into real values.

In Table 6, we consider a small sequence of four amino acids (SER TRP THR TRP—the first four amino acids of the protein 1LE0), and we generate all solutions. We report the values of the best and the average RMSD among the structures of the sets of solutions computed using FIASCO and the Gecode implementation after a complete enumeration of the

Number of solutions	FIASCO			Gecode		
	JM only	`alldistant`	`alldistant + centroid`	JM only	`alldistant`	`alldistant + centroid`
1000	**0.030**	**0.051**	**0.059**	0.358	2.531	3.807
10000	**0.312**	**0.476**	**0.612**	2.571	21.056	35.370
100000	**3.006**	**4.794**	**6.040**	25.407	209.569	347.831
1000000	**29.859**	**47.669**	**61.385**	252.815	2186.83	3632.39

Table 5: Comparison of the execution times of FIASCO and Gecode, for increasing number of solutions and with different sets of considered constraints.

domains. We can observe that FIASCO is significantly faster in exploring the search space, moreover, the approximation introduces errors that leads to the loss of feasible solutions.

	FIASCO				Gecode			
	N. sol.	Time (sec.)	RMSD	Avg. RMSD	N. sol	Time (sec.)	RMSD	Avg. RMSD
JM only	**810000**	**20.493**	**0.167**	**1.570**	**810000**	181.102	0.190	1.596
`alldistant`	**805322**	**33.493**	**0.167**	**1.564**	774463	252.974	0.190	1.591
`alldistant + centroid`	**805322**	**38.953**	**0.167**	**1.564**	169441	140.644	0.580	1.880

Table 6: Number of solutions, time, best RMSD, and average RMSD on the set of structures found by FIASCO and Gecode after a complete enumeration of the solution space using different constraints

We have encoded the same constraint satisfaction problem using the new version of Gecode that allows to employ float variables. We labeled the finite domain variables that allow to select fragments, while values for the point variables are obtained by constraint propagation. Since constraint propagation of float variables is based on interval arithmetics, it turns out that after few amino acids these intervals are too large for being able of reconstructing the protein and or evaluating the energy value. For instance, after a complete assignment of the variables related to fragments of protein 1ZDD, while the domains of the float variables associated with the position of the first two amino acids are singletons, those related to the tenth amino acids are intervals with size from one to two Å; even worse, the domains of the atoms of the eleventh amino acids are unbounded. A further stage of labeling of the float variables required computational time of orders of magnitude higher than those reported in Table 6 for the finite domain Gecode implementation.

Constraint solvers like ECLiPSe (Cheadle, Harvey, Sadler, Schimpf, Shen, & Wallace, 2003) and Choco (Choco Team, 2008) also support the mixed use of integer and real variables. ECLiPSe is a Prolog-based language which handles integer and real variables together. However, the great number of matrix operations required in our application does not fit well with a Prolog implementation. Furthermore, the current trend of ECLiPSe is to replace a direct constraint solving with a translation to FlatZinc. In the case of Choco, the current support of Real Variables is still under development (c.f. `http://choco.sourceforge.net/userguide.pdf`—page 3). Things may change with the next releases.

We also experimented with another constraint solver, by implementing the multi-body constraints using the JaCoP library (JaCoP Team, 2012), in a similar way as done for Gecode. Eventually, we tested the same protein used for the results reported in Table 5,

and we did not observe any substantial difference in terms of execution time, from the Gecode implementation.

In terms of protein structure prediction, the design of FIASCO has been influenced by our own previous work on the TUPLES system (Dal Palù et al., 2011). TUPLES is also a constraint solver for protein structure prediction, based on fragments assembly. Figure 16 compares the performance of TUPLES and FIASCO on the same set of proteins discussed in Section 5.2. To make the comparison fair, we make use of the same energy function in both systems and assume that the secondary structure elements are known. Note that there are some important differences between the two systems. TUPLES is implemented using constraint logic programming techniques, specifically, SICStus Prolog (Swedish Institute for Computer Science, 2012); TUPLES does not make use of floating point variables; on the other hand, TUPLES introduces a heuristic search mechanism based on large neighboring search.

The results show that the quality of the predictions computed by FIASCO (6.3 as average RMSD) is better than the quality of the predictions computed by TUPLES (9.4 as average RMSD). The complete sampling of the search space allows us to obtain better results for the proteins of longer length in the benchmark (≥ 63). Instead, for shorter proteins, we obtain comparable results. The similarity of the quality depends on the use of the same energy function for both the systems. Notice that the energy function used is designed for the simpler model adopted in TUPLES (C_α–C_α). Moreover, TUPLES is based on a Prolog implementation that does not provide floating point variables and hence each value must be rounded and approximated. These aspects explain both the quality differences between the RMSD and the Best RMSD found by FIASCO and the behavior for which for some proteins (e.g., *1ZDD*, *2GP8*) the (energy) RMSD values are better in FIASCO even if their corresponding energy (RMSD) values are higher than in TUPLES. The execution times of FIASCO are significantly faster than TUPLES, in spite of FIASCO's lack of a sophisticated search heuristic.

We also performed a comparison with the state-of-the-art online Robetta predictor (Raman, Vernon, Thompson, Tyka, Sadreyev, Pei, Kim, Kellogg, DiMaio, Lange, Kinch, Sheffler, Kim, Das, Grishin, & Baker, 2009) for the first four proteins of Table 6. We built the dictionary for 3 and 9 amino acid long peptides through the Robetta interface, and we disabled any homology inference, in order to maintain a fair comparison. The results are: 1ZDD computed in 21s with 5.92 RMSD, 2GP8 computed in 16s with 5.44 RMSD, 2K9D computed in 22s with 4.65 RMSD, 1ENH computed in 39s with 2.74 RMSD. It can be noted that our results are in line with Robetta predictor.

Let us conclude this section mentioning that the results reported in the previous section (where we compared FIASCO with TUPLES) provide also an implicit comparison with another off-the-shelf solver, the SICStus Prolog constraint logic programming solver (Swedish Institute for Computer Science, 2012).

6. Conclusions

In this paper, we presented a novel constraint (joined-multibody) to model rigid bodies connected by joints, with constrained degrees of freedom in the 3D space. We presented a polynomial time approximated filtering algorithm of the joined-multibody constraint, that

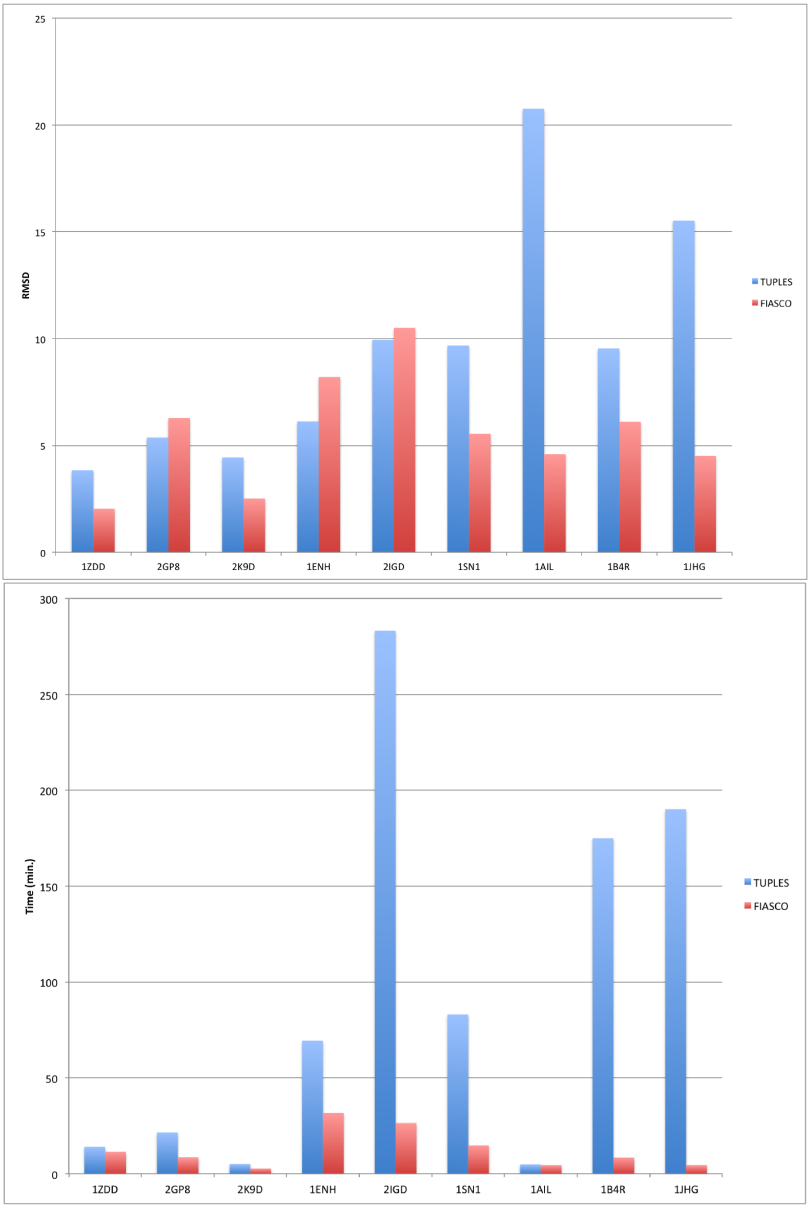

Figure 16: Comparison of RMSD and Execution Time between TUPLES and FIASCO

exploits the geometrical features of the rigid bodies. In particular, the filtering algorithm is combined with search heuristics that can produce a pool of admissible solutions that are uniformly sampled. This allows for a direct control of the quality and number of solutions. The filtering algorithm is based on a 3D clustering procedure that is able to cope with a high variability of rigid bodies, while preserving the computational cost. The practical advantages of the joined-multibody constraint are shown by an extensive set of real protein simulations for two main categories: protein loop reconstruction and structure prediction (ab-initio). The tests showed how the parameters of the constraint are able to control effectively the quality and computational cost of the search. In conclusion, the constraint solver FIASCO is able to model effectively various common protein case-studies analyses. As future work, from the applications side, we plan to explore the protein loop closure problem, with the use of specific databases and scoring functions. For the close problem of protein flexibility, we plan to use FIASCO solver to generate the conformational space of long scale movements for nuclear receptors (Dal Palù et al., 2012b). Finally, we plan to use FIASCO in the general context of protein structure prediction with the combination of local search methods and protein-ligand spatial constraints. From the constraint side, we plan to integrate the JM filtering algorithm with other distance constraints, in order to generate more accurate clusters; we plan to integrate spatial constraints inferred from bounds on energy terms (e.g., the favorable contributions provided by pairing secondary structure elements translate into energy bounds and distance constraints). We plan to investigate the use of multiple JM constraints to model super-secondary structures placement, which are useful to capture important functional and structural protein features. The latter can be thought of as imposing several spatial path preferences to a given chain of points. Finally, we intend to integrate the constraint solver with a visual interface to make it easily available to Biologist and other practitioners and porting some parts of this tool within a GPU-based framework as recently explored by Campeotto, Dovier, and Pontelli (2013).

Acknowledgments

We thank Federico Fogolari for his comments on several parts of this work. The authors would like to express gratitude to JAIR reviewers that helped us to sensibly improve the presentation.

References

Al-Bluwi, I., Simeon, T., & Cortes, J. (2012). Motion Planning Algorithms for Molecular Simulations: A Survey. *Computer Science Review*, *6*(4), 125–143.

Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., & Walter, P. (2007). *Molecular Biology of the Cell* (5th Edition edition). Garland Science.

Anfinsen, C. B. (1973). Principles that Govern the Folding of Protein Chains. *Science*, *181*, 223–230.

Apt, K. (2009). *Principles of Constraint Programming*. Cambridge University Press.

Backofen, R., & Will, S. (2006). A Constraint-Based Approach to Fast and Exact Structure Prediction in 3-Dimensional Protein Models. *Constraints*, *11*(1), 5–30.

Backofen, R., Will, S., & Bornberg-Bauer, E. (1999). Application of Constraint Programming Techniques for Structure Prediction of Lattice Proteins with Extended Alphabet. *Bioinformatics*, *15(3)*, 234–242.

Baker, D., & Sali, A. (2001). Protein Structure Prediction and Structual Genomics. *Science*, *294*(5540), 93–96.

Barahona, P., & Krippahl, L. (2008). Constraint Programming in Structural Bioinformatics. *Constraints*, *13*(1-2), 3–20.

Ben-David, M., Noivirt-Brik, O., Paz, A., Prilusky, J., Sussman, J. L., & Levy, Y. (2009). Assessment of CASP8 Structure Predictions for Template Free Targets. *Proteins*, *77*, 50–65.

Bennett, W., & Huber, R. (1984). Structural and Functional Aspects of Domain Motions in Proteins. *Crit. Rev. Biochem.*, *15*, 291–384.

Borning, A. (1981). The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory. *ACM Transactions on Programming Languages and Systems*, *3*(4), 353–387.

Cahill, S., Cahill, M., & Cahill, K. (2003). On the Kinematics of Protein Folding. *Journal of Computational Chemistry*, *24*(11), 1364–1370.

Campeotto, F., Dovier, A., & Pontelli, E. (2013). Protein Structure Prediction on GPU: a Declarative Approach in a Multi-agent Framework. In *International Conference on Parallel Processing (ICPP)*, pp. 474–479. IEEE Computer Society Press.

Campeotto, F., Dal Palù, A., Dovier, A., Fioretto, F., & Pontelli, E. (2012). A Filtering Technique for Fragment Assembly-Based Proteins Loop Modeling with Constraints. In Milano, M. (Ed.), *CP*, Vol. 7514 of *Lecture Notes in Computer Science*, pp. 850–866. Springer.

Canutescu, A., & Dunbrack, R. (2003). Cyclic coordinate descent: a robotics algorithm for protein loop closure. *Protein Sci*, *12*, 963–972.

Cheadle, A. M., Harvey, W., Sadler, A. J., Schimpf, J., Shen, K., & Wallace, M. G. (2003). ECLiPSe: An Introduction. Technical report IC-Parc 03–1, IC-Parc, Imperial College London.

Chelvanayagam, G., Knecht, L., Jenny, T., Benner, S., & Gonnet, G. (1998). A Combinatorial Distance-Constraint Approach to Predicting Protein Tertiary Models from Known Secondary Structure. *Folding and Design*, *3*, 149–160.

Choco Team (2008). Choco: an Open Source Java Constraint Programming Library. In *Workshop on Open-Source Software for Integer and Constraint Programming*. Available from `http://www.emn.fr/z-info/choco-solver/`.

Choi, Y., & Deane, C. M. (2010). FREAD Revisited: Accurate Loop Structure Prediction Using a Database Search Algorithm. *Proteins*, *78*(6), 1431–40.

Clementi, C. (2008). Coarse-grained Models of Protein Folding: Toy Models or Predictive Tools?. *Curr Opin Struct Biol*, *18*, 10–15.

Corblin, F., Trilling, L., & Fanchon, E. (2005). Constraint Logic Programming for Modeling a Biological System Described by a Logical Network. In *Workshop on Constraint-Based Methods for Bioinformatics*.

Cortes, J., & Al-Bluwi, I. (2012). A Robotics Apporach to Enhance Conformational Sampling of Proteins. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Vol. 4, pp. 1177–1186. ASME.

Crescenzi, P., Goldman, D., Papadimitriou, C., Piccolboni, A., & Yannakakis, M. (1998). On the Complexity of Protein Folding. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pp. 597–603. ACM Press.

Dal Palù, A., Dovier, A., Fogolari, F., & Pontelli, E. (2012a). Protein Structure Analysis with Constraint Programming. In Cozzini, P., & Kellogg, G. (Eds.), *Computational Approaches to Nuclear Receptors*, chap. 3, pp. 40–59. The Royal Society of Chemistry.

Dal Palù, A., Spyrakis, F., & Cozzini, P. (2012b). A New Approach for Investigating Protein Flexibility Based on Constraint Logic Programming: The First Application in the Case of the Estrogen Receptor. *European Journal of Medicinal Chemistry*, *49*, 127–140.

Dal Palù, A., Dovier, A., & Fogolari, F. (2004). Constraint Logic Programming Approach to Protein Structure Prediction. *BMC Bioinformatics*, *5*, 186.

Dal Palù, A., Dovier, A., Fogolari, F., & Pontelli, E. (2010). CLP-based protein fragment assembly. *Theory and Practice of Logic Programming*, *10*(4-6), 709–724.

Dal Palù, A., Dovier, A., Fogolari, F., & Pontelli, E. (2011). Exploring Protein Fragment Assembly Using CLP. In Walsh, T. (Ed.), *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*, pp. 2590–2595. IJCAI/AAAI.

Dal Palù, A., Dovier, A., & Pontelli, E. (2005). A New Constraint Solver for 3D Lattices and Its Application to the Protein Folding Problem. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pp. 48–63. Springer Verlag.

Dal Palù, A., Dovier, A., & Pontelli, E. (2007). A Constraint Solver for Discrete Lattices, its Parallelization, and Application to Protein Structure Prediction. *Software Practice and Experience*, *37*(13), 1405–1449.

Dal Palù, A., Dovier, A., & Pontelli, E. (2010). Computing Approximate Solutions of the Protein Structure Determination Problem using Global Constraints on Discrete Crystal Lattices. *International Journal of Data Mining and Bioinformatics*, *4*(1), 1–20.

Deane, C., & Blundell, T. (2001). CODA. A Combined Algorithm for Predicting the Structurally Variable Regions of Protein Models. *Protein Sci*, *10*, 599–612.

Debartolo, J., Hocky, G., Wilde, M., Xu, J., Freed, K., & Sosnick, T. (2010). Protein Structure Prediction Enhanced with Evolutionary Diversity: SPEED. *Protein Science*, *19*(3), 520–534.

Dotú, I., Cebrián, M., Van Hentenryck, P., & Clote, P. (2011). On Lattice Protein Structure Prediction Revisited. *IEEE/ACM Trans. Comput. Biology Bioinform*, *8*(6), 1620–1632.

Dunbrack, R. (2002). Rotamer Libraries in the 21st Century. *Curr. Opin. Struct. Biol.*, *12*(4), 431–440.

Erdem, E. (2011). Applications of Answer Set Programming in Phylogenetic Systematics. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, pp. 415–431. Springer Verlag.

Erdem, E., & Ture, F. (2008). Efficient Haplotype Inference with Answer Set Programming. In *National Conference on Artificial Intelligence (AAAI)*, pp. 436–441. AAAI/MIT Press.

Felts, A., Gallicchio, E., Chekmarev, D., Paris, K., Friesner, R., & Levy, R. (2008). Prediction of Protein Loop Conformations using AGBNP Implicit Solvent Model and Torsion Angle Sampling. *J Chem Theory Comput*, *4*, 855–868.

Fiser, A., Do, R., & Sali, A. (2000). Modeling of Loops in Protein Structures. *Protein Sci*, *9*, 1753–1773.

Fogolari, F., Esposito, G., Viglino, P., & Cattarinussi, S. (1996). Modeling of Polypeptide Chains as C_α Chains, C_α Chains with C_β, and C_α Chains with Ellipsoidal Lateral Chains. *Biophysical Journal*, *70*, 1183–1197.

Fogolari, F., Pieri, L., Dovier, A., Bortolussi, L., Giugliarelli, G., Corazza, A., Esposito, G., & Viglino, P. (2007). Scoring Predictive Models using a Reduced Representation of Proteins: Model and Energy Definition. *BMC Structural Biology*, *7*(15), 1–17.

Fogolari, F., Corazza, A., Viglino, P., & Esposito, G. (2012). Fast Structure Similarity Searches among Protein Models: Efficient Clustering of Protein Fragments. *Algorithms for Molecular Biology*, *7*, 16.

Fujitsuka, Y., Chikenji, G., & Takada, S. (2006). SimFold Energy Function for De Novo Protein Structure Prediction: Consensus with Rosetta. *Proteins*, *62*, 381–398.

Gay, S., Fages, F., Martinez, T., & Soliman, S. (2011). A Constraint Program for Subgraph Epimorphisms with Application to Identifying Model Reductions in Systems Biology. In *Workshop on Constraint-Based Methods for Bioinformatics*.

Gebser, M., Schaub, T., Thiele, S., & Veber, P. (2011). Detecting Inconsistencies in Large Biological Networks with Answer Set Programming. *Theory and Practice of Logic Programming*, *11*(2-3), 323–360.

Gecode Team (2013). Gecode: Generic Constraint Development Environment. Available from `http://www.gecode.org`.

Go, N., & Scheraga, H. (1970). Ring Closure and Local Conformational Deformations of Chain Molecules. *Macromolecules*, *3*, 178–187.

Graca, A., Marques-Silva, J., Lynce, I., & Oliveira, A. (2011). Haplotype Inference with Pseudo-Boolean Optimization. *Annals of OR*, *184*(1), 137–162.

Guns, T., Sun, H., Marchal, K., & Nijssen, S. (2010). Cis-regulatory Module Detection Using Constraint Programming. In *IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 363–368.

Handl, J., Knowles, J., Vernon, R., Baker, D., & Lovell, S. (2012). The Dual Role of Fragments in Fragment-Assembly Methods for De Novo Protein Structure Prediction. *Proteins: Structure, Function and Bioinformatics*, *80*(2), 490–504.

Hartenberg, R., & Denavit, J. (1995). A Kinematic Notation for Lower Pair Mechanisms Based on Matrices. *Journal of Applied Mechanics*, *77*, 215–221.

Hegler, J., Lätzer, J., Shehu, A., Clementi, C., & Wolynes, P. (2009). Restriction Versus Guidance in Protein Structure Prediction. *Proc Natl Acad Sci U.S.A.*, *106*(36), 15302–15307.

Jacobson, M., Pincus, D., Rapp, C., Day, T., Honig, B., Shaw, D., & Friesner, R. (2004). A Hierarchical Approach to All-atom Protein Loop Prediction. *Proteins*, *55*, 351–367.

JaCoP Team (2012). JaCoP web page, visited November 2012.. Available from `http://www.jacop.eu`.

Jamroz, M., & Kolinski, A. (2010). Modeling of Loops in Proteins: a Multi-method Approach. *BMC Struct. Biol.*, *10*(5).

Jauch, R., Yeo, H., Kolatkar, P. R., & Clarke, N. D. (2007). Assessment of CASP7 Structure Predictions for Template Free Targets. *Proteins*, *69*, 57–67.

Jones, D. (2006). Predicting Novel Protein Folds by using FRAGFOLD. *Proteins*, *45*, 127–132.

Karplus, K., Karchin, R., Draper, J., Casper, J., Mandel-Gutfreund, Y., Diekhans, M., & Source., R. H. (2003). Combining local structure, fold-recognition, and new fold methods for protein structure prediction. *Proteins*, *53*(6), 491–497.

Karplus, M., & Shakhnovich, E. (1992). Protein Folding: Theoretical Studies of Thermodynamics and Dynamics. In *Protein Folding*, pp. 127–195. WH Freeman.

Kim, D. E., Blum, B., Bradley, P., & Baker, D. (2009). Sampling Bottlenecks in De novo Protein Structure Prediction. *Journal of Molecular Biology*, *393*(1), 249 – 260.

Kinch, L., Yong Shi, S., Cong, Q., Cheng, H., Liao, Y., & Grishin, N. V. (2011). CASP9 assessment of free modeling target predictions. *Proteins*, *79*, 59–73.

Kirillova, S., Cortes, J., Stefaniu, A., & Simeon, T. (2008). An NMA-Guided Path Planning Approach for Computing Large-Amplitude Conformational Changes in Proteins. *Proteins: Structure, Function, and Bioinformatics*, *70*(1), 131–143.

Kolodny, R., Guibas, L., Levitt, M., & Koehl, P. (2005). Inverse Kinematics in Biology: The Protein Loop Closure Problem. *The International Journal of Robotics Research*, *24*(2-3), 151–163.

Krippahl, L., & Barahona, P. (2002). Psico: Solving Protein Structures with Constraint Programming and Optimization. *Constraints*, *7*(4-3), 317–331.

Krippahl, L., & Barahona, P. (2005). Applying Constraint Programming to Rigid Body Protein Docking. In *Principles and Practice of Constraint Programming*, pp. 373–387. Springer Verlag.

Krippahl, L., & Barahona, P. (1999). Applying Constraint Programming to Protein Structure Determination. In *Principles and Practice of Constraint Programming*, pp. 289–302. Springer Verlag.

Larhlimi, A., & Bockmayr, A. (2009). A New Constraint-Based Description of the Steady-State Flux Cone of Metabolic Networks. *Discrete Applied Mathematics*, *157*(10), 2257–2266.

LaValle, S. (2006). *Planning Algorithms*. Cambridge University Press.

Lazaridis, T., Archontis, G., & Karplus, M. (1995). Enthalpic Contribution to Protein Stability: Atom-Based Calculations and Statistical Mechanics. *Adv. Protein Chem.*, *47*, 231–306.

Lee, J., Kim, S., Joo, K., Kim, I., & Lee, J. (2004). Prediction of Protein Tertiary Structure using Profesy, a Novel Method Based on Fragment Assembly and Conformational Space Annealing. *Proteins*, *56*(4), 704–714.

Lee, J., Lee, D., Park, H., Coutsias, E., & Seok, C. (2010). Protein Loop Modeling by Using Fragment Assembly and Analytical Loop Closure. *Proteins*, *78*(16), 3428–3436.

Liu, P., Zhu, F., Rassokhin, D., & Agrafiotis, D. (2009). A Self-organizing Algorithm for Modeling Protein Loops. *PLOS Comput Biol*, *5*(8).

Lovell, S., Davis, I., Arendall, W., de Bakker, P., Word, J., Prisant, M., Richardson, J., & Richardson, D. (2003). Structure Validation by C_α Geometry: ϕ, ψ and C_β Deviation. *Proteins*, *50*, 437–450.

Mann, M., & Dal Palù, A. (2010). Lattice Model Refinement of Protein Structures. In *Workshop on Constraint-Based Methods for Bioinformatics*.

Micheletti, C., Seno, F., & Maritan, A. (2000). Recurrent oligomers in proteins: an optimal scheme reconciling accurate and concise backbone representations in automated folding and design studies. *proteins*, *40*(4), 662–674.

Moll, M., Schwarz, D., & Kavraki, L. (2007). *Roadmap Methods for Protein Folding*. Humana Press.

Molloy, K., Saleh, S., & Shehu, A. (2013). Probabilistic Search and Energy Guidance for Biased Decoy Sampling in Ab-Initio Protein Structure Prediction. *IEEE/ACM Trans. Comput. Biology Bioinform*, *PrePrint*.

Neumaier, A. (1997). Molecular Modeling of Proteins and Mathematical Prediction of Protein Structure. *SIAM Review*, *39*, 407–460.

Noonan, K., O'Brien, D., & Snoeyink, J. (2005). Protein Backbone Motion by Inverse Kinematics. *International Journal of Robotics Research*, *24*(11), 971–982.

Olson, B. S., Molloy, K., & Shehu, A. (2011). In Search of the Protein Native State with a Probabilistic Sampling Approach. *J. Bioinformatics and Computational Biology*, *9*(3), 383–398.

Raman, S., Vernon, R., Thompson, J., Tyka, M., Sadreyev, R., Pei, J., Kim, D., Kellogg, E., DiMaio, F., Lange, O., Kinch, L., Sheffler, W., Kim, B.-H., Das, R., Grishin, N. V., & Baker, D. (2009). Structure Prediction for CASP8 with All-atom Refinement using Rosetta. *Proteins*, *77*(Suppl. 9), 89–99.

Rapp, C. S., & Friesner, R. A. (1999). Prediction of Loop Geometries using a Generalized Born Model of Solvation Effects. *Proteins*, *35*, 173–183.

Ray, O., Soh, T., & Inoue, K. (2010). Analyzing Pathways Using ASP-Based Approaches. In *Algebraic and Numeric Biology, 4th International Conference*, pp. 167–183. Springer Verlag.

Rossi, F., van Beek, P., & Walsh, T. (2006). *Handbook of Constraint Programming*. Elsevier Science Inc.

Rufino, S., Donate, L., Canard, L., & Blundell, T. (1997). Predicting the Conformational Class of Short and Medium Size Loops Connecting Regular Secondary Structures: Application to Comparative Modeling. *J. Mol. Biol.*, *267*, 352–367.

Shehu, A. (2009). An Ab-Initio Tree-Based Exploration to Enhance Sampling of Low-Energy Protein Conformations. In *Proceedings of Robotics: Science and Systems V*.

Shehu, A. (2010). *Conformational Search for the Protein Native State*, pp. 431–452. John Wiley & Sons. Inc.

Shehu, A., & Kavraki, L. (2012). Modeling Structures and Motions of Loops in Protein Molecules. *Entropy*, *14*, 252–290.

Shen, M., & Sali, A. (2006). Statistical Potential for Assessment and Prediction of Protein Structures. *Protein Sci*, *15*, 2507–2524.

Shih, E., & Hwang, M.-J. (2011). On the Use of Distance Constraints in Protein-Protein Docking Computations. *Proteins: Structure, Function, and Bioinformatics*, *80*(1), 194–205.

Shmygelska, A., & Hoos, H. (2005). An Ant Colony Optimisation Algorithm for the 2D and 3D Hydrophobic Polar Protein Folding Problem. *BMC Bioinformatics*, *6*, 30–52.

Shmygelska, A., & Levitt, M. (2009). Generalized Ensemble Methods for De Novo Structure Prediction. *Proceedings of the National Academy of Science (USA)*, *106*(5), 1415–1420.

Simoncini, D., Berenger, F., Shrestha, R., & Zhang, K. (2012). A Probabilistic Fragment-Based Protein Structure Prediction Algorithm. *PLOS One*, *7*(7), e38799.

Simons, K., Kooperberg, C., Huang, E., & Baker, D. (1997). Assembly of Protein Tertiary Structures from Fragments with Similar Local Sequences using Simulated Annealing and Bayesian Scoring Functions. *J. Mol. Biol.*, *268*, 209–225.

Skolnick, J., Fetrow, J., & Kolinski, A. (2000). Structural Genomics and its Importance for Gene Function Analysis. *Nat. Biotechnology*, *18*(3), 283–287.

Soto, C., Fasnacht, M., Zhu, J., Forrest, L., & Honig, B. (2008). Loop Modeling: Sampling, Filtering, and Scoring. *Proteins: Structure, Function, and Bioinformatics*, *70*, 834–843.

Spassov, V., Flook, P., & Yan, L. (2008). LOOPER: A Molecular Mechanics-based Algorithm for Protein Loop Prediction. *Protein Eng*, *21*, 91–100.

Stuckey, P. J., Becket, R., & Fischer, J. (2010). Philosophy of the MiniZinc Challenge. *Constraints*, *15*(3), 307–316.

Sussmann, G., & Steele, G. (1980). CONSTRAINTS: A Language for Expressing Almost-Hierarchical Descriptions. *Artificial Intelligence*, *14*(1), 1–39.

Sutherland, I. (1963). Sketchpad: A Man-Machine Graphical Communication System. Tech. rep. 296, Lincoln Laboratory, MIT.

Swedish Institute for Computer Science (2012). SICStus Prolog Home Page. `http://www.sics.se/sicstus/`.

Thebault, P., de Givry, S., Schiex, T., & Gaspin, C. (2005). Combining Constraint Processing and Pattern Matching to Describe and Locate Structured Motifs in Genomic Sequences. In *Fifth Workshop on Modeling and Solving Problems with Constraints*, pp. 53–60.

Tsai, Y., Huang, Y., Yu, C., & Lu, C. (2004). MuSiC: A Tool for Multiple Sequence Alignment with Constraints. *Bioinformatics*, *20*(14), 2309–2311.

Xiang, Z., Soto, C., & Honig, B. (2002). Evaluating Conformal Energies: The Colony Energy and its Application to the Problem of Loop Prediction. *PNAS*, *99*, 7432–7437.

Xu, D., & Zhang, Y. (2012). Ab Initio Protein Structure Assembly Using Continuous Structure Fragments and Optimized Knowledge-based Force Field. *Proteins*, *80*(7), 1715–1735.

Yang, R. (1998). Multiple Protein/DNA Sequence Alignment with Constraints. In *International Conference on Practical Applications of Constraint Programming*.

Yap, R. (2001). Parametric Sequence Alignment with Constraints. *Constraints*, *6*, 157–172.

Yap, R., & Chuan, H. (1993). A Constraint Logic Programming Framework for Constructing DNA Restriction Maps. *Artificial Intelligence in Medicine*, *5*(5), 447–464.

Yue, K., & Dill, K. (2000). Constraint Based Assembly of Tertiary Protein Structures from Secondary Structure Elements. *Proteins Science*, *9*(10), 1935–1946.

Zhang, M., & Kavraki, L. (2002). A New Method for Fast and Accurate Derivation of Molecular Conformations. *Journal of Chemical Information and Computer Sciences*, *42*(1), 64–70.

Zhang, Y., & Hauser, K. (2013). Unbiased, Scalable Sampling of Protein Loop Conformations from Probabilistic Priors. *BMC Structural Biology*, *(to appear `http://www.indiana.edu/~motion/slikmc/papers/BMC_Zhang.pdf`)*.

Zhou, H., & Zhou, Y. (2002). Distance-scaled, Finite Ideal-gas Reference State Improves Structure-derived Potentials of Mean Force for Structure Selection and Stability Prediction. *Protein Sci*, *11*, 2714–2726.