A CONSTRAINT PROGRAMMING APPROACH FOR THE ANALYSIS OF PROTEINS CONFORMATIONS VIA FRAGMENT ASSEMBLY

BY

FERDINANDO FIORETTO

A thesis submitted to the Graduate School

in partial fulfillment of the requirements

for the degree

Master of Science

Subject: Computer Science

New Mexico State University

Las Cruces, New Mexico

November 2011

"A Constraint Programming approach for the analysis of proteins conformations via Fragment Assembly," a thesis prepared by Ferdinando Fioretto in partial fulfillment of the requirements for the degree, Master of Science, has been approved and accepted by the following:

Linda Lacey
Dean of the Graduate School

Enrico Pontelli
Chair of the Examining Committee

Date

Committee in charge:

Dr. Enrico Pontelli, Chair

Dr. Champa Sengupta-Gopalan

Dr. Enrico Pontelli

Dr. Son Cao Tran

DEDICATION

This Thesis is dedicated to my family: my mother, Anna, my father, Biagio, and my sister Ilaria.

ACKNOWLEDGMENTS

I thank Enrico Pontelli, my advisor, for his guidance and encouragement in my research topics and for making this Thesis possible. Enrico allowed me to work freely on what I present in this Thesis, and supported me in numberless discussions with a fresh, outside look, making my Master studies a great experience. My great gratitude goes to Alessandro Dal Palú, that originally offered me to join this project, and literally guided me into such research area, sharing with an uncountable number of ideas and suggestions. To him I owe nearly everything I know about computational biology.

The Knowledge representation, Logic, and Advanced Programming Laboratory at New Mexico State University has been an inspiring and gratifying working environment for the past year, giving me the opportunity to meet extraordinary people. I especially want to thank Khoi Nguyen, for the many interesting discussions and suggestions.

My friends and my family have been a constant source of support. Thanks to my parents, Anna and Biagio, and my sister, Ilaria, constantly present in my life. I also thank my friends at the NMSU, in particular, Oyoung, Reza and Kabi, with whom i shared many moments, in particular during the writing of this Thesis.

Field of Study

Major field: Constraint Programing and Computational Biology

Protein Structure Prediction

ABSTRACT

A CONSTRAINT PROGRAMMING APPROACH FOR THE ANALYSIS OF PROTEINS CONFORMATIONS VIA FRAGMENT ASSEMBLY

BY

FERDINANDO FIORETTO

Master of Science in Computer Science

New Mexico State University

Las Cruces, New Mexico, 2011

Dr. Enrico Pontelli, Chair

Understanding the tridimensional structure of proteins is essential to biomedical investigation, including drug design and the study of diseases such as Parkinson and Alzheimer.

This Thesis presents a novel approach to the protein structure analysis based on *Constraint Programming* methodologies. We employ fragment assembly techniques to derive the final protein conformation, using information derived from homology, prediction and/or experimental data. We show that information collected from various sources can be included as constraints describing the molecule's properties, guiding the prediction to achieve enhancement in performances and quality of the results.

TABLE OF CONTENTS

LIST OF ALGORITHMS xii

LIST OF TABLES xiii

LIST OF FIGURES xiv

INTRODUCTION xvi

1 CONSTRAINT PROGRAMMING 1

1.1 Introduction 1

1.2 Basic Concepts of Constraint Programming 3

1.2.1 Representation 3

1.2.2 Equivalence of CSPs 6

1.3 Constraint Propagation 8

1.4 Propagation techniques 10

1.5 Search Trees 12

2 BIOLOGICAL BACKGROUNDS 18

2.1 How cells work 18

2.2 Proteins 20

2.2.1 Amino Acids and Peptide bond 21

2.2.2 Levels of Protein structure 24
2.2.2.1 Secondary structures 26
2.2.2.2 Loops 27
2.2.2.3 Tertiary structures 28
2.3 Protein Structure Prediction 28
2.3.1 Motlen globule state 30
3 THE PROTEIN STRUCTURE PREDICTION PROBLEM 33
3.1 Model Description 34
3.1.1 The Backbone 35
3.1.2 Centroids 37
3.1.3 Structures, intuitions and block models 38
3.1.3.1 Fragments 39
3.1.3.2 Pairs 41
3.1.4 Fragment Assembly 45
3.1.5 Clustering 47
3.1.6 Energy Model 54
3.2 A brief analysis on complexity 58
3.3 Related works 61
4 FIASCO: FRAGMENT-BASED INTERACTIVE ASSEMBLY FOR PROTEIN STRUCTURE PREDICTION WITH CONSTRAINTS 66
4.1 Introduction 67
4.2 Problem modeling 69
4.2.1 Domain Manipulation 69

4.2.2 Modeling: Variables and Constraints 71
4.2.3 Constraints 73
4.2.3.1 Distance Constraints. 73
4.2.3.2 Energy Constraints 75
4.2.3.3 Geometric constraints 77
4.3 Constraint Solving 79
4.3.1 Searching solutions 79
4.3.2 Propagation and consistency 82
4.3.2.1 alldistant propagator. 82
4.3.2.2 energy propagator. 83
4.3.2.3 Fragment propagator. 84
4.3.2.4 Pair propagator. 85
4.4 Implementation details 87
4.4.1 Variables and constraints representation 87
4.4.2 Search Space 92
4.5 Results 99
4.5.1 Methods 99
4.5.2 Efficiency 102
4.5.3 Quality of the results 104
5 FIASCO: A PARALLEL AND MULTITHREAD SOLUTION 110
5.1 Parallelization of FIASCO 111
5.1.1 Concepts of search parallelism 111
5.1.2 Agents 113

5.1.3 Tasks and Task Queues 114
5.1.3.1 Task Description 116
5.2 The multi-thread model 120
5.2.1 Worker 120
5.2.2 Scheduler 123
5.3 Scheduling and Communication 125
5.3.1 Thread communication protocol. 125
5.3.2 Cluster level parallelization 127
5.3.2.1 Messages 128
5.3.2.2 Termination detection. 130
5.4 Load balancing 134
5.4.1 Partitioning the search space for the initial task 134
5.4.2 Statistical Information Table 137
5.4.3 A load balancing strategy 140
5.4.4 Some Implementation details 146
5.5 Experimental results 146
5.5.1 Performances 148
5.5.2 Scalability 153
6 ONGOING AND FUTURE WORKS 159
6.1 Current works and Special studies 160
6.1.1 Cases of Study 161
6.2 Intuitions, ideas and future directions 166
CONCLUSIONS 170

REFERENCES 172

LIST OF ALGORITHMS

1 Generalized AC3. 12

2 FIASCO's search procedure. 80

3 Arc consistency procedure 81

4 The algorithm for the variable selection strategy 97

5 FIASCO's path reconstruction procedure. 122

6 A general overview of the scheduler algorithm 124

7 Termination detection algorithm for agent A_k 133

8 Successor function algorithm for agent A_k 145

LIST OF TABLES

2.1 Table of the 20 amino acids. 22

4.1 Amino acid clustering classes. 100

4.2 Efficiency results 104

4.3 Qualitative results 105

5.1 Parallel experimental results 152

5.2 Computational time gain and qualitative results 154

LIST OF FIGURES

1.1 One of the 96 solutions to the 8-queens problem. 6

1.2 Propagation and splitting in a CSP. 14

1.3 Complete labeling tree for two variables ordering. 16

1.4 The effect of variable ordering in a prop labeling tree. 17

2.1 Central dogma of gene expression. 20

2.2 Amino acid chemical structure. 22

2.3 Location of amino acids in a protein structure. 23

2.4 A polypeptide. 25

2.5 An α-helix (left) and a β-sheet (right). 27

3.1 Bend angle (left). Torsion angle (right). 36

3.2 Full atoms and centroids representation. 38

3.3 Loop flexibility in the structure prediction. 42

3.4 Properties (1) (left) and (2) (right). 44

3.5 The anti-transitive closure for the pair relation ρ. 45

3.6 Assembly of two consecutive fragments 47

3.7 The amino acid clustering partition. 50

4.1 2ZKO prediction 107

5.1 Concurrent search tree exploration 112

5.2 Heavy and Light task description. 116

5.3 Light task queue population. 122

5.4 Thread and cluster communication within an agent 127

5.5 Parallel partitioning of the search space 137

5.6 Parallel Speddup 150

5.7 Parallel idle time 151

6.1 REBOV VP35 prediction 162

6.2 3EMN_X prediction 165

INTRODUCTION

In this Thesis we present an approach to tackle the *Protein Structure Prediction* problem using *Constraint Programming*.

Proteins are macromolecules of fundamental importance in the way they regulate vital function in all living organism. They participate in operations such as immunological defense, metabolic catalysis as well as forming the basis of cellular structural elements. The structure of a protein plays a central role in the domain of its functionality: *missfolded* proteins (those that are not described by a correct tridimensional structure) lose the ability to perform their natural physiological functions. For this reason the knowledge of the three-dimensional structure of proteins is essential for understanding diseases such as Parkinson and Alzheimer, and for biomedical investigation targeted in drug design.

In this work we focus on the problem of determining the tridimensional structure of a protein, given its chemical composition (*ab-inito* prediction).
We developed *FIASCO (Fragment Interactive Assembly for protein Structure with*

COnstraints), a novel framework dedicated to protein structure analysis, that uses Constraint Programming methodologies emerged from Logic Programming and Artificial Intelligence. Our solution is motivated by previous investigation in this field that show the feasibility of the proposed method [Dal06, Bac98b, BW03, Bac98a, BWBB99, DDFP11]. In particular we follow the approach of [DDFP11] in which Dal Palú et al. presented a declarative approach to the protein structure prediction problem, that relies on the *Fragment Assembly* model; they name their system *TUPLES*.

In our solution, the main constraint in use casts the local protein structural behavior and it is modeled with the concept of *fragment*. We adopt the *fragment assembly* model to reconstruct the final 3D protein structure. Fragments are built from a statistically significant library of short peptides extracted from already sequenced proteins, such as *NCBI* and *PDB*, and altogether allow us to use information about homologous sequences, alignments, and known structures of identified homologues. The consistent homologies with known structures are segmented into fragments which may range from 4 to hundreds amino acids and have various preferred conformations. Our solver reassembles them based on geometric compatibility constraints and free energy evaluation.

The main difference of FIASCO with respect to other protein structure predictors (e.g. *Rosetta*) is its inherent modular structure, that allow the final user to arbitrarily design ad-hoc constraints aimed at capturing properties of the

target sequence. A typical example is represented by the case in which specific information, related to a given target protein, is available (e.g. structural homologies, relative position of different protein regions, the presence of an active site in a determined geometric area, etc.). Such knowledge can be included in our model trough a constraint representation, without the necessity of reshaping the framework. In addition, modularity makes our solver suitable to exploit distinct classes or proteins, when they can be better described using different energy models.

The main contribution of this Thesis is the FIASCO solver and its careful imperative implementation, that allow us to gain over 3 order of magnitudes in terms of computational time, when compared to a declarative version.

From a computational complexity prospective, the protein structure prediction problem is proven to be NP-complete, when described by discrete model [CGP$^+$98, DDP06], and the exponential growth is in direct relation with the length of a protein. However we are not interested in studying unbounded instances of the problem, hence there is a great interest in producing efficient solutions. Biologists and researcher interested in the analysis of protein behaviors and structure are concerned in structures with maximum length of roughly $200 - 300$ amino acids, since bigger proteins are usually composed of small particles that can be traced to cases in which their length falls behind to the smaller proteins.

To guarantee scalability, we implemented a parallel version of FIASCO, exploiting a coarse-grained parallelism at a cluster level and using multiple threads

running on multi-core CPU's.

This Thesis is organized as follows. In Chapter 1 we provide a brief overview of *Constraint Programming* methodologies. Chapter 2 is dedicated at describing the biological backgrounds necessary to study the problem.

Chapter 3 defines the problem of the protein structure prediction, formalizing it according to the model adopted in this work. In this Chapter we describe formally the fragment assembly methodology, and the energy model used in our framework. At the end of the Chapter a brief summary of related works is reported.

In Chapter 4 we present a formal description and the implementation details of our solver, targeted in the ab-initio protein structure prediction and analysis. We provide a detailed discussion of the Constraint Programming formalisms adopted, describing the modeling of variables and constraints—aimed at capturing properties of amino acids and local structures—, and propagators—to prune the search space. In Chapter 5 we introduce a parallelization of the system, exploiting an MPI-based cluster distributed solution, in multi-core platforms with a thread-based approach. We focus on the scheduling and load balancing techniques adopted.

Both Chapter 4 and Chapter 5 are supplemented with a Section dedicated to the experimental results, aimed at testing the behavior of the proposed solution in the relative environments.

In Chapter 6 we present the current ongoing works, and discuss the preliminary

results on two case of study, in which we apply our work to two sets of unknown proteins,one associated to the *Ebola* virus and one derived from the study of the inner ear of the *Xenopous laevis*. Finally, we present different ideas and intuitions for the future directions, targeted at improving the quality of the predictions and achieving better computational speedup.

CHAPTER

ONE

CONSTRAINT PROGRAMMING

In this chapter we introduce the main characters of the Constraint Programming paradigm. We focus on the description of the essential concepts with the aim of providing an overview of the methodologies backgrounds on which this work relies. For a more detailed treatment of the argument we address the reader to the classical Constraint Programming books by Rossi, Van Deek and Walsh [RvBW06] or by Apt [Apt03].

1.1 Introduction

Constraint programming is a powerful paradigm for solving combinatorial problems. In programming languages, constraint programming differs from the classical notion of conventional programming for its clear separation between *model* and *solver*. The problem is modeled declaratively, in terms of constraints to be satisfied so that a consistent solution can be collected. In a successive

phase, this high-level model is given to a constraint solver, which aim is to find the solution(s) that verifies the model.

Constraint programming is successfully applied on a wide range of domains, such us job scheduling, routing, planning, interactive graphic system, numerical computations, networks, bioinformatics.

The basic idea in this programming paradigm relies on the use of relations that should hold among entities of the problem. A solution is the set of those values, associated to each entity, that satisfies the model—this notion is referred as a constraint satisfaction problem (CSP). As an example consider the problem of scheduling an industrial production chain, the entity of the problem (variables) might be the time necessary at producing a given object, its cost, and the resources needed to perform such production. The constraints might be on the limited budget, and on the availability of the resources and their use for a limited time.

A constraint solver is aimed at solving a problem as the one described above, represented in terms of variables and constraints, and find an assignment to all the variables of the model that satisfies the constraints over them.

The space of all the possible assignments is often referred as the search space of the problem, and a constraint solver explores it by using systematic or local search methods. Since the size of the problems tackled is usually intractable, constraint reasoning is employed to reduce the search space both by explicitly stating properties of the constraint problem in the modeling phase, and in the use

of constraint propagation. The former is usually tackled by exploiting constraint symmetries, or imposing an order on the instantiation of the variables of the problem. The latter embeds any form of reasoning which results in explicitly removing values from the set of those in which some variables can range, to prevent the unsatisfybility of some constraints.

In this Chapter we provide an introduction to the constraint programming paradigm, and to the constraint satisfaction techniques introduced above.

1.2 Basic Concepts of Constraint Programming

Constraint satisfaction problems (CSPs) plays a central role in Constraint Programming. The following sections are aimed at defining this concept formally.

1.2.1 Representation

Consider a sequence of variables $X = \langle x_1, x_2, \ldots, x_k \rangle$. Every variable x_i is associated with a domain D_i, that represents the set of values in which the variable can range.

Definition 1.2.1 (Constraint)**.** A constraint c is a relation defined on the domains of a sequence of variables X, that is: $c \subseteq D_1 \times D_2 \times \ldots \times D_k$, and it specifies the values of the variables that are compatible with each other.

When $k = 1$ the constraint is said *unary*, for $k = 2$ the constraint is said *binary*, for $k > 2$ we denote it with *k-ary* constraint.

The sequence of variables X over which a constraint c is defined, is called the *scheme* of c and denoted with $X(c)$. A tuple $\langle d_1, \ldots, d_k \rangle \in D_1 \times \ldots \times D_k$ *satisfies* a constraint c if (and only if) $\langle d_1, \ldots, d_k \rangle \in c$.

Definition 1.2.2 (CSP)**.** A Constraint Satisfaction Problem (CSP) is a triple $\mathcal{P} = \langle X, D, C \rangle$, where X is a sequence $x_1, \ldots, x_k$ of variables, D is the set of domains associated to the variables in X, and C is a finite set of constraints over a subsequence of X.

A *solution* to a CSP is an assignment of values to all its variables, such that all its constraints are satisfied. More formally, consider a CSP $\mathcal{P} = \langle X, D, C \rangle$. An n-tuple $d_1, \ldots, d_k \in D_1 \times \ldots \times D_k$ satisfies a constraint $c \in C$ on the variables $x_{i_1}, \ldots, x_{i_n}$ if and only if:

$$(d_{1_i}, \ldots d_{1_m}) \in c.$$

A *solution* to the CSP $\mathcal{P} = \langle X, D, C \rangle$ is an n-tuple $d_1, \ldots, d_k \in D_1 \times \ldots \times D_k$ that satisfies every constraint $c \in C$. The set of solutions generated by a CSP $\mathcal{P}$ is denoted $sol(\mathcal{P})$. A constraint $c \in C$ is *consistent* if there is an n-tuple $d_1, \ldots, d_k \in D_1 \times \ldots \times D_k$ satisfying it; a constraint is *inconsistent* otherwise. Similarly, a CSP $\mathcal{P}$ is said to be *consistent* (or satisfiable) if $sol(\mathcal{P}) \neq \emptyset$, and otherwise it is said to be *inconsistent* (or unsatisfiable).

A CSP may be associated to an optimization function f. Informally, the goal of solving such CSPs does not simply relies on finding admissible solutions,

but on finding an optimal solution, according to some optimization criteria.

Definition 1.2.3 (COP)**.** A Constraint Optimization Problem is a pair $\langle \mathcal{P}, f \rangle$, where $\mathcal{P} = \langle X, D, C \rangle$ is a CSP, and $f : sol(\mathcal{P}) \rightarrow \mathbb{R}$ is an optimization function ranging from the set of solutions of $\mathcal{P}$ to real numbers.

Without loss of generality, consider the optimization function f to define a minimization problem; a solution s to a COP $\langle \mathcal{P}, f \rangle$, with $sol(\mathcal{P}) \neq \emptyset$, is the solution of the CSP $\mathcal{P}$ satisfying the:

$$(\forall a \in sol(\mathcal{P}))(f(s) \leq f(a)).$$

Example 1. *The n Queens Problems.* Consider the problem of placing n queens on a chessboard of size $n \times n$, where $n \geq 3$, so that they do not attack each other. Figure 1.1 illustrates such problem for $n = 8$. A CSP $\langle X, D, C \rangle$ representation for such problem is given as follows. Let $X = x_1, \ldots, x_n$ be the set of variables representing the n queens. Each variable has domain $D_i = \{1, \ldots, n\}$ representing the columns of the chessboard. The idea is that the x_i denotes the position of the queen on the i^{th} column of the board. The configuration illustrated in Figure 1.1 corresponds to the sequence of values $(7, 4, 2, 8, 6, 1, 3, 5)$, i.e. the queens from left to right are placed on the 7^{th} row counting from the bottom, on the 4^{th} row, on the 2^{th} row, and so on.

A set of constraints to encode the n queens problem is the following: for $i \in [1, \ldots, n], j \in [i + 1, \ldots, n]$

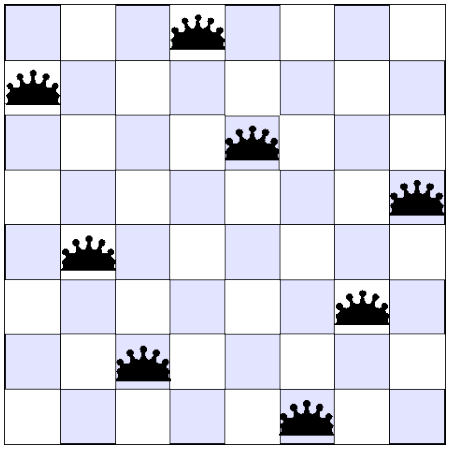

Figure 1.1: One of the 96 solutions to the 8-queens problem.

- $x_i \neq x_j$: every two queens must lie on different rows,
- $x_i - x_j \neq i - j$: every two queens cannot lie on the South-West – North-East diagonal,
- $x_i - x_j \neq i - j$: every two queens cannot lie on the North-West – South-East diagonal.

1.2.2 Equivalence of CSPs

Constraint programming can be viewed as the process of transforming a CSP into an other CSP that is easier to solve. Before stepping into this concept, let us introduce some terminology. Consider a constraint c on a sequence of variables $x_1, \ldots, x_k$ $(k > 0)$ with domains $D_1, \ldots, D_k$. c is said to be *solved* if $c = D_1 \times \ldots \times D_n$ and $c \neq \emptyset$. Similarly a CSP is said *solved* if all its constraints are solved and nonempty, and *failed* if some of its domains or constraints is empty.

Observe that a solved CSP is consistent, while a failed CSP is inconsistent.

Each solution $\langle X, D, C\rangle$ of a given CSP $\mathcal{P} = \langle X, D, C\rangle$ corresponds to a unique solved CSP of the form $\langle X, D, C'\rangle$, where $C' = \{c' \mid c \in C\}$ and c' is defined as the singleton set $\{d_{i_1}, \ldots, d_{i_m}\}$, for each $c \in C$ on a subsequence $x_{i_1}, \ldots, x_{i_m}$ of X.

A common way of solving CSPs, is by transforming them in different easier problems until a solution (respectively all solutions) is found, or the CSP is proven to be failed. If a solved CSP produces one and only one solution, all domains of the solution are singleton; if more then a solution is yielded, then some of its domains have more then one element; clearly if a CSP is failed it yields no solutions.

In order to apply the process introduced above, the transormation of a CSP into another must guarantee that the two are indeed equivalent.

Definition 1.2.4 (Equivalence)**.** Two CSPs $\mathcal{P}'$ and $\mathcal{P}''$ are *equivalent* if and only if they have the same set of solutions.

Example 2. Consider the CSPs:

$$\langle x^2 + y = 3; \quad x = \{0, \ldots, 4\}, y = \{0, \ldots, 3\}\rangle,$$

and

$$\langle 2x + 2y = 6; \quad x = \{0, 1\}, y = \{0, \ldots, 3\}\rangle.$$

Both yields to the set of solutions $x = 0, y = 3$ and $x = 1, y = 2$, therefore they are equivalent.

1.3 Constraint Propagation

Conventional methods adopted to solve a constrained problem involves a from of constraint reasoning to tackle the inherent intractability of the problem of satisfying a CSP. A general application of such form of reasoning is expressed in transforming the original CSP to produce a new simpler, yet equivalent one. The idea of simplicity of a CSP typically refers to narrow domains and/or constraints.

Constraint propagation is a technique used to seek this goal. It embeds any reasoning which consists in explicitly precluding the use of specific variables values that would prevent a given subset of constraints over such variables, to be satisfied. As an example, consider a crossword-puzzle, where one has to fill a six letter-spaces horizontal box with a word from the set of the European countries. Assume also that a new word is inserted in a vertical box, intersecting the second position of the previous considered box, with the letter 'R'. This information propagates a constraint that shrinks the domain of the six-letter European countries to the one having 'R' as the second letter.

When some information about a new status of the problem is propagated, constraint propagation rewrites a CSP into an equivalent one, applying rewriting rules to satisfy the local consistency of the CSP. In general, constraint solvers

aim at either reducing the domains of the considered variables (*domain reduction rules*) or at reducing the considered constraints (*transformation rules*). In other cases *introduction rules* can be used, to add a new constraint to the CSP, inferred by the existing ones. Following, we will briefly introduce some domain reduction rules, that, for efficiency reasons, are the most widely implemented.

Domain reduction rules are *equivalence preserving* (the application of such rules does not change the CSP solution set). They are denoted by the:

$$\frac{\phi}{\psi}$$

where $\phi = \langle X, D, C \rangle$ and $\psi = \langle X', D', C' \rangle$. In the application of domain reduction rules, the new domains are respective subset of the old domains (for $i \in [1, \ldots, n]$, $D' \subseteq D$) and the new constraint set C' is the result of restricting each constraint in C to the corresponding subsequence of the domains $D'_1, \ldots, D'_n$.

We now consider tree of the most implemented rules: *node consistency, arc consistency* and *bounds consistency.*

Node consistency. A CSP $\mathcal{P} = \langle X, D, C \rangle$ is *node consistent* if and only if for each variable $x_i \in X$ every unary constraint c that involves x is such that $c = D_i$.

Arc consistency. A binary constraint $c(x_i, x_j)$ is *arc consistent* if and only if for every $a \in D_i$ there exists a value $b \in D_j$ such that $(a, b) \in c$, and for every value $b \in D_j$ there is a value $a \in D_i$ such that $(a, b) \in c$.

Bounds consistency. A binary constraint $c(x_i, x_j)$ is *bound consistent* if and only if:

i $\big(\exists b \in [\min(D_j), \max(D_j)]\big)\big((\min(D_i), b) \in c\big)$ and
$\big(\exists b \in [\min(D_j), \max(D_j)]\big)\big((\max(D_i), b) \in c\big)$,

ii $\big(\exists a \in [\min(D_i), \max(D_i)]\big)\big((a, \min(D_j)) \in c\big)$ and
$\big(\exists a \in [\min(D_i), \max(D_i)]\big)\big((a, \max(D_j)) \in c\big)$.

A CSP $\langle X, D, C\rangle$ is bound consistent if every binary constraint in C is bound consistent.

1.4 Propagation techniques

In a constraint solver, the procedures for propagating the effect of the local consistencies are grouped together into a unique algorithm.

A standard approach consists of two phases: the first step is aimed at reaching node consistency by checking the unary constraints over all the variables in $\mathcal{P}$; the second step is aimed at reaching arc consistency.

The most well-known algorithm for arc consistency is the AC3 [Mac77] in its generalization to non-binary constraints [Mac75]. The algorithm is illustrated in Figure 1 and it is composed of two components: the *arc revision* and the *AC3* procedure. The arc revision component removes every value $a \in D_i$ that is not consistent with some constraint c. The function `ReviseArc`(x_i, c) takes each value

$a \in D_i$, and looks for a support on c over the space of the relations with scheme $X \backslash \{x_i\}$.

If such a support is not found (Line 4) a it is removed from D_i (Line 6) and this event is signaled by a `changed` flag (Line 7).

The main algorithm, AC3, is a simple loop that revises arcs until a fix point is reached (i.e. no changes occur in the domains of the variables considered). This condition ensures that all domains are consistent with all constraints. The procedure maintains a list Q (often referred as *constraint store*) of all paris (x_i, c) for which the arc consistency on c needs to be checked. At the beginning Q contains all paris (x_i, c) such that $x_i \in X(c)$ (Line 8). The main loop (Line 9–14) revises each element (x_i, c) of Q until either the domain D_i becomes empty (Line 12) or every element in Q has been checked. After the call to the `ReviseArc` routine, if D_i has been modified (i.e. it is `changed`) it may be possible that some other variable x_j has lost its support on a constraint c' involving both x_i and x_j. To take into account such case all pairs (x_j, c'), with $x_i, x_j \in X(c')$ must be pushed back into Q (Line 13). The algorithm guarantees that, when true is returned, all arcs have been revised and all the remaining variables in the domains of D are consistent with all constraints in C.

Algorithm 1: Generalized AC3.

```
1  function: ReviseArc(x_i, c):Boolean
2  changed← False;
3  foreach a ∈ D_i do
4      if ∄ a valid support on c for a then
5          remove a form D_i;
6          return (changed ← True);
7  function: AC3(X):Boolean
8  Q ← {(x_i, x) | c ∈ C, x_i ∈ X(c)};
9  while Q ≠ ∅ do
10     extract (x_i, c) from Q;
11     if Revise(x_i, c) then
12         if D_i = ∅ then return False;
13         else Q ← Q ∪ {(x_j, c') | (c' ∈ C\{c}) ∧ (x_i, x_j ∈ X(c')) ∧ (j ≠ i)};
14         return True;
```

1.5 Search Trees

A constraint solver can typically be described by two components: the *constraint propagator* and the *solution(s) searcher*. Solving a CSP can be expressed as the process of exploring a search tree where each node represents a possible variable value assignment, the arcs connecting nodes expresses the effect of propagating constraints, and a solution is described by a complete path from the root node to a leaf.

More precisely, the search tree associated to a CSP, is the tree derived by repeatedly splitting the original problem in two or more CSPs whenever the constraint propagation does not reach a the global goal.

Splitting a CSP is formalized by the *splitting rules* that involve either a

domain or a constraint splitting.

Domain splitting. Splitting a domain is formalized by a rule that transforms a domain expression into two or more expressions. Let $\mathcal{P} = \langle X, D, C \rangle$ be a CSP, and assume the domains $D_i \in D$ to be finite. The followings are typical domain splitting rules, which correspond to reasoning by cases:

- *Enumeration.*

$$\frac{x_i \in D_i}{x_i \in \{a\} | x_i \in D_i \backslash \{a\}}$$

  where $a \in D_i$.

  The semantics of this rules is expressed by two cases: in the first case, x_i is substituted by a; in the second, the domain D_i is reduced by the element a. It follows, that the original CSP $\mathcal{P}$ is replaced by two CSPs:

  1. $\langle X, D, C' \rangle$, where $D_i = \{a\}$, and
  2. $\langle X, D, C'' \rangle$, where $D_i = D_i \backslash \{a\}$,

  where C' and C'' are the restriction of the constraints in C to the new reduced domain D_i.

- *Labeling.*

$$\frac{x_i \in \{a_1, \ldots, a_k\}}{x_i \in \{a_1\} | \ldots | x_i \in \{a_k\}}.$$

  This rule produces k new CSPs from the original one.

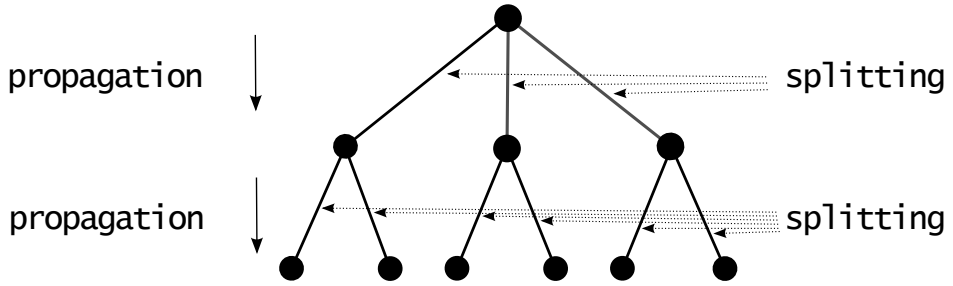

Figure 1.2: Propagation and splitting in a CSP.

- *Bisection.* Let the domain D_i be a continuous interval, denoted with $[a, b]$:

$$\frac{x_i \in [a, b]}{x_i \in [a, \frac{a+b}{2}] | x_i \in [\frac{a+b}{2}, b]}.$$

  In this case, a minimum interval $[a - \epsilon, a + \epsilon]$, should be defined in order to determine the termination of the splitting rule (the splits will be performed until a such size interval is reached). Clearly this rule can be applied to discrete intervals.

An illustration of the use of propagation and splitting rules in a CSP tree representation is given in Figure 1.2. The meaning of this illustration is the following: propagating the effect of constraints from a level of the tree to next enforces domain reduction. Over such (possibly) reduced domains the application of splitting rules (i.e. labeling) branches the tree.

Splitting rules can be employed to create a search tree associated to a CSP. The concept is formally described with the term *prop labeling tree*.

Definition 1.5.1 (Prop labeling tree)**.** Given a CSP $\mathcal{P} = \langle X, D, C \rangle$, a *prop*

labeling tree is a tree that has nodes labelled with expressions of the type:

$$x_1 \in A_1, \ldots, x_k \in A_k,$$

where the A_i are sets denoting the possible value choices for the variables x_i.

The *root node* is labelled with the expression:

$$x_1 \in D_1, \ldots, x_k \in D_k.$$

Each other node $i > 1$ is labeled with:

$$x_1 \in \{a_1\}, \ldots, x_i \in \{a_i\},\ x_{i+1} \in A'_{i+1}, \ldots, x_k \in A'_k,$$

where $A'_j \subseteq A_j$ are the sets of values associated to the variable x_j obtained by propagating the constraints contained at node i.

Each direct descendant of a node i $(1 \leq i < k)$ is either a fail node or a node of the form:

$$x_1 \in \{a_1\}, \ldots, x_i \in \{a_i\}, x_{i+1} \in \{a_{i+1}\},\ x_{i+2} \in A'_{i+2}, \ldots, x_k \in A'_k,$$

such that the assignment $(x_1, a_1), \ldots, (x_{i+1}, a_{i+1})$ is consistent with every constraint $c \in C$.

The nodes at level $i = k$ are *leaves* node and represent a solution for the CSP.

Observe that the number of solutions (or leaves nodes) does not depend on the order of the variable labeling choices. Consider the following example [Apt03]:

$$\langle x < y, y < z; x \in \{1, 2, 3\}, y \in \{2, 3\}, z \in \{1, 2, 3\}\rangle$$

Figure 1.3 shows the complete labeling tree, without employing propagation, for the order of variable instantiations: x, y, z (top) and x, z, y (bottom). It is easy to note that number of solutions is not affected by the variables instantiation order.

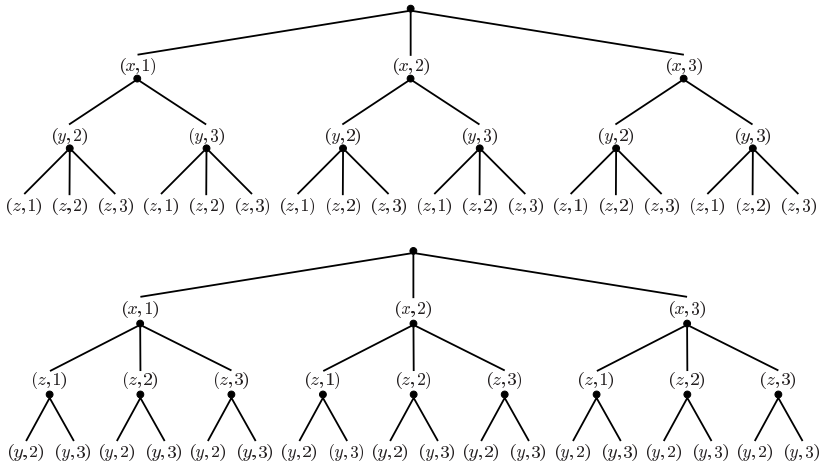

Figure 1.3: Complete labeling tree for two variables ordering.

Let now consider the same example used above, by employing the use of propagation for domain reduction. Figure 1.4 depict this scenario, where the variables instantiation order x, y, z (left) is compared to the order x, z, y (right). The resulting labeling trees have different number of nodes and leaves.

From the example above it is possible to observe the effect of the constraint propagation in action, by pruning those nodes that would not produce any solution for the CSP.

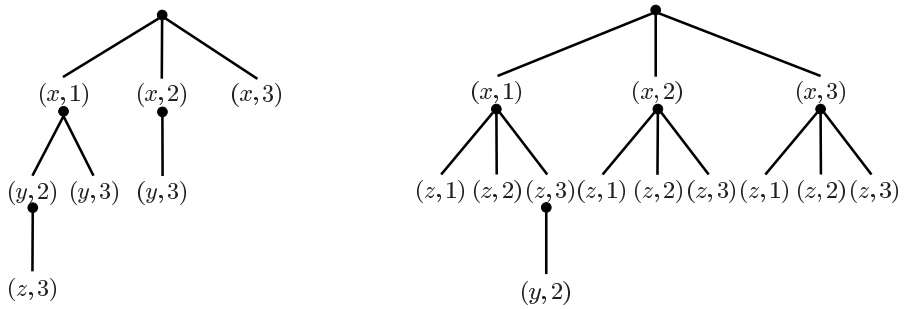

Figure 1.4: The effect of variable ordering in a prop labeling tree.

CHAPTER

TWO

BIOLOGICAL BACKGROUNDS

In this chapter we focus on the description of the fundamental notions necessary to the comprehension of the problem addressed in this work. In Section 2.1 we introduce some basic biological concepts. Section 2.2 provides a more detailed description of proteins, in terms of their composition and structural level of complexity. Finally in section 2.3 we introduce a central argument of this work, the *Protein Structure Prediction* (PSP) problem.

2.1 How cells work

Cells can be defined as the fundamental blocks of any living organism. At a broadest level, one of the most important distinction between organisms is described in terms of cell complexity. The simpler organisms are called *procaryotes*, and are characterized by the lack a cell nucleus, or any other membrane-bound organelles [Aea02]. In such taxonomy can be found some bacteria (like the *Es-*

cherichia coli) and archaea. The most complex cells are called *eukaryotes*, and include all the vertebrates as well as many unicellular organisms (like yeast). Eukaryotes are characterized by complex structures enclosed within membranes. The most recognized structure is the nucleus within which it is carried the genetic material [LNC08].

The cellular nucleus contains the information used to produce all the necessary components to regulate the activities of a living organism. Such information is encoded in a well packed double-helix structure molecule, called *Deoxyribonucleic acid* (DNA). A DNA molecule is a long coiled structure constituted of four building blocks: the *nucleotides*. They are adenine, thymine guanine, and cytosine (A, T, C, and G). The nucleotides are arranged in sequences, and the order in which they are listed determine the encoding for some particular product. Some parts of the DNA are meaningful for life, some other are instead considered "junk" [Bro99]. In the meaningful areas, the DNA contains the sequence of nucleotides that ultimately describe proteins.

Particular sections of the DNA are called *genes*. The genes are, in the Mendelian definition, the smallest unit of heredity that may influence the traits of an organism. We call *genotype* the collection of genes in an organism, and *phenotype* the manifestation of the observable traits encoded in the genes. The expression or suppression of such traits is characterized, in last analysis by the proteins. Protein products determine the phenotypical expression associated to

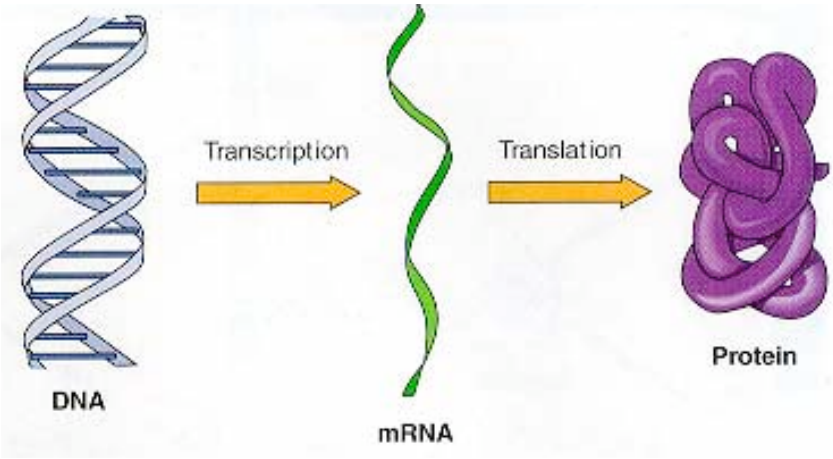

Figure 2.1: Central dogma of gene expression.

some gene.

The life processes of the living organisms are governed by the *central dogma* of biology [Cri58, Cri70]. This concept is illustrated in Figure 2.1: the genetic information encoded into the DNA is transcribed into a molecule called *messenger RNA* (mRNA); the mRNA contains the unit block information necessary for the synthesis (translation) of a particular protein. When the protein is constructed, the gene form which it derives, is said to be *expressed*.

2.2 Proteins

Proteins are organic molecules of fundamental importance in the way they regulate vital functions in all living organisms. They participate in operations such as immunological defense, metabolic catalysis as well as forming the basis of cellular structural elements [Lig74].

The structure of proteins at a cellular level reveals several discrete degrees of complexity, denoted with terms primary, secondary tertiary and quaternary structures. Each level can be seen as incrementally related to build up the final protein structure. In particular the tridimensional shape adopted by a protein plays a fundamental role in the ability of the protein to perform its functions. At a lower level, proteins are constituted by a precise sequence of small blocks, called *amino acids*. The amino acids give the first degree of freedom with respect to the possible shapes a protein can adopt, and we discuss them in the following Section.

2.2.1 Amino Acids and Peptide bond

Amino acids are small molecules, commonly occurring in nature, that can be seen as the unit building blocks of proteins. There are many type of amino acids in nature, but only a subset of them is subjected of the genetic control, as consequence of evolution processes. Proteins are made up of 20 different amino acids, that are commonly denoted with a three or one-letter code, as listed in Table 2.1.

Chemically, an amino acid is characterized by a central carbon, denoted carbon alpha ($C\alpha$), an amino group (NH_2) and a carboxylic group ($COOH$), bounded to the same $C\alpha$ atom. These three elements are constantly present in each amino acid. What characterizes the diversity of such molecules is the presence of an additional group, called *side chain* or R-group, that uniquely characterizes

Aspartic acid	ASP	(D)	Glutamic acid	GLU	(E)
Alanine	ALA	(A)	Arginine	ARG	(R)
Asparagine	ASN	(N)	Cysteine	CYS	(C)
Phenylalanine	PHE	(F)	Glycine	GLY	(G)
Glutamine	GLN	(Q)	Isoleucine	ILE	(I)
Histidine	HIS	(H)	Leucine	LEU	(L)
Lysine	LYS	(K)	Methionine	MET	(M)
Proline	PRO	(P)	Serine	SER	(S)
Tyrosine	TYR	(Y)	Threonine	THR	(T)
Tryptophan	TRP	(W)	Valine	VAL	(V)

Table 2.1: Table of the 20 amino acids.

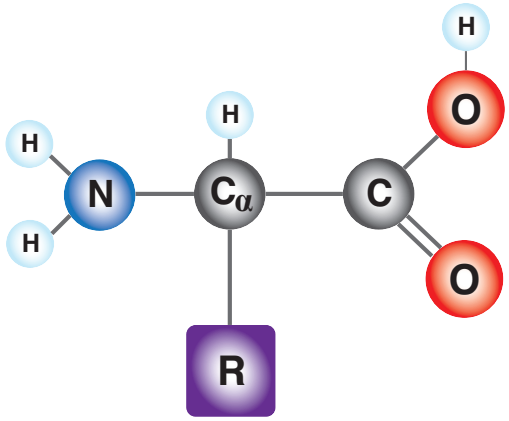

Figure 2.2: Amino acid chemical structure.

the physical and chemical properties of each amino acid. The structure of a general amino acid is shown in Figure 2.2.

In last analysis this molecule characterization makes each of the 20 amino acids unique, and for each of them specifies different roles in the context of protein structures. Depending on the physical and chemical properties of the R-group, for example the propensity to be in contact or repelled by a polar solvent like water, an amino acid is classified as: *hydrophobic*, *polar* or *charged*.

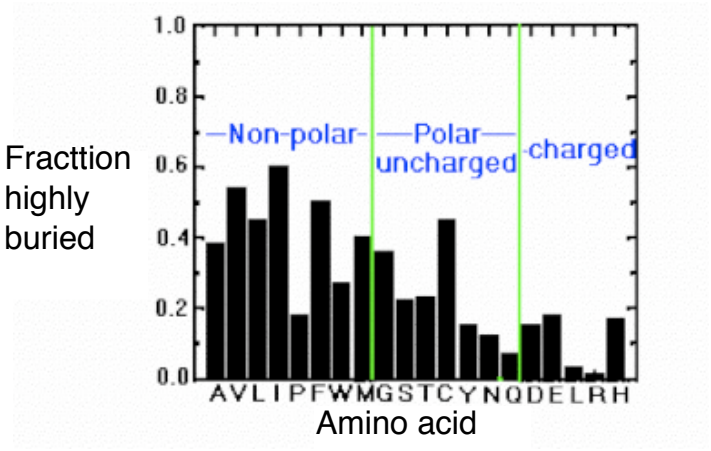

Figure 2.3: Location of amino acids in a protein structure.

The propensity of each of the 20 amino acids of being attracted or repelled by a solvent, strongly influences their distributions within the volume of a protein structure. In turns, many proteins show a hydrophobic core, surrounded by polar and charged residues, to prevent the access to the solvent into the protein core. Figure 2.3 illustrates the location of the 20 amino acids in different region of a protein. The graph shows the relation among the amino acids in terms of their preference to be located in a core protein area, inaccessible to the solvent (buried).

Positively and negatively charged amino acids, often interact forming *salt bridges*, while polar groups commonly participate in the formation of hydrogen bonds with side chains and backbone atoms and with the solvent.

The dimensions of the various R groups, which protrude from the polypeptide chain, the mutual affinity between the hydrophobic or polar groups, the at-

traction between acid and basic groups, and the interactions mentioned above are some of the forces that contribute to shape the conformation of the protein. In turn, such shape strongly contribute to the biological activity of the protein.

An important characteristic of amino acids, is that they can form long linear sequences, called polypeptides, (i.e. the protein). In particular, a strong covalent bond, called *peptide bond*, is formed when the carboxyl group of an amino acid reacts with the amino group of an other. The carbon and nitrogen atoms, associated with peptide bond, constitute the main chain, called protein *backbone*. In the process of the polypeptide formation, the two amino acids at the extremes of the chain, are denoted *C-terminus* characterized by a free carboxyl group, and *N-terminus* having a free amino group.

The peptide bond is an extremely rigid bond, that causes the structure around it to be planar and therefore incapable of rotations. On the other hand the bonds between $C\alpha$–$COOH$ and NH–$C\alpha$ allow a degree of freedom. The angle defined by these rotation are denoted ψ (psi) and ϕ (phi). Figure 2.4 illustrates a chain of amino acids involved in peptide bonds, where the black curved arrows indicated the possible rotations between the $C\alpha$–$COOH$ and NH–$C\alpha$ bonds.

2.2.2 Levels of Protein structure

Protein structure complexity is defined according to a hierarchic level of structural information. We distinguish four structural levels:

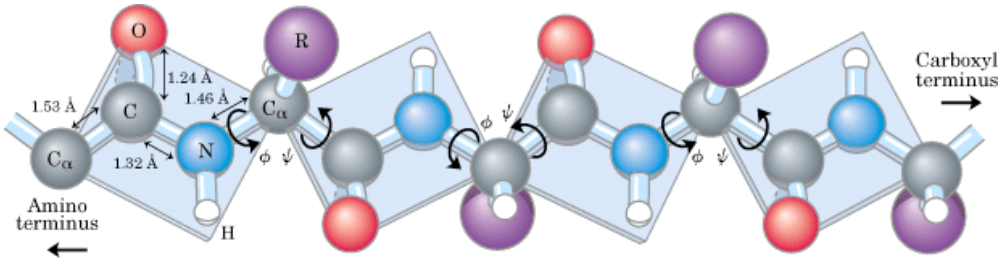

Figure 2.4: A polypeptide.

1. *Primary structure.* It corresponds to the linear amino acid sequence, connected by peptide bonds in a polypeptide chain.

2. *Secondary structure.* Depending on the nature of the amino acids and on the bond angles, the polypeptide tends to shape in more complex conformations, locally stable, which are called motifs of secondary structures.

3. *Tertiary structure.* The further folding of the protein is represented by a three-dimensional structure, produced as a result of the interactions between amino acids located in different parts of the macromolecule. The tertiary structure corresponds to the structure assumed by the protein when it is in the so-called *native state*.

4. *Quaternary structure.* Large proteins are often composed of various subunits. The quaternary structure concerns the topological and spatial arrangement of these subunits.

2.2.2.1 Secondary structures

Secondary structures are the regular repetitions of local structures stabilized by hydrogen bonds. The most common classes of secondary structures are α-helixes and β-sheets (see Figure 2.5).

A α-helix is a right handed spiral in where the planes of the peptide bonds are almost parallel with the axis of the helix, and the amino acid side chains are projected toward the external part of the helix. The most common α-helix is characterized by a regular structure repetition at every 5.4 Å. Such structure is stabilized by the presence of hydrogen bonds between the N–H and the C=O groups.

A β-sheet is a less compacted structure, characterized by the presence of hydrogen bonds between some local areas of the protein. This secondary structure is composed by different β-strands; the connected β-strands can be parallel or anti parallel, according to the direction of its components. In a β-strand the N–H groups lies on one side and the C=O groups on the other, such characteristic imposes the order in which β-strands are connected into the β-sheet. These secondary structures are usually twisted rather than flat, due to the effect of the solvent in which they are immersed.

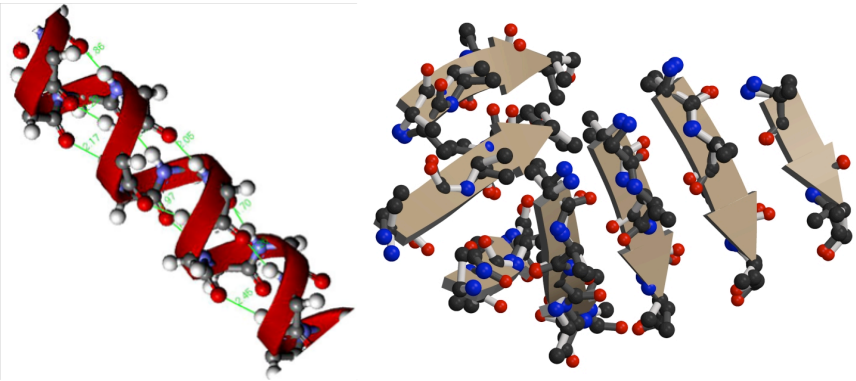

Figure 2.5: An α-helix (left) and a β-sheet (right).

2.2.2.2 Loops

The secondary structure are connected by aperiodic chains of amino acid called *loops*. The length of these region may vary drastically (from 2 to 20 amino acid), but in the majority of the cases they range from 2 to 10 amino acids. The shortest known loops (2–5 amino acids) are called *Hairpin Loops*, also called *reverse turn* for their property of connecting two anti parallel β-strands.

Loops are generally located on the outer regions of the protein and are therefore mostly constituted by polar side chains. In addition, the hydrogen bonds between the amino acids of the loop and the surrounding water molecules are more numerous than those made with the amino acids adjacent to these regions. This characteristic, in contrast to the secondary structures, gives the loops great degree of flexibility.

2.2.2.3 Tertiary structures

When various combination of secondary structures pack together, they form a high compacted body, defined *tertiary structure*. The tertiary structure is characterized by a high stability, due to local interaction established among various side chains. These interaction involves hydrophobic amino acids, dipolar forces between amino acids with opposite charge, hydrogen bonds, and disulfide bridge links. Is the combination of these weak forces that leads the polar parts of the polypeptide to be exposed to the solvent, and consequentially to the creation of a hydrophobic core.

2.3 Protein Structure Prediction

In natural conditions proteins assume a peculiar tertiary structure. The phenomena by which proteins achieve their three-dimensional structure is called *protein folding*. This process can be described as an intramolecular self-assembling operation, in which the protein assumes a specific shape through non-covalent interactions such as hydrogen bonds, hydrophobic forces, Van der Waals forces, etc.

The physiological function of a protein, i.e. enzymatic, catalytic, transporter, is in a one to one relation with its native structure. For this reason, the problem of determine wether or not a protein assumes a correct folding is a problem of great interest.

Despite their complexity, the proteins have a well-defined native state, which is reached by a fast folding process (1μs - 1s). This property is known as the *termodynamic hypothesis* postulated by Christian B. Anfinsen [Lev68] (Nobel Prize for Chemistry in 1972). This postulate states that, at least for small globular proteins, the native structure is encoded in the primary amino acid sequence. In turn, by the entropic consideration, at the environmental conditions at which the folding occurs, the native structure of a protein is a unique, stable and kinetically accessible conformation characterized by the lowest free energy.

The *Protein Structure Prediction* (PSP) problem, is the problem of determining the tridimensional structure of a protein, given its primary sequence.

Levinthal paradox. The duration of the folding process varies according to the nature of the protein. When extrapolated from the cellular context, the slowest folding observed is completed in few minutes or hours, and it requires many intermediate steps. On the other hand, small proteins (within 100 amino acids) typically fold in one step [Jac98] and the majority of protein folds spontaneously in a time of the order of milliseconds or even microseconds [KHE04]—the time to generate a E. Coli is roughly twenty minutes, which means that all the proteins essential for the life of this organism, are produced at a very high speed (at most in the order of minutes).

The Levinthal's paradox [Neu97] observes that, due to the high number of

degrees of freedom in the polypeptide chain movements, if a protein would reach its final conformation, gradually passing through all the possible configurations, it would require a time well beyond the currently estimated age of the universe. This would occur even if the transition from a conformation to another was defined to be a very small interval as picoseconds or nanoseconds.

2.3.1 Motlen globule state

Experimental observations suggests the presence of a unique protein conformation in the native state, evincing the presence of a single global minimum with a significantly lower energy value, when compared to other local minima [OW83]. This view is confirmed by experimental evidence [KB95], which suggests that in nature, the process of protein folding proceeds in two phases: a first rapid one, which leads to a state very close to that of the optimal folding, followed by a longer phase, necessary to stabilize the folding in the final configuration. Such argument is also supported by an evolutionary point of view: proteins executing a specific task must have similar shape—i.e. folding—in order to be successful in such operations (most of these function require "mechanical compatibility" with the interaction sites). In contrast, peptide chains not meeting such requirement would be unable to perform their biological function, and therefore less competitive in evolutionary terms.

Recent studies [HT92, EK87] seem to have reached an agreement on the

thesis that the native state of a protein is represented by a dynamic global minimum reached via a sequence of transition states (local minima), separated from the global minimum by a large energy gap.

Kiefhaber et al. [KB95] show that, under certain physical conditions, there are stable states in which the protein is partially folded. These states, called *molten globules*, show a secondary structure similar to that of the native state of a protein and a very compact tertiary structure.

A note on computational issues in the PSP. One of the most remarkable challenge in computational approaches addressing the PSP problem, is the presence of a great number of local minima in the search landscape. Such characteristic discourages the use of local searches, because of their high chances to get trapped into one of such local optimization point.

In general, the number of local minima is expected to grow exponentially with length of the protein. However most of these local minima might have a large energy contribution, and therefore irrelevant for a global optimization search approach.

For problems rich in symmetries, like the problem of the optimal configuration of a cluster of n identical atoms with a Lennard-Jones interaction the number of local minima is estimated to grow even sharply ($O\big(1.03^{n^2}\big)$) [WV85]. However, the number of "significant" local minima (those for which the associ-

ated free energy is close enough to the global minimum) are likely to grow simply exponentially in n.

In the course of the next Sections we will present an approach, based on Constraint Programming, to address the protein structure prediction problem.

CHAPTER

THREE

THE PROTEIN STRUCTURE PREDICTION PROBLEM

In this chapter, we provide a formalization of the Protein Structure Prediction (PSP) problem accordingly to the model adopted in this work. We make use of the *Constraint Programming* (CP) paradigm, that allow us to encode the problem of finding admissible conformation as a constraint satisfaction problem (CSP). The knowledge about chemical and physics properties of proteins are encoded into constraints to guide the search in the space of the putative conformations. We adopt a simplified model for the protein representation in the cartesian space.

The chapter is structured as follow: in section 3.1 we provide a formal description of the PSP problem and of the model adopted in this Thesis, including its levels of simplification, a description of the fragment assembly technique, and a brief description of the energy function, used as a quantitative descriptor for a structure prediction. In section 3.2 we discuss about some complexity issues of

the defined problem, and in the last Section we report a brief summary of related works.

3.1 Model Description

In the following formalization we focus on proteins composed of amino acids coded by the human genome. Let $\mathbb{A}$ be the set of amino acids, where $|\mathbb{A}| = 20$. Given a protein primary sequence $S = a_1, \ldots, a_n$ ($a_i \in \mathbb{A}$), we represent with the variable $P_i^\alpha \in \mathbb{R}^3$ the position of the $C\alpha$ atom of the amino acid a_i.

Definition 3.1.1 (Folding)**.** A folding is a function $\omega : \underbrace{\mathbb{A} \times \ldots \times \mathbb{A}}_{n} \rightarrow \underbrace{\mathbb{R}^3 \times \ldots \times \mathbb{R}^3}_{n}$. $\omega : \mathbb{N} \rightarrow \mathbb{R}^3$ that maps amino acids of a primary sequence into points of the tridimensional space. The notation $\omega(i) = P_i^\alpha$ describes the mapping ω of the i^{th} amino acid on P_i^α of the tridimensional space.

Definition 3.1.2 (PSP problem)**.** The (ab-initio) protein structure prediction problem is the problem of finding the folding ω that minimizes the free energy measure.

Since proteins tends to adopt the most stable 3D conformation and, the entropic measure describes the stability of a protein tertiary structure (see Section 2.3.1), it follows that the best folding is the one described by the lowest entropy measure.

3.1.1 The Backbone

Recall, from Section 2.2.1 an amino acid is a molecule composesd by an amino group, a carboxylic acid group and a side-chain (that may contain from 1 to 18 atoms). To deal with the complexity of the problem, the model adopted in this work, introduces several degrees of approximation in the protein representation. Each amino acid is treated as atomic unit and shown on the 3D space as a single point describing its $C\alpha$. More formally, every amino acid $a_i \in S$ is represented by the triple (x_i, y_i, z_i), where $x_i, y_i, z_i \in \mathbb{R}$. In the following Sections we will use the terms $C\alpha$ and amino acid interchangeably—when referring to the model in use.

The degrees of freedom for the positions of consecutive amino acids are determined by two components: *bend* and *torsion* angles. A sequence of three consecutive amino acids a_1, a_2, a_3, represented by their $C\alpha$ atoms, with positions $P_1^\alpha, P_2^\alpha, P_3^\alpha$, defined *bend* angle denoted by $\widehat{a_1 a_2 a_3}$. Figure 3.1 (left) shows a fragment of four amino acids, where a bend angle (θ) is emphasized on the amino acid a_2 [DDFP10]. Bend angles tend to be characteristic of the amino acid types involved, and vary little, thus we consider all the bend angles as fixed. Consider now four consecutive amino acids a_1, a_2, a_3, a_4, the angle formed by $n_2 = (a_4 - a_3) \times (a_3 - a_2)$ and $n_1 = (a_3 - a_2) \times (a_2 - a_1)$ is called *torsion angle*. Figure 3.1 (right) shows the torsion angle ϕ on a 4 amino acid fragment [DDFP10]. The backbone of a protein, can be considered as a concatenation of short sequences

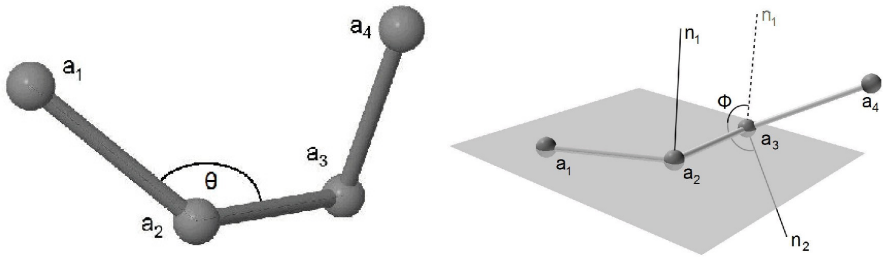

Figure 3.1: Bend angle (left). Torsion angle (right).

of four amino acids. If the torsion angles are known for all the consecutive subsequences, they uniquely describe the 3D positions P_i^α of all the $C\alpha$ atoms of the protein. Given a spatial configuration of four consecutive $C\alpha$ atoms, any of the different conformations arising restricting the rotation around the torsion angle so defined, is called *rotamer*. The rotamers describe the degree of freedom for the position of the side chain. Due to the small variation represented by rotamers our model does not consider an additional degree of movement for the backbone. These information are only used implicitly: the information provided by the torsional angles define wether two consecutive blocks of contiguous amino acids are compatible in the Fragment Assembly model (discussed in the next Section). Avoiding an explicit treatment of bend and torsion angles reduces the number of degrees of freedom and therefore the computational complexity needed to represent and manipulate protein structures.

3.1.2 Centroids

We introduce a supplementary amino acid descriptor, in addition to the $C\alpha$'s position, to represent the side chains. This concept is formalized as a point in the 3D space (denoted as *centroid*) placed at the center of mass of the amino acid side chain. The notation P_i^c is used to represent coordinates of the centroid associated to the i^{th} amino acid of S. Informally, the center of mass of a side chain is the average of all its atoms positions, weighted by their specific weight. The positions for the centroids are assigned once the $C\alpha$ atoms positions are known. In particular, the centroid of the i^{th} residue is constructed by using the locations of $C\alpha_{i-1}$, $C\alpha_i$ and $C\alpha_{i+1}$ as reference and by considering the average center of mass of the amino acid sampled from a non-redundant subset of the PDB. The parameters that uniquely determine its position are: the average $C\alpha$-side-chain center of mass distance, the average bend angle formed by the side chain center-of-mass-$C\alpha_i$-$C\alpha_{i+1}$, and the torsional angle formed by the $C\alpha_{i-1}$-$C\alpha_i$-$C\alpha_{i+1}$-side-center of mass.

It is important to mention that, the introduction of the centroid model allow us to use a more refined energy model, that in turn results in a more accurate prediction. In the CP representation model adopted by our system, an energy function can be modularly enriched by including additional parameters such us side chains contact potential contributions. Moreover, once the positions of the

all $C\alpha$ atoms and centroids are known, the structure of the protein is already sufficiently determined, i.e., the position of the remaining atoms can be identified almost deterministically with a reasonable accuracy.

Figure 3.2 [DDFP11] helps to appreciate the role played by the centroids w.r.t. protein volume representation, showing a comparison between the full atom and the centroids protein model. In the two illustrations the backbone of the polymer is represented by thick lines. The left hand side shows a full atom representation for the side chains, while the right hand side shows the centroid representation (the centroids are represented by grey spheres connected to the side chain).

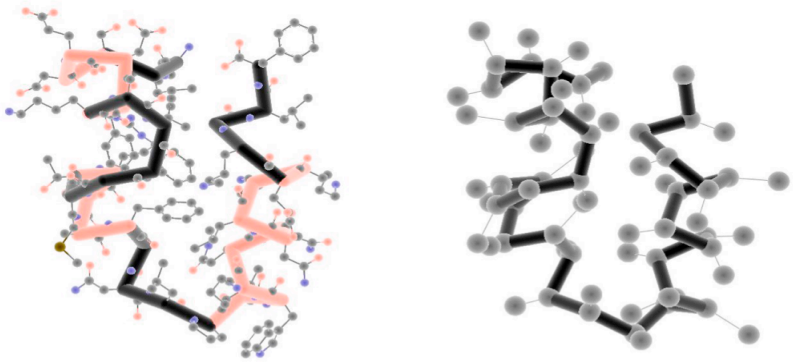

Figure 3.2: Full atoms and centroids representation.

3.1.3 Structures, intuitions and block models

In this section we introduce the formalisms adopted to capture homologies and affinities among proteins structures. These information will be used to guide the search across the space of the possible conformations.

3.1.3.1 Fragments

Consider a sequence of amino acid $S = a_1, \ldots, a_n$; we introduce the following defintion.

Definition 3.1.3 (Fragment)**.** A *fragment* is a sequence of contiguous points in the 3D space: $f = p_1, \ldots, p_L$ $(p_i \in \mathbb{R}^3)$. The p_i are associated to the i^{th} amino acids of S and describe their position P_i^α in the cartesian space.

According to homology observation [MSM00], each subsequence of S can be modeled through a specific well defined 3D shape. A fragment, in this respect, represent a rigid spatial conformation for such subsequences. Each subsequence of S can hence be associated to a given number of fragments modeling the possible 3D candidate shapes. The idea is driven from the observation that, during the exploration of the space of the possible protein 3D structures, many random conformations are generated. Using homologies we can reduce the number of combination to explore, focusing only on the biologically meaningful ones. In this respect highly conserved structures (e.g. α-helices, or β-sheets), can be used to describe specific subsequences of the target proteins—namely the ones that present high homology affinity with such well defined structures.

Let denote with $\mathscr{F}$ the *fragment space*, as the set of all fragments that could be compatibly used to model every subsequence of a given target protein. We design two fragments sets:

$\mathscr{F}_{std} \subseteq \mathscr{F}$ is the set of the fragments imported from the *Assembly Data Base* in the clustering phase (see 3.1.5). A fragment $f \in \mathscr{F}_{std}$ is referred as a *standard fragment*.

$\mathscr{F}_{spc} \subseteq \mathscr{F}$ is the set of user specified fragments, and contains the ad-hoc designed structures to model a specific area of a target protein. A fragment $f \in \mathscr{F}_{spc}$ is referred as a *special fragment*.

The set $\mathscr{F}_{std}$ is automatically generated by the analysis of large protein structure data sets. It constitutes the basic structural unit adopted to model a given short protein subsequence during the structure prediction. The set $\mathscr{F}_{spc}$ is instead build out of a semi-automatic process and it is aimed at targeting longer subsequences like α-helix and β-sheets, or other *structural blocks*— contiguous subsequences— like motifs or blocks of super secondary structures. The special fragments are created through a GUI (graphic user interface) that allow the user to visualize the protein structures from which to extract the *structural blocks* to be associated to the target sequence. We plan to integrate secondary structure prediction features (like PSIPRED, I-TASSER or PEP-FOLD) to automatize the special fragments selection process by suggesting the splitting sites for possible blocks candidates.

3.1.3.2 Pairs

Special fragments are targeted in modeling those parts of the protein that are highly conserved within other homologous polymers with known structure. These conformations are likely to form strong local interaction between each other according to the nature of their components, and in last analysis to their specific shape. Therefore, a natural assumption is that special fragments will interact among each other forming core parts in the final prediction.

We have observed that, in the exploration of the conformational search space, many "blind moves" are tried before the special fragments can be placed with a good relative spatial orientation. This is caused by the high degree of freedom in the parts of the proteins modeled by standard fragments. As an example, consider two helices connected by a short loop as shown in Figure 3.3; a typical modeling consists of the two helices as special fragments $f_i, f_j \in \mathscr{F}_{spc}$, and the loop as a sequence of standard fragments. In this example after placing the first helix (f_i) every feasible choice for the loop modeling connecting the second helix (f_j) is tried, but only few combinations will contribute in enhancing the global structure stability — the one that minimize the entropy energy level. To gain computational speed, we present a formalism that allow us to pre-compute the "good" orientations for pairs of special fragments. This suggestion mechanism is used as heuristic to guide the search in selecting the possible candidate prediction

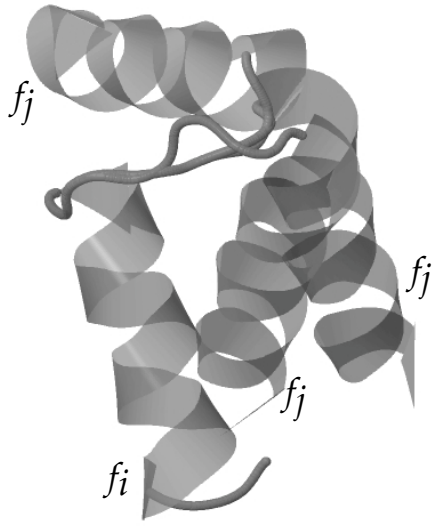

Figure 3.3: Loop flexibility in the structure prediction.

in the protein conformations search space.

Definition 3.1.4 (Pair)**.** A *pair* $\rho \subseteq \mathscr{F}_{spc} \times \mathscr{F}_{spc}$ is a binary relation between two special fragments. Given $f_i, f_j \in \mathscr{F}_{spc}$ and $i \neq j$, we denote with $\rho(f_i, f_j)$ the pair relation between f_i and f_j.

The pair relation describes the relative spatial positions between two fragments. Each pair is described by a rotation matrix $R \in \mathbb{M}(3,3)$, and a translation vector $\vec{v} \in \mathbb{R}^3$ associated to the relative positions of the first fragment of the pair. R and $\vec{v}$ describe the affine transformations, subjected to the second fragment of the pair, necessary to couple the two fragments.

More precisely, given $f_i, f_j \in \mathscr{F}_{spc}(i \neq j)$, f_i and f_j are in a pair relation $\rho(f_i, f_j)$ if there exists a rotation matrix R and a translation vector $\vec{v}$ such that the evaluation

of the torsional and orientation contributions of the entropy energy model (see section 3.1.6 for details) of f_i and $R \times (f_j \cdot \vec{v})$ exceeds a threshold $\Theta_{pair} \in \mathbb{R}$.

Pair Properties. The following properties hold for pair relation ρ over the set of special fragments $\mathscr{F}_{spc}$:

- ρ **is anti-reflexive**,

$$\forall f_i \in \mathscr{F}_{spc}, \;\; \neg\rho(f_i, f_i)$$

  A special fragment, can not be in a pair relation with itself (it trivially follows from the pair definition, since $i \neq j$).

- ρ **is symmetric**,

$$\forall f_i, f_j \in \mathscr{F}_{spc}, \;\; \rho(f_i, f_j) \implies \rho(f_j, f_i)$$

  Trivially, if there exists a rotation matrix R and a translation vector $\vec{v}$ to couple f_j to f_i, we can also define a pair relation between f_j and f_i, using R^{-1} and $-\vec{v}$.

- ρ **is anti-transitive**, Let f_a, f_b, f_c be fragments of $\mathscr{F}_{spc}$:

$$\rho(f_a, f_c) \wedge \rho(f_b, f_c) \implies \neg\rho(f_a, f_b) \wedge \neg\rho(f_b, f_a) \tag{1}$$

$$\rho(f_a, f_c) \wedge \rho(f_a, f_b) \implies \neg\rho(f_c, f_b) \wedge \neg\rho(f_b, f_c) \tag{2}$$

A graphical representation for the equations (1) and (2) is given in Figure 3.1.3.2.

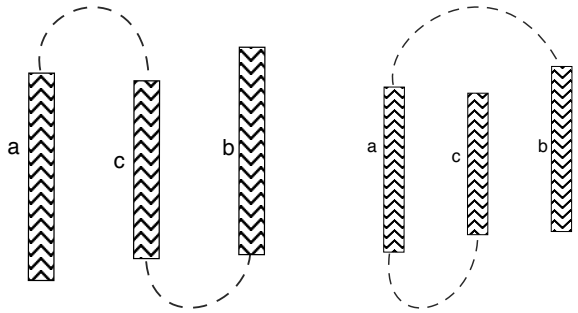

Figure 3.4: Properties (1) (left) and (2) (right).

The generalization of the anti-transitive properties defined by the (1) and (2) is given introducing the following concept. Let R be a generic binary relation. Let the *transitive closure* of R on a set of elements $a, b_1, \ldots, b_n$ $(n \geq 2)$ be defined as follows:

$$aRb_1 \wedge b_1Rb_2 \wedge \ldots \wedge b_{n-1}Rb_n \implies aRb_n \tag{3}$$

By the definition above, we define the *anti-transitive closure* of a relation, in the intuitive way. Given a set of fragments $\{f_a, f_{b_1}, \ldots, f_{b_n}\} \subseteq \mathscr{F}_{spc}$, the *anti-transitive closure* of the pair relation ρ holds for any $n \geq 2$:

$$\rho(f_a, f_{b_1}) \wedge \rho(f_{b_1}, f_{b_2}) \wedge \ldots \wedge \rho(f_{b_{n-1}}, f_{b_n}) \implies \neg\rho(f_a, f_{b_n}) \wedge \neg\rho(f_{b_n}, f_a) \tag{4}$$

The (4) is illustrated by Figure 3.1.3.2. If a pair relation would exists between fragments f_a and f_{b_n}, it would invalidate the definition of amino acid chain. Indeed a chain of residues has start and end amino acids not connected to each other. Moreover, every non trivial substring of the residue chain has a different start and end point position. Another way to see the anti-transitive closure of ρ is by the mean of a graph G where the vertices represent the fragments and the

edges represent the pair relations between two fragments. In this respect the anti-transitive closure of ρ states that G is acyclic.

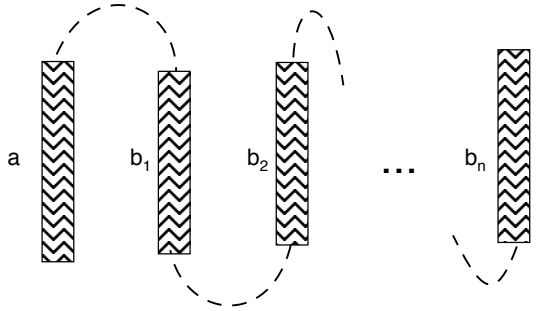

Figure 3.5: The anti-transitive closure for the pair relation ρ.

3.1.4 Fragment Assembly

In this section we describe the motivations and the intuitions behind the Fragment Assembly model adopted in this work.

The gap between the number of protein sequences and predicted 3D structures is rapidly increasing, enhanced by the great number of genomes decoded in recent years (including the human genome [Vea01]). *Comparative models* for three-dimensional protein structures, are widely considered suitable to fill this gap. These models, are based on the hypothesis that the *fragments* found to model the protein sequence of interest (target sequence) could be related by homology (common ancestor). The benefit of this approach is in that, sections confidently inherited from the fragments extracted have intrinsically good geometry, and can be used to model particular portions of the target protein.

The model exploited for the Fragment Assembly model provides a simplification

of the protein conformational space exploration. The degrees of freedom for the positions of every amino acid of a protein are almost illimitate, in a $\mathbb{R}$-continuous model as the one described above; it follows that the space of the possible candidate solution for a protein conformation is remarkably enormous. Our approach uses the *Fragment Assembly (FA)* technique [DDFP10, DDFP11] to restrict the number of possible placements of an single amino acid. Such placements are based on the chain of fragments assignments done to build the protein structure up to the amino acid of interest.

The key idea relies on assembling protein parts from the *standard* or *special* fragments set to construct the target protein structure conformation. Let $f_i = p_i, \ldots, p_{i+L_i}$, and $f_j = p_j, \ldots, p_{j+L_j}$ ($p_i \in \mathbb{R}^3$) (either of type standard and special), we introduce the following concept.

Definition 3.1.5 (Compatible Fragments)**.** Two fragments f_i, f_j are compatible if:

$$\mathsf{RMSD}\big(\langle p_{i+L_i-2}, p_{i+L_i-1}, p_{i+L_i}\rangle, \langle p_j, p_{j+1}, p_{j+2}\rangle\big) < \theta \tag{5}$$

for some $\theta \in \mathbb{R}$.

In other words, f_i and f_j are compatible fragments if the last three amino acid of f_i and the first three amino acid of f_j have a similar bend angle.

In the fragment assembly procedure, two compatible fragments f_i, f_j with $j = L_i - 3$, are assembled by superimposing the first three points of f_j to the

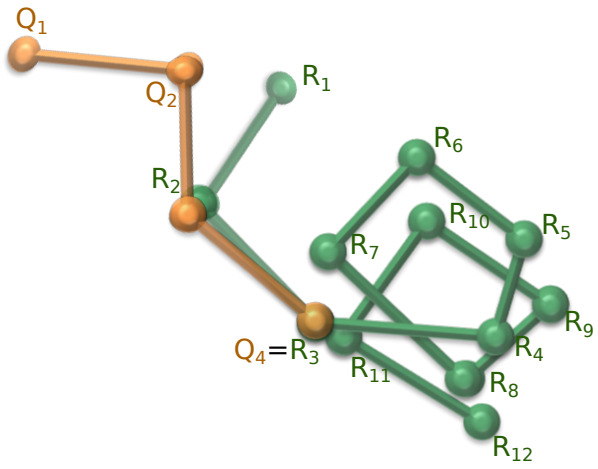

Figure 3.6: Assembly of two consecutive fragments

last three points of f_i, so that p_{L_i-2} matches the position of p_j and the rmsd measure between the common points is minimized. Figure 3.6 shows an example of fragment assembly step where $L_i = 4$ and $L_j = 12$.

3.1.5 Clustering

The *Protein Data Bank* (PDB) is a large repository for the 3D protein structures, which are typically obtained by X-ray crystallography or NMR spectroscopy. The analysis of the PDB plays an important role in this work: it provides a data set of fragments representatives to be used as general components when building a protein by fragment assembly. The *Assembly Data Base* (Assembly-db) is generated by searching for recurrent conformations among a non redundant set of proteins of the PDB. Two important considerations aries when building the

Fragment Assembly-db:

Redundancies. The PDB contains more then $70k$ protein structures, including many similar proteins deposited in several variants.

Impracticable search space. Consider a peptide sequence of length $k > 0$, $[s_1, \ldots, s_k]$, with $s_i \in \mathbb{A}$. Note that the number of different combinations such a sequence may have is $|\mathbb{A}|^k$ that is 20^k, a truly remarkable value as k increases.

In order to solve the redundancy issue, the program `tuple-generator`[DDFP11] devises a library of structures, for fragments of short length, from a subset of the PDB (called *top-500*). *top-500* is a set constituted by the 500 proteins structures, crystalized at a resolution of at least 1.8 Å and containing the largely frequent fragments among the whole PDB [LDA+00]. In this work we adopt fragments of length 4. The choice is motivated by the statistically relevant number of fragment representative that can be generated in order to ensure a good coverage for each permutation of four amino acids. This property makes it possible to engender a general data base suitable to model different classes of proteins. At the same time sequences of four amino acid are long enough to embed important structural information related to the nature of their components.

For the second issue, note that, when considering $k = 4$, the number of different 4-tuples of amino acids, is $20^4 = 160,000$. Since we only generated

roughly $60k$ fragments representatives, this means that the largest part of the 4-tuple set would remain uncovered. To overcome this problem, we introduce a partition for the amino acid set $\mathbb{A}$,

$$\mathbb{A} = \{\mathcal{A}^1, \ldots, \mathcal{A}^m\} \text{ where } \mathcal{A}^i \neq \emptyset \wedge \mathcal{A}^i \cap \mathcal{A}^j = \emptyset, \qquad \forall \mathcal{A}^i, \mathcal{A}^j \in \mathbb{A}.$$

The $\mathcal{A}^i$ are called *amino acid clustering classes*, and are determined according to the similarity of the torsional angles of amino acid bonds [FPD+07].

The number of amino acid clustering classes chosen to partition the amino acid set $\mathbb{A}$ is $m = 9$. Figure 3.7 shows the amino acid clustering classes used in this work. We define the mapping from the amino acid set to the classes $\mathcal{A}^i$, trough the function $\lambda : \mathbb{A} \to \{0, \ldots, 8\}$. Similarly let denote with $\lambda^{-1}(i) = \{a \in \mathbb{A} \mid \lambda(a) = i\}$. Considering fragments of length 4, the majority of the $9^4 = 6561$ elements has a representative in the Fragment Assembly DB. We stress that the elements of a given class are treated in the same manner during the search of a possible conformation.

We introduce the following definitions:

Definition 3.1.6 (Equivalence)**.** Let f_1, f_2 be two fragments of a primary sequence S. f_1 and f_2 are said equivalent ($f_1 \sim f_2$) iff:

i. $|f_1| = |f_2|$

ii. f_1, f_2 have same amino acid clustering pattern:

$$\forall s_i^1 \in f_1, \forall s_i^2 \in f_2, \exists c \text{ such that } \quad s_i^1 \in \mathcal{A}^c \wedge s_i^2 \in \mathcal{A}^c$$

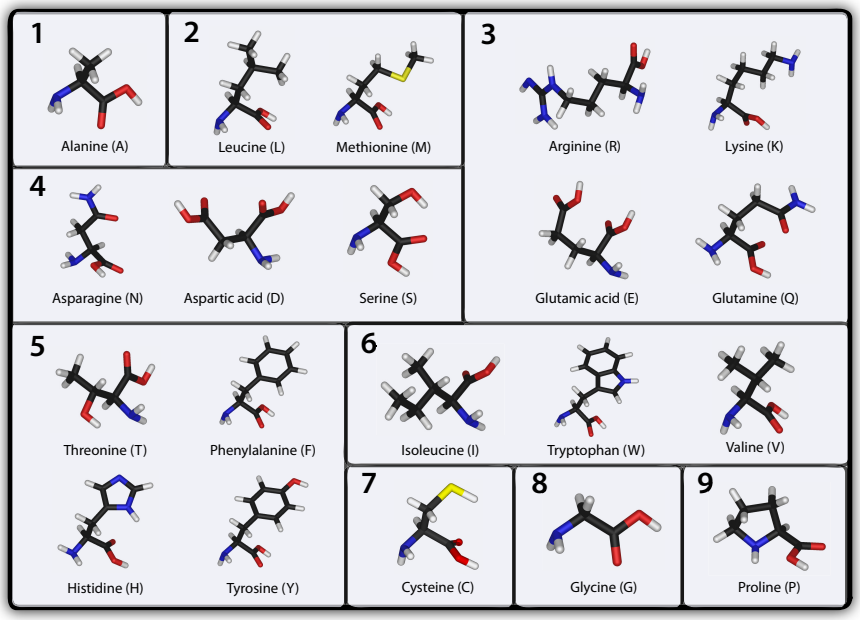

Figure 3.7: The amino acid clustering partition.

Theorem 3.1.1. *The binary relation "$\sim$" between two fragments, is an equivalent relation over the fragment space $\mathscr{F}$.*

Proof. Let f_1, f_2, f_3 fragments of generic sequences $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$. The relation "$\sim$" is trivially refexive ($f_1 \sim f_1$).

Assume, by hypothesis $f_1 \sim f_2$; follows:

$$|f_2| = |f_1| \ \wedge \ (\forall s_i^1 \in f_1,\, s_i^2 \in f_2)\ (\exists c)\ :\ (s_i^1 \in \mathcal{A}^c \ \wedge \ s_i^2 \in \mathcal{A}^c) \qquad \text{(from i and ii)}$$

that is the symmetric property for "$\sim$".

Let $f_1 \sim f_2$ and $f_2 \sim f_3$, that is:

$$|f_1| = |f_2| \ \wedge \ (\forall s_i^1 \in f_1, s_i^2 \in f_2)\ (\exists c_1)\ :\ (s_i^1 \in \mathcal{A}^{c_1} \ \wedge \ s_i^2 \in \mathcal{A}^{c_1}) \qquad \text{(a)}$$

$$|f_2| = |f_3| \ \wedge \ (\forall s_i^2 \in f_2, s_i^3 \in f_3)\ (\exists c_2)\ :\ (s_i^2 \in \mathcal{A}^{c_2} \ \wedge \ s_i^3 \in \mathcal{A}^{c_2}) \qquad \text{(b)}$$

Follows, for transitivity of the equivalence relation:

$$|f_1| = |f_3|, \tag{*}$$

more over, for all $s_i^1 \in f_1, s_i^2 \in f_2, s_i^3 \in f_3$ there exist c_1, c_2 such that:

$$s_i^1 \in \mathcal{A}^{c_1} \ \wedge \ s_i^2 \in \mathcal{A}^{c_1} \ \wedge \ s_i^2 \in \mathcal{A}^{c_2} \ \wedge \ s_i^3 \in \mathcal{A}^{c_3}. \qquad \text{(from (a) and (b))}$$

Since the amino acid clustering classes are a partition for $\mathbb{A}$, follows: $c_1 = c_2$ (from $s_i^2 \in \mathcal{A}^{c_1}$ and $s_i^2 \in \mathcal{A}^{c_2}$). Let denote with c such a index:

$$(\forall s_i^1 \in f_1, s_i^2 \in f_2)\ (\exists c)\ :\ (s_i^1 \in \mathcal{A}^c \ \wedge \ s_i^3 \in \mathcal{A}^c) \tag{**}$$

Combining (*) with (**) follows $f_1 \sim f_3$; that is the transitivity of "$\sim$". Hence the claim. □

A subset of $\mathscr{F}$ containing all and only the elements equivalent to some $f \in \mathscr{F}$ defines an equivalence class of f under "$\sim$" ($[f]_\sim$), $[f]_\sim := \{x \in \mathscr{S} \mid x \sim f\}$. The equivalence relation "$\sim$" defines a partition over $\mathscr{F}$. We denote $\mathscr{F}/\sim$ the collection of the equivalence classes over $\mathscr{F}$, (that is the quotient set of $\mathscr{F}$ by "$\sim$"): $\mathscr{S}/\sim \ := \{[f] \mid f \in \mathscr{S}\}$.

Let now introduce the concept of *root mean square deviation*. Let $\vec{v} = (v_1, \ldots, v_n)$, $\vec{u} = (u_1, \ldots, u_n)$ two vectors, where $v_i, u_i \in \mathbb{R}^3$. The $\mathsf{rmsd}(\vec{v}, \vec{u})$ is defined by the following:

$$\mathsf{rmsd}(\vec{v}, \vec{u}) = \sqrt{\frac{\sum_{i=1}^{n} \left((v_{i,x} - u_{i,x})^2 + (v_{i,y} - u_{i,y})^2 + (v_{i,z} - u_{i,z})^2 \right)}{n}} \tag{6}$$

Definition 3.1.7 (RMSD)**.** Let $\vec{v}, \vec{u}$ two vectors, with $v_i, u_i \in \mathbb{R}^3$. We introduce the function RMSD to measure the similarity of two vectors $\vec{v}, \vec{u}$:

$$\mathsf{RMSD}(\vec{v}, \vec{u}) = \min_{\phi,\theta,\psi \in \mathbb{R}} (\mathsf{rmsd}(\vec{v},\, rot(\phi, \theta, \psi)(\hat{u}))) \tag{7}$$

where $rot(\phi, \theta, \psi)$ is the function that rotate a vector in the 3D space along the components ϕ, θ, ψ; $\hat{u}$ is the translation $\hat{u} = \vec{u} + \vec{l}$, where $\vec{l}$ is defined by the components:

$$l_x = \frac{\sum_i v_{i,x} - u_{i,x}}{n}, \quad l_y = \frac{\sum_i v_{i,y} - u_{i,y}}{n}, \quad l_z = \frac{\sum_i v_{i,z} - u_{,z}}{n}$$

In other words the points of $\vec{u}$ are rotated and translated so to minimize the measure of the deviation. The (7) gives a measure of similarity for two fragments of equal length. The more its value is close to 0, the better the fragments points relate to each other.

To reduce even further the number of fragments for a given class we group similar fragments representing them by a unique candidate.

Definition 3.1.8. (Similarity) Two fragments f_1, f_2 are said similar if they are equivalent and $\mathsf{RMSD}(f_1, f_2) \leq \epsilon$, for some $\epsilon \in \mathbb{R}$. We denote it with $f_1 \simeq f_2$.

Given a set of fragments $F = \{f_1, \ldots, f_m\}$, a fragment f_k is *representative* of F if: $f_k \in F \wedge \forall f_i \in F \quad (f_k \simeq f_i)$. For the definition of similarity adopted in this work, we refer the reader to Section 4.5.1 where we discuss the methods used for the system evaluation.

The format of the data base generated by the program `tuple-db` is the following:

$$[c_1, c_2, c_3, c_4 \quad X_1^\alpha, Y_1^\alpha, Z_1^\alpha, X_2^\alpha, Y_2^\alpha, Z_2^\alpha, X_3^\alpha, Y_3^\alpha, Z_3^\alpha, X_4^\alpha, Y_4^\alpha, Z_4^\alpha, \ FREQ, \ ID, Pid]$$

where c_1, c_2, c_3, c_4 defines an *amino acid clustering pattern*, with $c_i \in \{0, \ldots, 8\}$, $X_1^\alpha, \ldots, Z_4^\alpha$ are the coordinates of the 4 $C\alpha$ atoms of the fragment, $FREQ \in [0, 1] \subset \mathbb{R}$ describe the frequency of occurrence of a given pattern in the *top-500* set, ID is a unique identifier for the current DB element, and Pid is the first protein found containing such fragment.

The Assembly Db is given as input for our solver which will extract the subset of fragments to be used for the prediction of a given target sequence.

Given a target primary sequence $S = a_1 \ldots, a_n$, it is unlikely that a subsequence of L amino acids will not appear as an element of $\mathscr{F}$; nevertheless, to handle this possibility, if $\gamma(a_i), \ldots, \gamma(a_{i+L-1})$ has no statistics associated to it, we map it to a "general fragment" with pattern $[-1, -1, \ldots, -1]$. This class contains the most common fragments of the Assembly-DB. We also introduce other two general patterns:

- $[-2, -2, \ldots, -2]$ that contains the fragment describing a generic subsequence of α-helices.

- $[-3, -3, \ldots, -3]$ that contains the fragment describing a generic subsequence of β-sheets.

These patterns are applied to those region of the sequence where a secondary structure constraint is, possibly, enforced.

3.1.6 Energy Model

A candidate protein conformation is evaluated by an energy function that estimates its entropy measure. In general these energies are aimed at modeling force fields representing atomic interactions. Examples of functions empirically tested (either statistically or through small quantum mechanic observations) are AMBER [JTR98], CHARMM [BBO+83], ECEPP [FAMCMS74], MM3 [AYL89].

The energy function used in this work relies on an interaction model which entails energy terms for the following components:

1. A *contact energy* component, to represent force fields involving backbone components ($C\alpha$'s atoms) and side chains (centroids). Moreover, we consider *pseudo bonds* between two consecutive $C\alpha$'s and between a $C\alpha$ and the corresponding centroid (except for glycine which entails only one center of interaction for the backbone).

2. A *torsional energy* component, defined by four consecutive pseudo-bonded centers of interaction ($C\alpha$ atoms). This energy term is defined by the potential of the mean force derived by the distribution of the corresponding torsional angle in the PDB [FPD+07]. This terms are aimed at describing local conformations (e.g. derived from secondary structures).

3. An *orientation energy* component, defined for every three pseudo-bonded centers of interaction. This component is described by a *pseudo-torsional* energy term, used to maintain proper chirality of side chain orientation with respect to the main chain.

Contact Energy Contribution. For this component we use the contact energy table described in [BMF03]. This table defines the quantitative contact interactions between pairs of amino acids, and it is determined by statistical observation from contact interactions among a set of proteins with known native structure. A *contact*, in this context, is defined as the interaction between centroids and $C\alpha$'s, within their Van der Waals radius.

The energy assigned to the contact between backbone atoms (represented by the $C\alpha$'s) is the same energy assigned to ASN-ASN, which involve mainly similar chemical groups contacts. We employ a *cutoff* (distance within which the energetic forces are maximized) of 4.8Å between backbone contacts ($C\alpha - C\alpha$). For centroids the cutoff is given by the summation of their radius. For distances greater then the cutoff value, the contributions decay quadratically (w.r.t the distance itself).

We report the formal definition of the contact energies for $C\alpha - C\alpha$'s and centroid–centroid interactions. Let $P_i^{ca}, P_j^{ca} \in \mathbb{R}^3$ be the points describing the coordinates for $C\alpha$'s ca_i, ca_j, and let $P_i^{cg}, P_j^{cg} \in \mathbb{R}^3$ be the points describing the

coordinates for the centroids cg_i, cg_j, and r_i, r_j be the radius of these centroids, the contact energies are defined as follows:

$$\texttt{ENcont\_ca}(i,j) = \Delta_{ca}(P_i^{ca}, P_j^{ca}) \cdot M_{cont}[i,j]. \tag{8}$$

$$\texttt{ENcont\_cg}(i,j) = \Delta_{cg}(P_i^{cg}, r_i, P_j^{cg}, r_j) \cdot M_{cont}[i,j]. \tag{9}$$

where M_{cont} is the contact matrix as defined in [BMF03] and Δ_{ca}, Δ_{cg} are the decay functions defined according to the cutoff distances.

Torsional Energy Contribution. Torsional contributions are defined by four consecutive centers of interaction ($C\alpha$ atoms). The energetic terms involved in these contribution are calibrated by considering two center of interaction for each amino acid: the $C\alpha$ atom, and the centroid. In last analysis, the torsional contribution capture the covalent bonds formed by each $C\alpha$ atom within its own center of mass and other two adjacent $C\alpha$ atoms. The parameters are devised from statistical analysis in the *Top-500* protein set.

Torsional components express a characterization of the propensity of a given subsequence to generate local structures. This property is described by the dihedral angle defined among four consecutive center of interaction of the backbone: $\texttt{tors}(\phi(C\alpha_i, C\alpha_{i+1}, C\alpha_{i+2}, C\alpha_{i+3}))$. Moreover, we introduce a term to describe relations between adjacent local conformations. This term is defined by

the dihedral angle of two adjacent 4-tuple of consecutive $C\alpha$'s:

$$\texttt{corr}(\phi(C\alpha_i, C\alpha_{i+1}, C\alpha_{i+2}, C\alpha_{i+3}), \phi(C\alpha_{i+1}, C\alpha_{i+2}, C\alpha_{i+3}, C\alpha_{i+4})).$$

The formal description of the torsional energy contribution is given in the equation (10). We consider two energy tables: M_{tors}, a tridimensional matrix describing the torsional contribution, provided two amino acids and the discrete value of the torsional angle defined over four consecutive $C\alpha$'s; and M_{corr}, a two dimensional matrix describing the correlation contribution. Let a_i be the i^{th} amino acid of a protein sequence, and P_i be the coordinates for the $C\alpha$ associated to the amino acid a_i, the torsional contribution for an amino acid a_i us given by the:

$$\texttt{ENtors}(i) = M_{tors}[a_i, a_{i+1}, \phi_t(i)] + M_{corr}[\phi_t(i), \phi_t(i+1)]], \tag{10}$$

where $\phi_t(i) = \frac{178 - \phi(P_i, \ldots, P_{i+3})}{5}$.

Orientation Energy Contribution. The orientation contributions describe the symmetries of the geometric distributions among every three consecutive $C\alpha$'s. These components are used to maintain the proper chirality of side chains orientation w.r.t. the backbone. The main idea here is to consider the directionality of the hydrogen bonds between pairs of amino acids using samples obtained from an analysis of the PDB. More specifically, following an approach similar to the one presented in [HTS+04], we consider triple of adjacent $C\alpha$'s to be compatible if the radius of the circumference tangent to the three $C\alpha$'s is within $[2.5, 7.9]$Å.

In this model we also take account of the hydrophobic effects of a solvent, by applying a force field to those atoms that are separated by a distance smaller than 7.9Å. The characterization of an hydrogen bond is given by the geometrical properties of the local planes defined over two pairs of three consecutive $C\alpha$'s. The intuition here, is to gather information about the presence of an hydrogen bond by focusing on the vectors normal to these planes and intersecting the points defined by the two $C\alpha$ considered, together with the distance vector from such atoms. These properties are hence compared and evaluated w.r.t. the ones derived from a statistical analysis of such local system in the PDB. For an exhaustive treatment of the argument see [HTS+04].

The description of the contact contributions is given in the equation (11)

$$\texttt{ENori}(i,j) = ori_{aux}\big((P_i - P_j), (P_i - P_{i-1}), (P + i + 1 - P_i), (P_j - P_{j-1}), (P_{j+1}) - P_j)\big), \tag{11}$$

where ori_{aux} describe the geometrical characterizations discussed above.

More details on the energy components can be found in [BMF03].

3.2 A brief analysis on complexity

In this Section we consider the computational complexity of the protein structure prediction problem.

One of the most well studied protein representation model is the Dill's HP-lattice model [Dil85]. According to the observation of that hydrophobic amino

acids tend to pack together avoiding the watery environment, the HP-model classifies each amino acid in accordance to its propensity to escape or tolerate water. Under the simple HP-model, in a 2D lattice for protein representation, the PSP problem has been shown to be NP-complete [CGP+98]. Such problem can be formulated as the problem of finding the hamiltonian path that maximizes the number of H's pairs within a unit lattice distance; the NP-completeness proof for the PSP problem under the HP-model for lattice relies on a formalism that maps nodes of a graph onto an hyper-cube.

In the model presented in this work, a solution to the PSP is a mapping from the sequence of amino acids $S = a_1, \ldots, a_n$, to the sequence of points $p_1, \ldots, p_n$ in $\mathbb{R}^3$, satisfying the imposed constraints. Proving the complexity of the PSP under such model, shall take into account that the space of the admissible solutions is restricted to the ones satisfying the CSP.

In doing so, we consider a simplification of the space modeling the proteins, i.e. 2D or 3D lattice models. In order to undertake the complexity discussion of the PSP we need to introduce the notions of two global constraints: the `alldifferent` and the `contiguous` constraint. Consider a set of n variables $x_1, x_2, \ldots, x_n$, with associated domains $D_1, D_2, \ldots, D_n$. Then,

$$\texttt{alldifferent}(x_1, \ldots, x_n) = \{\langle a_1, \ldots, a_n\rangle \mid \langle a_1, \ldots, a_n\rangle \in D_1 \times \ldots \times D_n \ \wedge \ \forall i, j \quad (1 \leq i < j) \Rightarrow (a_i \neq a_j)\}$$

and,

$$\texttt{contiguous}(x_1, \ldots, x_n) = \{\langle a_1, \ldots, a_n\rangle \mid \langle a_1, \ldots, a_n\rangle \in D_1 \times \ldots \times D_n \ \wedge \ \forall i \in \mathbb{N}, \quad (1 \leq i < n) \Rightarrow (a_i, a_{i+1} \in E)\}$$

where E is the set of lattice edges. The `alldifferent` constraint, states that every point in the lattice has a unique position, and it cannon be used to model two distinct amino acids; the *contiguous* constraint states that contiguous amino acids in a primary sequence are mapped as contiguous points in the lattice.

Note that, the minimal requirements, in terms of constraints to be satisfied, in order to generate a biologically meaningful conformation, consists of the global constraint $\texttt{alldifferent} \cap \texttt{contiguous}$. Such property is described as *Self Avoiding Walk* (`saw`), that is, a lattice path that does not visit the same point more then once.

In [DDP06] Dal Palú and co-workers proved that the satisfiability of the `saw` constraint is an NP-complete problem. Moreover it follows that the problem of determining wether such constraint is generalized arc consistent, is in NP-complete as well[1]. The proof is a reduction from Hamiltonian Circuits, and relies on the use of special planar graphs [DDP06].

The results for the `saw` constraint consistency verification problem for lattice points, has been extended to fragments whose point are represented by given

[1]In [DDP06] such property is proven to be NP-hard.

coordinates in $\mathbb{Z}^2$ trough the constraint `scfsaw` [Cam11]. Intuitively a solution for the `scfsaw` constraint is a sequence of fragments, modeling adjacent amino acids, for which a geometric superimposition is feasible (the latter is expressed by the `compatibility` constraint—see Section 4.2.3.3).

Verifying the consistency of the `scfsaw` constraint over 2D lattices is NP-complete, and as corollary determining wether the generalized arc consistency for the `scfsaw` constraint over 2D lattices is satisfied, is an NP-complete problem [Cam11].

However, even if the results established proved the intractability of the problem for general inputs, it is important to stress that we are not interested in studying unbounded instances of the PSP problem. Tackling proteins for lengths of $n = 200 - 300$ would be considered a remarkable contribution for biologists and researcher interested in the analysis of protein behaviors and structure.

3.3 Related works

De novo protein structures prediction is one of the most important challenges in computational biology—many progresses have been done in recent years, although a definitive solution to the problem remains elusive. Research in this area can be divided into *fragment assembly* [SKHB97, KKD+03, LKJ+04] and *molecular dynamics*. The former attempt to assign a fragment with known structure to a portion of unknown target sequence. The latter uses simulations of physical movements of atoms and molecules that can interact to each other, to

simulate the folding process.

Molecular dynamics methods require a full detailed representation of the protein model. Being able to explicitly represent a high number of degree of freedom has the advantage of creating accurate results, however, due to the cost of treating the electronic degrees of freedom, explicit solvent molecular dynamics simulations of protein folding are still beyond current computational potential, limiting its domain of application to small proteins, or protein particles.

The other class of ab-initio method, arising in the recent years, uses approximated representation for proteins and force fields (*reduced models*), evolutionary information from multiple alignments, and fragment assembly techniques. These methods rely on assembling a protein structure using fragments (obtained from structural databases) that present homologous affinity with the target sequence. The simplified models adopted for protein and space representation, allow to compute the problem more efficiently. The solutions generated in the reduced models can hence be refined with more computationally expensive methods (e.g. molecular dynamic simulations). The motivations on the use of fragment assembly rely on the intuition that it may be possible to reconstruct a complete protein structure from a set of possible compatible substructures—selected from proteins whose conformations are known.

A state-of-the-art predictor is represented by *Rosetta*, developed by Baker et al. [Rea09]. This approach has been showed to be successful, during the re-

cent editions of the CASP, performing as the best ab-initio predictor[RSMB04]. *Rosetta*'s approach is based on a simulated annealing search to compose a conformation, and a fragment assembly technique to combine substructures extracted from a PDB-based fragment library.

The approach used in this work is based on the use of the constraint programming paradigm. The general idea relies on the use of information, gathered from different sources—homologies, secondary structure predictions, amino acid types, etc.—to reduce the computational complexity of the problem by removing, from the conformational space, those solutions that do not satisfy the addressed constraints, i.e. the conformations that would result to be biologically manfulness or irrelevant in entropic terms.

Backofen and Will have made use of constraints over a simple lattice model in the Dill's HP-lattice model [LD89, LD90, Bac98a, BW02, BW03, BW01a, Bac98b, BWBB99, Bac04, BW01b]. In their work they show high accuracy and efficiency in predicting proteins up to 160 amino acid. The process relies on capturing information about the possible shapes of the protein region containing all H-monomers (defined as *core shapes*) to constrain the search space[Bac98b].

Krippahl and Barahona proposed a constraint-based approach to determining protein structures compatible with distance constraints obtained from Nuclear Magnetic Resonance data [KB02, KB99, KB03]. In [KB03] they provide a constraint programming approach to protein docking. In their work they prune the search

space by ensuring bounds consistency on interval constraints that model the requirement for the not overlapping shapes to be in contact.

The model adopted by the work presented in this Thesis follows the approach of *TUPLES* [DDFP11] in which Dal Palú et al. presented a declarative approach to the protein structure prediction problem, that relies on the *Fragment Assembly* model. Their solution uses a data base of small fragments used to assembly the final conformation, which is subjected to spatial and geometrical constraints. They also take account of the secondary structures present in a given conformation, using specific fragment templates according to the nature of the local structure.

A novel CP-based solver for protein structure analysis. In this work, we propose an efficient implementation encoding a constraint solver for protein structure analysis, using the fragment assembly model. We extend the solution proposed by Dal Palú et al., using an imperative programming approach (coded in C), that embeds a declarative logic. In particular we guarantee high modularity in the way the information used to describe a target protein are encoded. For example, ad-hoc distance or geometrical constraints can be easily added, or different energy model can be used as optimization functions, without the need of reshaping the model.

We adopt an equivalent protein model representation and energy function model

of those proposed in [DDFP11]. In addition we introduce a new concept of *special* fragments, that allow the final user to impose any type of geometrical constraints over a subsequence of the target protein. We present novel constraints and propagators to model interactions among amino acids and parts of the protein. Finally we design a refined concurrent version of the system to increase scalability. We show that our approach, in the sequential version, produces computational time improvements gaining up to 3 order of magnitude when compared to the work presented in [DDFP11] (see Section 4).

CHAPTER

FOUR

FIASCO: FRAGMENT-BASED INTERACTIVE ASSEMBLY FOR PROTEIN STRUCTURE PREDICTION WITH CONSTRAINTS

This chapter provides a formal description and the implementation details of an efficient constraint programming framework, targeted in solving the ab-initio *Protein Structure Prediction (PSP) problem* via fragment assembly.

The *Fragment-base Interactive Assembly for protein Structure prediction with COnstraint* (FIASCO)[BBC+11] is a general constraint solver, implemented using the Constraint Programming common formalism. We model variables and constraints aimed at capturing properties of amino acids and local shapes (see discussion in Chapter 3), and we developed ad-hoc propagators to prune the search tree by ensuring bound consistency on the constraints that model feasible protein conformations.

We show that the our system is able to produce predictions within a marginal error range for short and medium peptides for which a weak homologies information are supported. Moreover, we show that our system is in average more then 3 order of magnitudes faster then a declarative version (TUPLES [DDFP11]).

4.1 Introduction

FIASCO is a novel framework, based on constraint programming, targeted at studying the ab-initio protein structure prediction via fragment assembly, and encodes the formalisms described in the previous Chapter.

FIASCO is the product of an intense ongoing research, originally started by the early investigation of dal Palú et al. [DDP06, DDP07], who presented a CLP solver, *COLA* and its imperative version to address the PSP problem on discrete lattices and a simplified model for proteins representation. In [DDP06] a protein backbone is identified by the chain of $C\alpha$ atoms identifying an amino acid and each $C\alpha$ is modeled as a CP variable which domain ranges on lattice points. They implemented constraints, such as, self avoiding walks, distance constraints and block constraints to prune the search space. In later works dal Palú and coworkers also explored fragment assembly techniques via Constraint Logic Programming [DDFP10]. In [DDFP10] the problem of assembling fragments into a complete conformation is mapped to a constraint solving problem and solved using CLP.

In this work we present an optimized C implementation, of the CP model

studied to address to the PSP problem. Our approach uses a simplified protein representation model, based on a $C\alpha$-backbone structure and a side chain centroid model. This model offers an efficient representation of the polypeptide with a good spatial representation and volume occupancy approximation.

The prediction of the tridimensional shape of a protein by fragment assembly, is modeled as a constraint optimization problem, subjected to an energy function evaluation. The contribution of this work is twofold: on one side we offer an efficient sequential implementation of the solver and the encoding of the PSP problem in a CSP (in this Chapter we discuss the variable encoding, constraints and propagators); on the other hand we investigate the parallelization of the model, implementing concurrent solvers operating on different areas of the search spaces. We also implement a distributed job balancing strategy to overcome at the irregularity of the search space (the parallelization of FIASCO will be discussed in Chapter 5).

We show that the encoding of the PSP problem using fragment assembly in a CP based framework, is suitable to solve small and medium protein structures (< 100 amino acids) when partial knowledge about target protein homologies (or analogies) is given.

4.2 Problem modeling

In this Section we describe the constraint framework addressed to the general problem of the Protein Structure Prediction, which is encoded as a *constraint satisfaction problem* (CSP). This modeling allow us to encode the PSP problem using the classical formalisms common in the constraint programming paradigm. The problem is encoded by defining variables over real domains and constraints over them, and the conformations are generated by searching the space of admissible solutions.

4.2.1 Domain Manipulation

Let us introduce the domain definition and their manipulation operation as used in FIASCO.

Definition 4.2.1 (Domain)**.** A domain D is described by a pair $\vec{L}, \vec{U} \in \mathbb{R}^3$, where $\vec{L} = (L_x, L_y, L_z)$ and $\vec{U} = (U_x, U_y, U_z)$.

The above domain encoding represent the minimal information needed to describe the set of points contained in the cube having the left-lower corner at position $\vec{L}$ and the right-upper corner at $\vec{U}$, defined as $Box(D)$ in [Dal06].

The simplicity of this representation pays in the efficiency in handling the possible values of a variable. Moreover, using a single variable to represent a three-dimensional point is more effective in terms of consistency checking then consider each coordinate independently [KB03].

Using the *Box* terminology introduced in [DDP06] we introduce the following notation: D is *admissible* if $Box(D) \neq \emptyset$; D is *ground* if $|Box(D)| = 1$ (i.e. $\vec{L} = \vec{U}$); D is *failed* if not admissible.

Let us introduce the domain manipulation operation:

Definition 4.2.2 (Dilatation)**.** Let D be a domain and $h \in \mathbb{R}^+$, the domain dilatation $D + h$ is defined as:

$$D + h = \langle (L_x - h_x,\, L_y - h_y,\, L_z - h_z),\ (U_x + h_x,\, U_y + h_y,\, U_z + h_z) \rangle$$

The domain dilatation is used to enlarge the set of possible values for a variable by $2h$ units.

Definition 4.2.3 (Union)**.** Let D, E be two domains, the union of $D \cup E$ is defined as:

$$D \cup E = \langle \min(\vec{L}^D, \vec{L}^E), \max(\vec{U}^D, \vec{U}^E) \rangle$$

where:

$$\min(\vec{L}^D, \vec{L}^E) = \{\min(L_x^D, Ł_x^E), \min(L_y^D, L_y^E), \min(L_z^D, L_z^E)\}$$

$$\max(\vec{U}^D, \vec{U}^E) = \{\max(U_x^D, U_x^E), \max(U_y^D, U_y^E), \max(U_z^D, U_z^E\}$$

Definition 4.2.4 (Intersection)**.** Let D, E be two domains, the intersection of $D \cap E$ is defined as:

$$D \cap E = \langle \max(\vec{L}^D, \vec{L}^E), \min(\vec{U}^D, \vec{U}^E) \rangle$$

In terms of modeling, each atom is represented by a (point) variable V, which is associated to a domain $D(V) = \langle \vec{L}^{D(V)}, \vec{U}^{D(V)} \rangle$ representing the set of the possible values (coordinates) such atom can take. We will use the following notation to express the same concept: $D(V) = \langle \vec{L}^{V}, \vec{U}^{V} \rangle$.

4.2.2 Modeling: Variables and Constraints

The inputs to the modeling phase is composed of: a sequence $S = a_1, \ldots, a_n$, where a_i denotes the i^{th} amino acid of the primary structure; a set $\mathscr{F}_{std}$ of *standard fragments*, containing the fragments imported from the *Assembly DB*; and, a set $\mathscr{F}_{spc}$ of *special fragments*, containing the fragments created and manipulated through a GUI (a java-based graphic user interface for FIASCO).

The constraint model derived from these inputs makes use of three different types of constraint variables: `Point`, `Fragment` and `Pair`.

Points. The solver generates a list Pt^{α}_{LIST} (resp., Pt^{C}_{LIST}) of n variables `Point`, representing the 3D position of the $C\alpha$ atoms (resp., of the centroids). The domain of a `Point` variable is described by a pair $(\vec{L}, \vec{U})$, as discussed above.

Fragments. An additional list F_{LIST} of $n - L$ variables of type `Fragment` is generated (where L denotes the length of the fragments). The i^{th} element of F_{LIST} corresponds to the fragment to be used for the tuple $\gamma(a_i), \cdots, \gamma(a_{i+k})$ $(3 \leq k \leq n - L)$. The possible values for a variable $\mathbf{F}_i$ of F_{LIST} are the IDs of the

tuple:

$$\langle [\gamma(a_i), \cdots, \gamma(a_{i+k})], \ldots, \texttt{type}, \texttt{Freq}, \texttt{ID} \rangle$$

where `type` denotes the type of fragment (*standard* or *special*), and γ is the function that associates amino acids to the amino acid classes defined in Section 3.1.5. This domain is sorted according to the frequency `Freq` in a decreasing order, and by listing the *special* fragments first.

`Fragment` variables are correlated to `Point` variables trough compatibility constraints (discussed in sections 4.2.3.3 and 4.3.2.3).

Pairs. A pair relation between two fragments of type *special* is defined as a binary constraint. Each pair is described by a rotation matrix M, and a translation vector $\vec{v}$ associated to the relative positions of the first fragment involved in the pair relation. M and $\vec{v}$ describe the affine transformations to be applied to the second fragment of the pair, in order to couple the two fragments (see section 4.3.2.4 for a more detailed discussion).

In order to model these relations we introduce an additional list P_{LIST} of m variables ($m < n - 3$) of type `Pair`. The i^{th} element $\mathbf{P}_i$ of P_{LIST} corresponds to a pair $\langle f_a, f_b \rangle$, where $f_a \in \mathbf{F}_i, f_b \in \mathbf{F}_j$ ($i \neq j$). f_a and f_b are *special* fragments, referred, from now on, as the possible choices for the `Fragment` variables $\mathbf{F}_i$ and $\mathbf{F}_j$, respectively. The possible values for a pair $\mathbf{P}_i$ are the triples $\langle \langle f_a, f_b \rangle, M_{i,j}, \vec{v}_j \rangle$, where $M_{i,j}$ and $\vec{v}_j$ are the rotation matrix and the translation vector used to

orient the fragment f_b to the reference system of f_a that minimize the energy contributions.

4.2.3 Constraints

In this section we describe the constraints used to encode the Protein Structure Prediction problem. A discussion on constraint propagation is given in Section 4.3.2.

We model a candidate protein conformation through the use of several classes of constraints and we categorize them into *distance constraints*, *geometric constraints* and *energy constraints*.

4.2.3.1 Distance Constraints.

Distance constraints model spatial properties among points in the 3-dimensional space. We first define a formalism to encode the lower and upper bound constraints on the Euclidian distance between points of the cartesian space, and then we define the constraints used by the FIASCO solver.

Given a primary sequence $S = a_1, \ldots, a_n$ with $a_i \in \mathbb{A}$, let V_i, V_j be two variables representing the positions of points a_i, a_j of the sequence $\mathcal{S}$, $d \in \mathbb{N}$ and $P_i, P_j \in \mathbb{R}^3$ be two points of the cartesian space:

$$\delta(V_i, V_j) \leq d \iff \exists P_i \in \mathbb{R}^3, \exists P_j \in \mathbb{R}^3, \ s.t. \ ||P_j - P_i|| \leq d \tag{12}$$

$$\delta(V_i, V_j) \geq d \iff \exists P_i \in \mathbb{R}^3, \exists P_j \in \mathbb{R}^3, \ s.t. \ ||P_j - P_i|| \geq d \tag{13}$$

We will also use the constraint $\delta(V_i, V_j) = d$, to express the constraints $\delta(V_i, V_j) \leq d \ \wedge \ \delta(V_i, V_j) \geq d$.

next constraint.

$$\forall i \in \{0, \ldots, n-1\}, \ \delta(V_i, V_{i+1}) = 3.8\text{Å}.$$

Adjacent amino acids in the primary sequence (represented by their $C\alpha$) are separated by a distance of 3.8 Å. This constraint is implicitly expressed in the fragment definition; indeed fragments, being polypeptide subsequences themselves, are naturally subjected to the next property. Also the application of the fragment constraint (described below) ensure this property to be held by points modeling adjacent fragments.

alldistant constraint.

$$\forall i, j \in \{0, \ldots, n-1\}, |j-i| > 1, \ \delta(V_i, V_j) \geq \texttt{ALLDIST\_THS}_{\texttt{C}\alpha}\text{Å}.$$

Two non consecutive amino acids must be separated by a distance of at least $\texttt{ALLDIST\_THS}_{\texttt{C}\alpha}$ Å. The alldistant constraint is also applied for `Point` variables describing centroids. The value for two centroids that must be exceeded (or matched) is fixed to $\texttt{ALLDIST\_THS}_{\texttt{cg}}$ Å.

Disulfide bond constraint.

$$\forall i, j \in \{0, \ldots, n\} | j-i| > 1, (\gamma(a_i) = \text{s} \wedge \gamma(a_j) = \text{s}) \implies \delta(V_i, V_j) \leq D.$$

The presence of disulfide bridges constrained the amino acid involved to be separated by a distance of at most $D = 6\,\text{Å}$.

compact factor.

$$\forall i, j, \in \{0, \ldots, n\},\ \Delta(V_i, V_j) \leq 5.68 n^{0.38} \text{Å}.$$

A `diameter` parameter is used to bound the maximum distance between different $C\alpha$ atoms (i.e., the diameter of the protein). As argued in [FPD+07], a good diameter value is $5.68\, n^{0.38}$ Å.

4.2.3.2 Energy Constraints

Energy components values are treated as constraints and computed incrementally any time a local consistent solution is found. Allowing the energy components to be computed incrementally provides a significant speedup in the search for the putative conformations. This follows from that most of the consistent solutions found share many variable's value with other solutions. The idea here is to compute the energy components, for those common variables, only once. This constraint involves a `Fragment` variable and the relative `Point` variables associated to it.

energy constraint. The energy constraint is awaken every time a a `Fragment` variable F_i is `changed`. Let $\mathbf{F}_i$ be a fragment involved in the CSP modeling, with

$V_i, \ldots, V_j$ be variables representing points of $\mathbf{F}_i$. The energy constraint involves the following components:

- `cg-cg contribution.` The centroid–centroid interaction is computed by pairing every point involved in the `Fragment` variable F_i propagating the constraint, with every other `ground` point. More formally:

  $\forall k \in \{i+2, \ldots, j-1\}, \forall h \in \{1, \ldots, i+1\} \cup \{k+1, \ldots, j-1\} \cup \{j \ldots, n-1\}$

  $$\text{EN\_CONT\_CG} = \sum_{k,h} \mathtt{g}(a_{h-1})\mathtt{g}(a_h)\mathtt{g}(a_{h+1})\mathtt{ENcont\_cg}(\mathtt{a_k}, \mathtt{a_h}) \tag{14}$$

  where $\mathtt{g}(a_i) = 1$ if V_i is `ground`, 0 otherwise.

- *Cα-Cα* `contribution.` For *Cα–Cα* contribution we skip the first three points of the `Fragment` variable involved in the constraint, because already considered in a previous local consistent solution. Analogously to the centroid–centroid contribution we consider the pairs of every point involved in F_i, with every other `ground` point:

  $\forall k \in \{i+3, \ldots, j\}, \forall h \in \{0, \ldots, i-1\} \cup \{k+1, \ldots, j\} \cup \{j+1, \ldots, n\}$

  $$\text{EN\_CONT\_CA} = \sum_{k,h} \mathtt{ground}(a_h)\mathtt{ENcont\_ca}(a_k, a_h) \tag{15}$$

- `orientation contribution.` The energy orientation contributions involve a pair of three consecutive amino acids (see (11)), hence analogous consideration as the one made for the centroids contact contributions holds.

$$\forall k \in \{i+2, \ldots, j-1\}, \forall h \in \{1, \ldots, i+1\} \cup \{k+1, \ldots, j-1\} \cup \{j, \ldots, n-1\}$$

$$\text{EN\_ORI} = \sum_{k,h} \texttt{ENori}(a_k, a_h) \tag{16}$$

- `torsional contribution.` Torsional energy contributions are constituted by torsional and correlation of torsion component.

  $\forall k \in \{i, \ldots, j\}, \forall h \in \{i-1, \ldots, j\}$

$$\text{EN\_TORS} = \sum_{k} \texttt{ENtors}(a_k) + \sum_{h} \texttt{ENcorr}(a_h) \tag{17}$$

where `ENcont_cg`, `ENcont_ca`, `ENori`, `ENtors` and `ENcorr` are functions associated to the partial energy components, defined in Section 3.1.6.

4.2.3.3 Geometric constraints

Geometric constraints are those that involve rigid body transformations during the propagation phase. Given a list of `Point` variables $\vec{V} = V_1, \ldots, V_k$ and a list of tree-dimensional position $\vec{P} = P_1, \ldots, P_n$ describing the positions of the `Point` variables in $\vec{V}$, a geometric constraint is a k-ary constraint, whose solutions are assignment of the tree-dimensional points in $\vec{P}$ to the variables in $\vec{V}$. Following we discuss this class of constraints implemented in FIASCO.

compatibility constraint. Two adjacent fragments are said to be *compatible* if the three common points share a similar bend angle according to a given threshold.

More formally, let f_i, f_j be two adjacent fragments, with $f_i = p_i, \ldots, p_m$ and $f_j = p_j, \ldots, p_n$, and ths the threshold considered, f_i and f_j are defined to be compatible if:

$$\left| \arccos\left(\frac{a_i \cdot b_i}{|a_i||b_i|}\right) - \arccos\left(\frac{a_j \cdot b_j}{|a_j||b_j|}\right) \right| \leq \texttt{ths} \tag{18}$$

where $a_i = p_{m-2} - p_{m-1}, b_i = p_{m-2} - p_m, a_j = p_j - p_{j+1}, b_i = p_j - p_{j+2}$, and $\texttt{ths} = 0.5$.

fragment constraint. A fragment constraint is used to correlate a variable `Fragment` with the variables `Point`. A fragment constraint on a `Fragment` variable $\mathbf{F}_i$ $(i > 0)$ describe a geometric transformation of an element $f_a \in D(\mathbf{F}_i)$ which is rotated and translated according to the reference system of a previous adjacent fragment.

pair constraint. A pair constraint is used to correlate a `Pair` variable with two variables `Fragment`, and implicitly with two sequences of variables `Point`. A pair constraint $P(f_a, f_b)$, involving fragments f_a, f_b, is a geometric constraint used to transform the fragment f_b to the relative system R_a of f_a, and to couple f_b with f_a according to the pairing operation definitions. To avoid having an expensive fit operation during the exploration of the search space, both the reference system of f_a and the affine transformations, needed to couple the two fragments, are computed in the preprocessing phase.

4.3 Constraint Solving

The constraint satisfaction problem, it is implemented as a combination of consistency techniques and an *assignment+backtrack* search procedure that works by attempting to extend a local consistent solution to a global one. Exploring the complete space of all putative solution is computationally very expensive (see discussion in Section 3.2). The idea behind constraint propagation is to make the CSP more explicit so that backtrack search commits into less inconsistent assignments by detecting local inconsistencies earlier.

4.3.1 Searching solutions

The overall structure of the search algorithm used by FIASCO is shown in Procedure 2. This algorithm alternates constraint *rules iteration* (discussed in the next section) with a labeling step. The rules iteration step is performed by the AC-3 procedure (as showed in Procedure 3).

The selection step selects a variable $F \in F_{LIST}$ or a variable $P \in P_{LIST}$ representing the next fragment to be placed in the cartesian space. A `Pair` variable P over fragments F_i, F_j is selected only if the search procedure tries to extend a local consistent solution containing a choice for `Fragment` variable F_i.

According to the selection strategy, the variable selected is the one that satisfies the leftmost order—the non-labeled variable satisfying the compatibility property with the rightmost variable placed—, or the pair-first property—the

non labeled `Pair` variable satisfying a pair relation with some `Fragment` variable already ground. The instantiation step assigns every value $f \in D(F)$ to the variable F, or equivalently every value $f \in F$ referred to the second element of the pair P. An instantiation I of a value f of $D(F)$ is local consistent iff I is valid ($I[f] \in D(F)$) and $\forall c \in \mathscr{C}$ with $\mathscr{V}(c) \subset F$, $I[\mathscr{V}(c)]$ satisfies c. If the instantiation happens to be local consistent, the propagation step will result in an assignments of the point variables associated to F. The AC-3 procedure ensures

Algorithm 2: FIASCO's search procedure.

```
1  procedure: search(𝒱, 𝒞);
2  if F_LIST = ∅ then
3      return SUCCESS;
4  select F_j from F_LIST or P_{i,j} from P_LIST;
5   // F_j has length l and models Point variables P_j, ..., P_{j+l};
6  foreach f ∈ D(F_j) do
7      instantiate value f on F_j;
8      if the instantiation of f on F_j is locally consistent then
9          AC-3(𝒞 ∪ {F_j = f}, F_LIST);
10         status=search(F_LIST ∖ {F_j, ..., F_{j+l−3}}, 𝒞);
11         if status ≠ FAIL then
12             return status;
13         else
14             backtrack;
```

bounds consistency by propagating constraints. In other words it guarantee every value in a domain to be consistent with every constraint by removing those values that would cause inconsistencies.

The general structure of a propagator takes a constraint $c \in \mathscr{C}$ and, for each

variable V it involves, and each value $x \in D(V)$ checks if there exists a support on c for x. If such a support is not found, x is removed from $D(V)$. Shrinking the domain of a variable is flagged (and detected in the AC-3 procedure). An inconsistency arises when all the possible values in $D(V)$ do not satisfy c (and hence $D(V) = \emptyset$).

The AC-3 procedure uses a queue Q which contains only constraints involving those variables that have been flagged by the constraint propagation effect of some propagator. The procedure terminates whenever constraint propagation is not able to restrict any domain, or in the case it detect an arc inconsistency. The general structure of the algorithm is given in Procedure 3.

Algorithm 3: Arc consistency procedure

```
1  procedure: AC-3(𝒞, 𝒱);
2  Q ← {(V_i, c) | c ∈ 𝒞, V_i ∈ 𝒱};
3  while Q ≠ ∅ do
4      select and remove (V_i, c) from Q;
5      apply propagator associated to c;
6      if D(V_i) has been modified then
7          if D(V_i) = ∅ then
8              return false
9          else
10             Q ← Q ∪ {(V_j, c') | c' ∈ 𝒞 ∧ c' ≠ c ∧ V_i, V_j ∈ 𝒱(c') ∧ j ≠ i;
```

The backtrack procedure allows to correctly explore the space of all putative solutions, by restoring the modifications caused by the instantiation of the value f on variable F_i. It is employed whenever an instantiation makes the CSP

inconsistent, or in the case all the values of some variable's domain have been instantiated.

4.3.2 Propagation and consistency

Constraint propagation embeds the reasoning explicated by the constraints defined in Section 4.2.3, by explicitly forbidding values or combination of values of variables involved in some constraint that cannot be satisfied.

The constraint propagation rules employed in this work are based on *domain-based rules iteration*. For each constraint $c \in C$, the propagators apply a set of reduction rules with the effect of shrink the domain of the variables involved in c by removing those values that would not be contained in a tuple satisfying c. Reductions rules are sufficient conditions to rule out values that would make a local solution inconsistent. We also guarantee that every assignment satisfies the *local consistency* property. This property characterizes necessary conditions on values to belong to a solution.

As follows, we present a discussion on the implementation details of the *propagators* employed during constraint propagation. Each rule is associated to a constraint definition (given in Section 4.2.3).

4.3.2.1 alldistant propagator.

The alldistant constraint is propagated according to the following procedure. Let V_k be the `Point` variable that has caused the propagation of the alld-

istant constraint during the consistency phase, and let $P_k^\alpha = (x_k, y_k, z_k)$ be the position associated to the variable V_k (recall that V_k must be `ground`). If every point P_i^α in the neighborhood of P_k^α ($\langle x_k - d, y_k - d, z_k - d; x_k + d, y_k + d, z_k + d \rangle$) is such that $\delta(V_k, V_i) \leq d$ (here $d = 3.2$Å) the position P_k^α of `Point` variable V_k is stored, and the constraint propagation is said to be successful; otherwise the constraint is not satisfied and the propagation fails causing a backtrack. The changed values are trailed in a *trail-value stack*.

Analogously to $C\alpha$'s, centroids positions are subjected to the alldistant constraint, and they are propagated with the same mechanism described above. Centroids' alldistant constraint is propagated during the fragment and pair constraint propagation.

4.3.2.2 energy propagator.

The energy constraint is propagated whenever a `Point` variable, associated to some fragment is `changed`, and set to `ground`. The energy value is computed summing the components described by the (14), (16), (17), (15), according to the:

$$\begin{aligned} \text{Energy} \ &= (w_{\text{cont\_cg}})\texttt{EN\_cont\_cg} + (w_{\text{cont\_ca}})\texttt{EN\_cont\_ca} \\ &+ (w_{\text{cont\_ori}})\texttt{EN\_ori} + (w_{\text{cont\_tors}})\texttt{EN\_tors} \end{aligned} \tag{19}$$

where the weights are defined according to the following values: $w_{\text{cont\_ca}} = w_{\text{cont\_cg}} = 0.5$, $w_{\text{cont\_ori}} = 2$, $w_{\text{cont\_tors}} = 1$. The individual energy components are trailed in a *trail-value stack*.

Note that the energy components for a local consistent solution are computed incrementally while exploring the search tree. Most of the time, indeed, different global solutions share big portions of the local conformations. Incremental computation of the evaluation function allow us to compute the energy components for a specific local assignment only once.

4.3.2.3 Fragment propagator.

The selection of a fragment, during the exploration of the search tree, triggers a fragment constraint (in addition to the distance constraints associated to the `Point` variables involved in the fragment representation and the energy constraint). Let $\mathbf{F}_i = a_i, \ldots, a_{i+k_i}$ be a fragment of length k_i, $\vec{V_i} = V_i, \ldots, V_{i+k_i}$ represent the position of the points of $\mathbf{F}_i$, and let f_a be a labeling choice for `Fragment` variable $\mathbf{F}_i$. A fragment constraint, associated to a fragment $\mathbf{F}_i$ is woken up whenever the first tree variables $V_i, V_{i+1}, V_{i+2} \in \vec{V_i}$ are labeled and thus the remaining $\vec{V_i} \backslash \{V_i, V_{i+1}, V_{i+2}\}$ are not labeled. These labeling choices determine univocally the shift vector and the rotation matrix necessary to correctly place the points of the fragment f_a in the new reference system. More specifically, let $P^{\alpha}_i, P^{\alpha}_{i+1}, P^{\alpha}_{i+2}$ be the `Point` variables describing the amino acids a_i, a_{i+1}, a_{i+2}—recall that these variables are ground—the effect of the propagation shifts the rotated version of f_a so that it overlaps the position of V_{i+2} with the position of P^{α}_{i+2}.

More formally, let R_i be the rotation matrix which its orthonormal basis represents the reference system of $\mathbf{F}_i$, and let M_i be the rotation matrix used to best fit the first tree points of $\mathbf{F}_i$ with the points $P^{\alpha}_i, P^{\alpha}_{i+1}, P^{\alpha}_{i+2}$. Let $V^r_k = \hat{R}_i \times V_k$, with $k \in \{i, \ldots, i+k_i\}$ and $\hat{R}_i = R_i \times M_i$, be the rotated fragment associated to f_a. The shift vector $\vec{s} = P^{\alpha}_{i+2} - V^r_{i+2}$ is used to constrain the position of $a_{i+3}, \ldots, a_{i+k_i}$ as follows:

$$P^{\alpha}_k = \vec{s} + \hat{R}_i \times V_k, \;\; (k \in [i+3, i+k_i]). \tag{20}$$

Note that the first three point of $\mathbf{F}_i$ and the `Point` variables $P^{\alpha}_i, P^{\alpha}_{i+1}, P^{\alpha}_{i+2}$ identify two overlapping planes that are used to properly transform $\mathbf{F}_i$ to best place the common points, and the point V_{i+2} is superimposed to the position of P^{α}_{i+2}. Since adjacent points in fragments are naturally subjected to the next constraint, the effect of the propagation of the fragment constraint on f_a guarantees that its fourth point V_{i+3} is at 3.8Å from P^{α}_{i+2} (next property).

For every three adjacent amino acids whose positions are determined by the propagation of the fragment constraint, the alldistant constraint for centroids is woken up and propagated in an analogous way as presented above for $C\alpha$'s. In this case the distance d used is 1 Å.

4.3.2.4 Pair propagator.

Let $\mathbf{P}_i$ be a pair variable involving two fragments f_a and f_b (elements of the domain of `Fragment` variables $\mathbf{F}_i, \mathbf{F}_j$ respectively). Let $P^{\alpha}_i, \ldots, P^{\alpha}_{i+k_i}$ and

$P_j^\alpha, \ldots, P_{j+k_j}^\alpha$, $(j > i + k_i + 2)$ be the `Point` variables describing the $C\alpha$ atoms involved in f_a, f_b, and let $V_j \ldots, V_{j+k_j}$ be the local coordinates for the `Fragment` variable $\mathbf{F}_j$. Once the fragment variable $\mathbf{F}_i$ is `ground`, with position $V_i \ldots, V_{i+k_i}$, the pair constraint rotates and translates f_b according to the rotation matrix $M_{i,j}$ and translation vector $\vec{v}_j$ computed in preprocessing:

$$P_k^\alpha = \vec{v}_j + (R_i \times M_{i,j}) \times V_k, \;\; (k \in [j, j + k_j]). \tag{21}$$

where R_i is the rotation matrix describing the reference system of f_a.

In order to efficiently handle the geometric transformation, we precompute the positions of the $V_j, \ldots, V_{j+k_j}$ of f_b in the reference system of f_a. Moreover the computations for the shift vector $\vec{v_j}$ and the rotation matrix $M_{i,j}$ used to couple the two fragments is computed in $O(1)$: once the fragment f_a has been placed, we store the rotation matrix and the translation vector in a temporary array (used in the branch exploration) and use these information to transform f_b to the new reference system according to the (21).

A domain dilation is hence applied to the `Point` variables P_j, P_{j+1}, P_{j+2}, to allow some degree of freedom necessary to connect $\mathbf{F}_j$ with a predecessor fragment. Namely, the domain of P_k is dilated of a distance $d \in \mathbb{R}$ accordingly to the following:

$$D(P_k) + d = [\vec{L}^k - d, \vec{U}^k + d], \;\; k = j, j + 1, j + 2.$$

Here the value of d is 1.8 Å for $C\alpha - C\alpha$ distances and 0.5 Å for centroid-centroid

distances.

For the sake of simplicity, we omit the formal description of the constraints associated to the centroids. The centroids' positions are rotated and translated accordingly, as soon as the corresponding position of the $C\alpha$ atoms are determined.

4.4 Implementation details

4.4.1 Variables and constraints representation

The set of variables adopted in the CSP is represented by a static C struct (`VARIABLES`), that contains:

- An array of `Point` variables (`var_point`)
- An array of `Fragment` variables (`var_fragment`)
- An array of `Pair` variables (`var_pair`)
- An array of 3D-positions for the centroids representation (`centroids`)
- A real value, modeling the energy value built during the branch exploration.

All the arrays defined in this structure are static, and created during the problem definition phase.

Each variable is encoded as a C object that contains an identifier `idx` i.e. the record position in the variable-type array (according to the type: point, fragment, or pair); the flags indicating whether the variable is `labelled, ground, failed` and `changed`; the domain representation for the variable; the index of the last

variable trailed in the value-trail stack `lasttrailed`, and a list (implemented as a dynamic array) for the constraints dependencies `constr_dep`, i.e. the constraints to be checked whenever the current variable changes its values. According to the type of variable represented the domain is modeled as a static array of `FRAGMENT`, a static array of `PAIR` or as a pair of `POINT` (`upper_bound, lower_bound`) to represent the upper and lower corner of a point variable box domain.

The possible states of each variable are:

- `labeled`. Indicates that the variable has been selected, labeled and included in the search tree. A value of -1 indicates that the variable has not been labeled; any value grater than 0 denotes a labeling that was made on the variable. For `Point` variables it is treated as a flag, while for `Fragment` and `Pair` variables its value indicates the index of the domain element chosen for the branch exploration.

- `ground`. For `Point` variables, it indicates that the variable has a unitary domain size, i.e. `lower_bound` = `upper_bound`. This case is verified either by an explicit value assignment or because of application of propagation techniques. For `Fragment` variables it indicates that every `Point` variable associated to the fragment has unitary domain size. For `Pair` variables, it indicates that the first `Fragment` of the pair is ground and that all the `Point` variables from the third to the last, that are associated to the second

fragment of the pair, have a unitary domain size.

- `failed`. Indicates that the domain of the variable has become empty.
- `changed`. Indicates that the variable is in none of the other states and its domain has just been modified.

When a `Point` variable is set ground, we store its domain value into a data structure (referred as *grid*) that represent a discretized version of the 3D space. Every *cell* of the grid is associated to a discrete 3D interval. More formally, every cell g of the grid G is described by a function: $h : \mathbb{N}^3 \rightarrow G$

$$h(x, y, z) = (x + \frac{d}{2}) + (y + \frac{d}{2})d + (z + \frac{d}{2})d^2,$$

where d is the grid dimension i.e. the original number of cells present on one side of the grid.

The grid structure is useful to efficiently verify the consistency of k-ary distance constraints in local consistent solutions. We employ such formalism to verify local consistencies of the alldistant constraint: whenever a `Point` variable P is set to ground, the alldistant constraint consistency is exploited looking at the neighboring points P_i of P. That is, all the points P_i such that:

$$P.x - \epsilon \leq P_i.x \leq P.x + \epsilon,$$

$$P.y - \epsilon \leq P_i.y \leq P.y + \epsilon,$$

$$P.z - \epsilon \leq P_i.y \leq P.z + \epsilon.$$

where $\epsilon = \lceil \frac{\texttt{ALLDIST\_THS}_{C\alpha}}{d} \rceil$. The aim of exploiting such constraints during the search phase is anticipating the consistency and the propagation effect of the active constraints.

We carefully calibrated the space such grid should reserve, and we dynamically allocate more gird cells when needed (e.g. when the conformation folds on those areas of the grid not yet defined), preventing the deallocation—the maximum number of cells needed is bounded by the radius linear extension of the target protein.

A constraint is encoded as a C object and described by an unique identifier `idx`, a `type` that describe the type of constraint it represent, a distance value `dist` (used by the application of distance constraints) representing the Euclidean distance. The list of `Point`, `Fragment`, and `Pair` variables possibly involved in the constraint are represented trough dynamic arrays: `point_var, frag_var, pair_var`, respectively. An additional list of `Point` variables (`caused_by_point_var`), one of `Fragment` variables (`caused_by_frag_var`) and one of `Pair` variables (`caused_by_pair_var`) are stored in the constraint data structure. These lists are used to store the information about the variable which values is changed due to the propagation of current constraint, and which will possible cause further propagations.

The collection of constraints present in the CSP is represented by a *constraint store* implemented as a dynamic array. At each moment of the search exploration, the constraint store maintains the constraints to be satisfied in order to fulfill the local consistency properties.

In order to efficiently handle the consistency procedures, an additional list of constraints `clist` is created. This is modeled as a queue and contains all the constraints to be checked during the Arc Consistency procedure. More specifically, during the consistency phase, after the modification of a domain of some variable, the set of constraints involved in possible further propagation operations is retrieved from the list of constraints (`constr_dep`), present in each variable. This allow to efficiently add the dependency constraints (the one linked with the variable changed) to the Constraint Store. Recall that the `constr_dep` arrays lists the constraints in which the variable is involved. Moreover we introduce additional lists of constraints (`changed_point`, `changed_frag`, `changed_pair`) that provides direct access to the constraints involving a specific `Point`, `Fragment`, or `Pair` variable. The variables changed by the application of some constraint—stored in the lists `caused_by_point_var`, `caused_by_frag_var`, `caused_by_pair_var` of a woken up constraint—are used in order to efficiently add the constraints to which they are associated with (stored in the `constr_dep` arrays) to the Constraint Store for further application of arc reductions during the AC-3 procedure (see Algorithm 3).

The exploration of the search space is handled by a standard *propagation + backtrack* procedure. To efficiently handle the backtracking procedure, we have implemented a *trail-value stack* data structure. Every time a variable is going to be in the `changed` state, its value is stored in the trail stack, together with the variable description. The backtracking procedure is called whenever a domain assignment is found to be inconsistent or to move to a different branch of the search tree. The call to this procedure, enforces the restoring of the domain values of the variables involved in the branch exploration. These values are retrieved from the *trail-value stack* (in a reverse order w.r.t. the order in which they were stored).

4.4.2 Search Space

Searching all putative protein conformations can be represented as the exploration of a search tree. In this abstraction, a path represents the sequence of possible choices (nodes of the tree) made during the assembly of a local consistent solution. At each level, the tree branches correspond to the candidate choices (compatible fragments, or pairs) that can be selected, and describe the domain for the selected variable. Selecting one of the possible candidates choices is referred, in the search procedure, as the *labeling phase*. The edges connecting nodes of the tree denote the propagation effect of the candidate choice made at the branch exploration. A solution to the CSP is represented by a complete path from the root node to a leaf.

The search procedure (see Algorithm 2) is performed in three main steps:

1. Select a variable to instantiate (*variable selection*).
2. Guess a value for the selected variable (*labeling*).
3. Check the consistency of the guessed value and propagate the effect of the choice.

For the first point we have implemented two variable selection strategies: a *left-most* strategy that selects the non instantiated variable with lowest index and a *pair-first* strategy to build the conformation blocks marked by a pair relation with higher precedence. More details about the selection strategies will be provided in the following Section.

In turn, in the labeling step, the values of the domains of the variables selected are attempted starting with the most likely choice, or with the one maximizing local interaction (when applicable).

At the implementation level, the information related to the branches explorations is stored in the data structure `domain_table`, a matrix representation for the explored choices (`var_domain`) associated to every variable selected. The columns of this matrix represent the variable selected to solve the CSP—associated to each level of the tree—while the rows identify the domains associated to the selected variables—implemented as a bitmask to represent the status of the possible variable labeling. In order to allow an efficient retrieval of the list of choices made

(the path from the root node to the node heading the current subtree) we store the information about the type of variable selected at each branch (`var_type`), and the indexes of the variables domain element labelled (`label`). These information are, in addition, used to enable a fast reconstruction of the node in the concurrent exploration of the search space (see Section 5.3).

The information about the path exploration, stored in the `domain_table`, allow an efficient handling of the backtracking procedure, and an efficient detection of the completion of a branch. Moreover, this representation avoid the need of an explicit search tree—siblings nodes are represented only once for every level of choice.

The effect of the labeling phase results in propagating the constraints associated to the labeled variable (whenever the values chosen result to be consistent with the current local solution). This, in turn, as discussed in section 4.3.2, has the ultimate effect of determine the position of a set of points in the cartesian space.

Each step of propagation leads to modifications of the variable domains, and consequentially of the data structures that represent them. In order to properly apply the backtrack procedure, we implemented a mechanism that restores the correct state of computation allowing the search to move to a different branch. This operation is handled by keeping track of the effect of propagation in a *value-trail stack*, and using it to undo the modifications while backtracking. The value

trailed in the stack include information about `Point`, `Fragment` and `Pair` variables values, energy components and position of the points in the grid support structure. These values are restored in the reverse order in which they where pushed into the stack.

Variable Selection discussion. In this paragraph we report a detailed discussion about the variable selection strategy implemented in this work. We have implemented two variable selection strategies:

leftmost strategy. This strategy does not make use of the pair elements. The collection of `Fragment` variables is viewed as a list in which its i^{th} element is the `Fragment` variable associated to fragments having a_i as first point. This strategy selects the leftmost non-`ground fragment` variable for the successive labeling step.

pair-first strategy. This strategy is described in Procedure 4. Let f be the labeling choice for the previous variable F_i instantiated. There are two basic cases:

1. The last variable instantiated F_i is of type `Fragment` and the element of the domain selected for the labeling phase $f \in D(F_i)$ is a fragment of type *special*. Note that this case also cover the one in which the last variable selected was a `Pair` P describing a relation in which f is the

second fragment of the pair.

Three cases may occur:

(a) If there exists a relation $\rho(f, g)$ between fragments $f \in D(F_i)$ and $g \in D(F_j)$ with variables F_j not yet instantiated, then the `Pair` variable P describing this relation is selected (lines 4-5). Labeling a pair variable will propagate the effect of wakening up a pair constraint over the point variables associated to F_j as described in section 4.3.2.4.

(b) The second case is illustrated in lines (6-9) when no `Pair` variable can be selected (yet being the P_{LIST} set non-empty). Let $I_1, \ldots, I_h$ be the intervals of non instantiated (`Point`) variables and let denote with $s(I)$ $(e(I))$ the index of the first (last) variable in I. The search continues by selecting the fragment variable starting the shortest interval of non instantiated variables containing some special fragment (that is possibly involved in a pair relation).

(c) If no special fragment can be found in any of the intervals of non instantiated variables, the variable selection returns a candidate according to the leftmost strategy (line 11). This case also applies whenever all the `Pair` variables have been already instantiated—$P_{LIST} = \emptyset$—(line 15).

2. The last variable instantiated F_i is of type *standard*. In this case F_i can only be a `Fragment` variable. Again two cases may occur. If there are `Pair` variables not still instantiated, points 1b and 1c are re-proposed. Analogously to the previous case, if all the `Pair` variables have been already instantiated the selection returns the variable that satisfies the leftmost property.

Algorithm 4: The algorithm for the variable selection strategy

```
1  procedure: SelectVariable(f ∈ F_i);
2  if f ∈ 𝓕_spc then   // Includes case F_i.type = PAIR
3      if P_LIST ≠ ∅ then
4          if ∃g ∈ F_j ∧ g ∈ 𝓕_spc, ∃P ∈ P_LIST s.t. P = ρ(f, g) then
5              return P
6          else if ∀g ∈ 𝓕_spc, ∄P ∈ P_LIST s.t. P = ρ(f, g) then
7              if ∃F_j ∈ 𝓕_spc s.t. j ∈ I_h then
8                  select F_j s.t. j = min_h{j − s(I_h)};
9                  return F_j
10             else
11                 use leftmost strategy;
12     else                                   // P_LIST = ∅
13         use leftmost strategy;
14 else if f ∈ 𝓕_std then
15     if P_LIST ≠ ∅ then
16         execute lines: 6–11;
17     else                                   // P_LIST = ∅
18         use leftmost strategy;
```

Essentially the *pair-first* selection strategy tries to label first all the choices for the `Fragment` variables of type *special*, possibly involved in some pair relation. The rationale behind this choice is the hope in producing the highest constrained

search space, before trying the more difficult task of "closing the loops" (i.e. complete that parts of the search space representing the choices of high variability in a protein conformation, where no homology information are retrievable).

Note that the selection strategy described above attempts to place `Pair` variables in a cascade effect: the selection of a pair $\langle f_a, f_b \rangle$ can trigger at the next step the selection of a pair $\langle f_b, f_c \rangle$ if such a relation exists. The intuition here is that, multiple pair relations among fragments might denotes the presence of groups of structures in the target protein, characterized by strong local interactions. These local forces, bestowing high stability to the polypeptide, might give a strong hint in how the final 3D protein structure looks like. Indeed, once these blocks have been built, we guarantee the highly homologous conserved structures, to be positioned optimally w.r.t to their pair mate—recall that the position of the fragments in a pair are driven-oriented in order to maximize the energy function.

In last analysis, we exploit the homologies information of the *special* fragments to simplify the CSP by reducing, as much as possible, the number of "blind choices" to be performed in order to place fragments blocks. Note that, once all the *special* fragments have been placed, the remaining choices covers those areas of the polypeptide characterized by higher degree of freedom (i.e. random coils and loops).

4.5 Results

In this Section we provide the experimental results obtained using FI-ASCO. We propose a test set of 20 proteins, including globular proteins and viruses, for which a predicted structure exists in the PDB.

In Section 4.5.1 we report the methods and the parameters adopted to configure our solver. In Section 4.5.2, we analyze the performances of the system comparing it with TUPLES [DDFP11], a declarative-based solver, which inspired our current solution. Finally, in Section 4.5.3, we discuss the quality of the solutions by comparing our predictions with their native state structures from the PDB.

4.5.1 Methods

Assembly DB fragments. Recall, from Section 3.1.3, that a *fragment* of the Assembly-DB is a sequence of L contiguous points in $\mathbb{R}^3$, and it is used to model "short" part(s) of a target protein sequence. These elements constitute the building blocks to assembly a candidate protein conformation and, when additional information about the protein secondary structure is given, such fragments are mainly employed to model loops and those parts of a protein characterized by high variability. The value L, for the fragments generated in the Assembly-DB is set to be 4[1]. Note that, such setting represent the shortest length it can be employed, in our model, to satisfy the compatibility constraint guaranteeing the

[1]Each fragment in $\mathscr{F}_{std}$ has a backbone composed by 4 $C\alpha$ atoms.

presence of an overlapping plane (see Section 4.3.2.3). Using this setting every selected fragment contributes to model a single point in the construction of a conformation.

Amino acids clustering. In order to produce a statistically significant analysis of the PDB, when generating the Assembly DB, we cluster the set of amino acid $\mathbb{A}$ into 9 classes (see 3.1.5). The amino acid clustering classes used in this work are shown in Table 4.1. The number of tuples of length 4 generated from the *top-500* data-set is $62,831$ and the number of different 4-tuples generated according to the partition of Table 4.1 is 6561, where 5830 of them are covered by some template structure. To model the remaining structures, we use a "general fragment" containing the most statistically relevant elements of the Assembly-DB. The threshold ϵ defining the similarity (3.1.8) of two fragments is set to $\epsilon = 0.5$ Å.

$\mathcal{A}_1 = \{\texttt{Ala}\}$	$\mathcal{A}_2 = \{\texttt{Leu, Met}\}$	$\mathcal{A}_3 = \{\texttt{Arg, Lys, Glu, Gln}\}$
$\mathcal{A}_4 = \{\texttt{Pro, Asp, Ser}\}$	$\mathcal{A}_5 = \{\texttt{Thr, Phe, His, Tyr}\}$	$\mathcal{A}_6 = \{\texttt{Ile, Trp, Val}\}$
$\mathcal{A}_7 = \{\texttt{Cys}\}$	$\mathcal{A}_8 = \{\texttt{Gly}\}$	$\mathcal{A}_9 = \{\texttt{Pro}\}$

Table 4.1: Amino acid clustering classes.

It follows that for fragments of length 4, the sum of squared Euclidean distances between the components of the two fragments will be within 1 Å.

Variable domain size. The number of different fragment representatives generated for a given 4-tuple can be huge. For the most recurrent 4-tuples we record

over 100 representatives. Such values are in direct relation to a `Fragment` variable domain, and implicitly to the size of the protein conformation search space. To prevent a huge search space, growth of the search space, we limit the size of the `Fragment` variables domains to (at most) 10 elements. According to statical analysis, such elements are among the top-10 most statistically relevant.

Energy contributions weights. The energy contributions `EN_CONT_CG`, `EN_CONT_CA`, `EN_ORI`, `EN_TORS`, defined respectively in Equations 14, 15, 16 and 17 are weighted according to the following values:

$$w_{\texttt{en\_ori}} = 2.0, \quad w_{\texttt{en\_cont\_ca}} = 0.5, \quad w_{\texttt{en\_cont\_cg}} = 0.5, \quad w_{\texttt{en\_tors}} = 1.$$

The distance within which the centroid contact potential contributions are maximized, is set to the summation of their radius; the $C\alpha - C\alpha$ contact contributions it is set to 4.8 Å.

Distance constraints. The distance between two $C\alpha$ atoms, $\texttt{ALLDIST\_THS}_{C\alpha}$, that must be exceeded in order to satisfy for the `alldistance` constraint is fixed to: 3.2 Å. For pairs of centroids, the distance $\texttt{ALLDIST\_THS}_{\texttt{cg}}$ is set to 1 Å.

Recall that we use a discretized representation of the 3D space to efficiently handle the `alldistance` constraint thought the *grid* data structure (see 4.4.1). The grid size d, specifying the number of cells for each dimension is set to $d = 128$, and the size cell size is set to 3 Å. That is, for every `Point` variable P in a cell

$c_{i,j,k}$, the consistency of the `alldistance` property is exploited by looking at the points contained in the neighboring cells ($c_{i\pm\Delta,j\pm\Delta,k\pm\Delta}$), and $\Delta = 2$.

Special Fragments. In every protein tested we specify the secondary structure information as defined in the corresponding PDB report. We model the subsequence corresponding to such predictions by the use of user defined (or *special*) fragments. However it is important to note that the tested proteins are not included in the `top-500` database used to generate the Assembly-DB.

For the experimental results, discussed in the following Sections, each computation was performed on an Intel Xeon E5335 2.00GHz (using a single processor core) with 4096 KB of cache size. The operating system is Linux, distribution CentOS release 5.2, and we compile the code using g++ 4.1.2 with the $O3$ optimization flag.

We evaluate 20 proteins with known conformation from the PDB database, and perform an exhaustive search of 2 days.

4.5.2 Efficiency

The aim of this section is to evaluate the efficiency of our solver in terms of computational time. We do so by comparing FIASCO with *TUPLES*, by Dal Palú et al. [DDFP11].

Recall that FIASCO is the imperative (C) extension of TUPLES. A fair comparison is ensured in terms of model and search strategies adopted.

The test set chosen for this evaluation, is the one provided in [DDFP11], and we use the original results [DDFP11] of TUPLES for a direct comparison. We need to point out that for the experiments showed in [DDFP11], TUPLES has been tested on an AMD Opteron 2.2GHz machine, whilst our system runs on a 2GHz CPU. For a fair evaluation, this computational power gap should be taken into account. However, since the computational power adopted to test our system is strictly smaller then the one used by TUPLES, we will ignore such differences, still being able to deliver a substantial speed improvement.

Table 4.2 reports the results obtained for 9 proteins of length ranging between 12 and 60 amino acids, for which an exhaustive search of 2 days was performed. The first two columns indicate the protein name and length. For each solver we report Rmsd, Energy and time obtained by the generated solution. The execution times (expressed in seconds (s) or minutes (m)) indicates the time needed to compute the best solution within the a fixed 2 days time limit (according to the specifications of [DDFP11]).

In both comparisons the knowledge deriving from secondary structure predictions is available and modeled as a constraint.

As a first consideration we would like to point out that, conversely from TUPLES, FIASCO was able to exhaustively explore the search space for the majority of the protein considered (only the tests for 2IGD and 1ENH were interrupted due to time limit). In every test executed, FIASCO outperformed TU-

Protein		FIASCO			TUPLES		
PID	N	RMSD	Energy	Time	RMSD	Energy	Time
1KVG	12	3.42	-11.66	0.05s	2.79	-59.122	9.88s
1LE0	12	4.16	-18.51	0.04s	3.12	-45.575	3.20s
1LE3	16	4.28	-81.62	0.28s	3.90	-69.017	218.79s
1EDP	17	3.34	-38.73	0.38s	3.04	-112.755	73.00s
1PG1	18	2.82	-14.85	0.23s	3.22	-109.456	11.00s
1ZDD	34	3.75	-229.17	0.75s	4.12	-231.469	1290m
1VII	36	5.04	-241.18	0.148s	7.06	-263.496	1086m
2GP8	40	5.08	-296.90	0.91s	5.96	-266.819	5794.88s
2K9D	44	5.59	-356.92	17.31s	6.99	-460.877	1453.44s
1ENH	56	5.58	-634.43	72m	5.10	-467.014	142m
2IGD	60	7.27	-672.50	1992m	16.35	-375.906	2750m
1AIL	73	6.67	-823.18	6.2m	9.78	-711.302	301m

Table 4.2: Efficiency results

PLES, in terms of computational time, being able to generate optimal solutions (or sub-optimal solutions for targets 2IGD and 1ENH) remarkably faster. The estimated average computational time gains are within 3 orders of magnitude.

The cause of such speedup are attributed to the careful imperative implementation (C) on which FIASCO relies.

4.5.3 Quality of the results

In this Section we analyze the predicted structure by a comparison with their native state retrieved from the PDB data base. The accuracy of a prediction is evaluated in terms of RMSD (see Equation (7)) where only the backbone atoms are taken into account.

For this experiment we return the set of the 10-best predictions in terms of energy minimization, and we report the best conformation found (in terms of

RMSD). According to our observations the top-10 solutions constituite a good set of representatives for the portion of the search space explored.

In Table 4.3 we report the RMSD of the best conformation, among the top-10 predicted by FIASCO within a 2 days computational time limit. For every protein analyzed, we also report the energy associated, the time needed to generate the reported solution, the value (min RMSD) obtained using the *RMSD* as the COP objective function, and the number of user defined fragments F_{spc} given in the input specifications.

PID	N	RMSD	Energy	Time	min RMSD	N F_{spc}
1KVG*	12	4.10	-14.27	0.59s	2.26	0
1LE0*	12	4.16	-33.06	0.04s	1.28	0
1KVG	12	3.42	-11.66	0.05s	2.26	2
1LE0	12	4.16	-18.51	0.04s	1.28	2
1LE3	16	2.99	-12.45	0.06s	1.93	2
1EDP	17	3.34	-38.73	0.38s	2.39	2
1PG1	18	2.82	-14.85	0.23s	2.21	2
1EN0	25	3.04	-83.51	0.07s	1.66	2
1ZDD	34	3.59	-194.28	0.04s	1.73	2
1VII	36	3.75	-227.61	0.24s	2.12	3
1E0M	37	6.15	-274.18	73,616s	3.79	3
2GP8	40	5.08	-296.90	0.91s	3.90	2
2K9D	44	5.59	-356.92	17.31s	1.85	3
1ED0	46	9.70	-408.07	2432s	3.92	4
1ENH	56	5.58	-634.43	72m	3.26	3
2IGD	60	7.27	-672.50	1992m	5.84	4
1H40	69	8.35	-836.91	20,626s	3.73	4
2ZKO	70	4.77	-840.46	109.22s	2.73	3
1AIL	73	6.67	-823.18	376s	2.86	2
1AK8	76	9.54	-738.64	114,3s	5.45	4
1DGN	89	7.74	-991.41	3851.6	3.62	5
1KKG	108	10.79	-1,266.04	13,2s	9.33	4

Table 4.3: Qualitative results

It is interesting to note that, for small proteins an accurate prediction can be given (within 3 Å) and for medium size proteins the given predictions often ranges between 3 Åand 5 Å. This indicates that the best solution returned, describes well the folding of the polypeptide, preserving the dominant structural characteristics. In Figure 4.1 we show the predicted structure for protein 2ZK0[2], together with the user defined fragments inputed (respectively displayed at the bottom and top of the Figure). The native structure is displayed in blue (darker color) and the predicted one is in orange (lighter color). In the fragment block representation we show the protein primary sequence together with its secondary structure prediction. Each yellow box denote a user defined fragment $F_i \in \mathscr{F}_{spc}$, selected according to the suggested secondary structure prediction, and defines a geometric constraint. From the representation displayed it is possible to observe that the structural traits of the protein are well captured. On the other hand, the areas characterized by the loops and random coils are not often modeled with high accuracy, negatively affecting the whole predicted conformation.

We also report two de-novo prediction (no additional information used in terms of secondary structures) for proteins 1KVG and 1LE0 (indicated with a "*" in Table 4.3). Even in these experiments, we observed that the main characters of the chains (e.g. secondary structures) can be reconstructed with good affinity,

[2]Chain A of Structural basis for dsRNA recognition by NS1 protein of human influenza virus A.

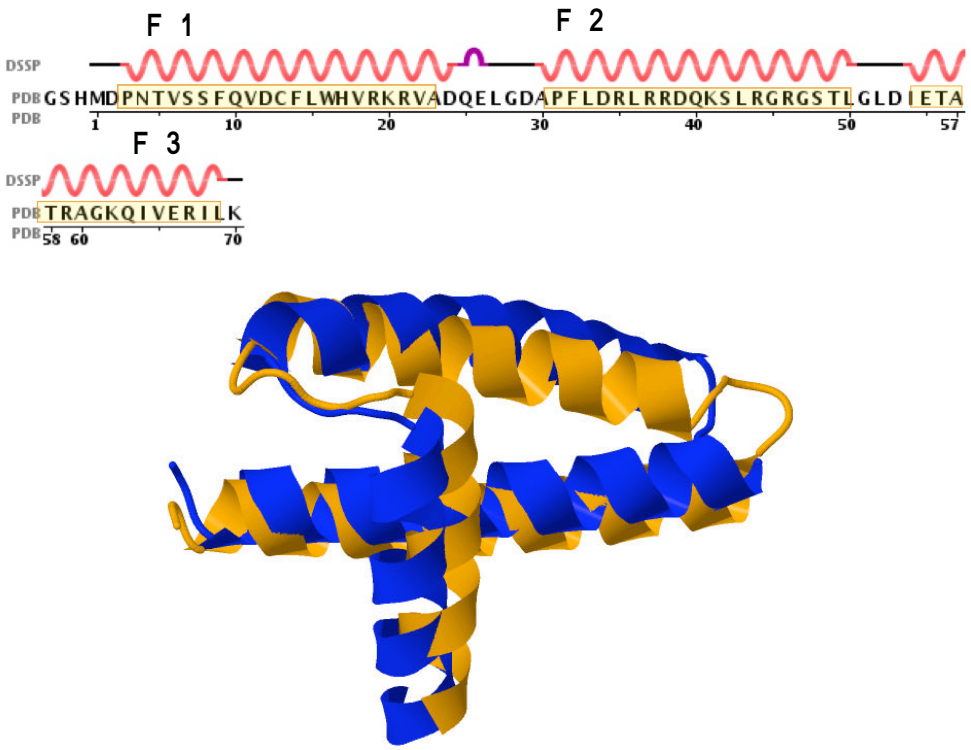

Figure 4.1: 2ZKO user defined fragment (top) and structure prediction (bottom). The native structure is displayed in blue (darker) and predicted structure in orange (lighter).

using the current fragment set, although the most variable parts of the proteins can difficultly be captured. At a first glance this may be interpreted as a weakness of the fragment set employed for the predictions, in that does not provide a good structural coverage. Interestingly, in a successful investigation, we observed that in the vast majority of the cases, the Assembly DB suffices as a provider of structural information to build a good final conformation (i.e. one that is close enough to its native state). In such experiments we set the constraint optimization problem to minimize the RMSD measure rather then the energy function. The results are reported in the penultimate column of Table 4.3. It is possible to observe that, the

best structure are predicted with an error ranging within 1–2 Å for small proteins, and 2–3 Å for medium size proteins. This implies that a good solution to the problem exists in the search space generated.

The source of the lack of high accuracy in modeling the loops, could find its roots in the use of the approximated protein representation and in the energy model adopted. In particular the latter, is accurate in capturing the proteins substructures occurring with high frequency in nature (e.g. secondary structures), but unconditionally packs those protein regions characterized by a higher flexibility. These regions shall be characterized by a less stability in terms of free energy (weaker bonds are formed) and consequentially they should not be tightly packed as the rest of the sequence does.

Investigations in these aspects is object of future works. We plan to build a loop closure model relying on statistical considerations and on the topology of the super-secondary structures connected by such loop [Lau04, FFOF06, TBOdB09]. Moreover, a migration to a full atom model for protein backbone representation, would reflect in a more accurate fragment set to be used by the assembly procedure, and a more refined energy model.

An additional observation needs to be provided for the prediction of proteins 1GP8, 2IGD and 1KKG. In these tests the search space generated is remarkably large due to the branching factor. Indeed even minimizing the RMSD measure does not provide predictions that are close enough the native state in the

fixed computational time.

CHAPTER

FIVE

FIASCO: A PARALLEL AND MULTITHREAD SOLUTION

In this chapter we explore the use of high performance computing solutions to enhance the performances of FIASCO and improve the quality of the results.

In Section 5.1 we describe the model adopted in the parallel investigation. In particular, we exploit an MPI-based cluster distributed solution, in multi-core platforms with a thread-based approach (described in section 5.2). In Section 5.3 we focus on the scheduling and communication techniques on which FIASCO's parallel system relies. In section 5.4 we discuss the main techniques used to exploit an efficient load balancing strategy. Finally, in Section 5.5 we present a discussion on the efficiency of the new framework and its scalability.

We show that the use of concurrent constraint solvers, in the context of fragment assembly, allows to exploit the analysis of proteins of size up to 300

amino acids.

5.1 Parallelization of FIASCO

The process of exploring different choices of fragments allows us to view the space of all the putative conformations as a search tree; the nodes of the tree represent points of choice (e.g., choice of the fragment for a given small amino acid sequence) and the creation of a conformation for a primary sequence corresponds to the exploration of a path from the root node to a leaf of the tree.

In order to speedup the process of exploring the search space, we developed a *parallel* version of the constraint solver. The approach introduces a number of concurrently operating constraint solvers—from now on referred to as *agents*—each operating on a different subtree of the search space.

5.1.1 Concepts of search parallelism

The parallel version of FIASCO is based on the exploitation of *search parallelism* from the search tree [Per99, GPC+01, MSVH09]. The main idea consists in allowing different agents to explore different subproblems. In the context of resolution of a CSP, expanding local consistent solutions, located at different positions of the search tree, is an independent task and can be performed concurrently by the agents without need of communication. The need of communication arises in the context of relocating a concurrent solver to avoid overlapping subproblems and in the context of the resolution of a COP (e.g. to propagate bound information).

We focus on the nondeterministic exploration of the search space by dynamically characterizing the decisional branches of the tree, so to be able to fulfill an efficient rescheduling mechanism. Following the approach of [Per99] we define accessible parts of the search tree to allow different agents to conduct an exploration of a portion of the tree. Figure 5.1 illustrates two agents concurrently exploring different parts of the search tree. The agents share a common path, that identifies the decisional choices made to head the root node of the target subtree. An agent that accesses to a specific subtree is in charged of complete the associated search space exploration autonomously. However, upon request, parts of the subtree can be shared with other agents.

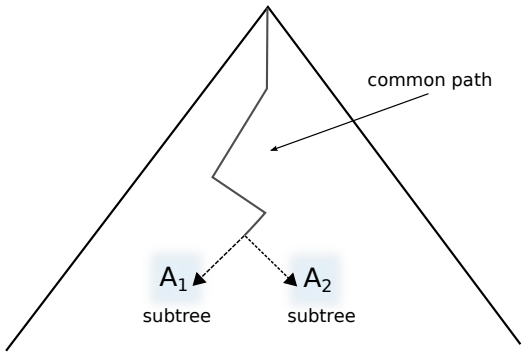

Figure 5.1: Concurrent search tree exploration

To efficiently share parts of the tree, avoiding overlapping searches and unless re-computations, we propose a fully decentralized solution implemented upon a two level parallelization: at a cluster level, each agent, possibly executed on a different processor, communicates to other agents through the MPI protocol; at a multithread level, we define two main tasks to be carried by an agent (com-

munication and search exploration) and implemented them as two independent threads. The cluster based together to the thread-based parallelization results in a minimization of the overall amount of communications in a clean modular structure.

5.1.2 Agents

Let us introduce the following notation. Let $\mathcal{A}$ be the set of agents used for a parallel computation, with $|\mathscr{A}| = n$. Denote with A_i the a^{th}gent in $\mathscr{A}$ (when a ring topology is considered).

Each agent in $\mathscr{A}$ is in charge of conducting two activities, each implemented as a separate thread:

1. constructing the subtree of the search space assigned to the agent;

2. enabling the agent to relocate to different parts of the search tree whenever necessary (e.g., whenever the subtree is completed).

The two activities are relatively independent of each other; while the construction of the subtree is a purely "local" activity (conducted by a *worker thread*), that affects exclusively the specific agent, the movement of agents in the search tree requires communication among them (e.g., to locate unexplored subtrees)—this is performed by the *scheduler thread*.

The idea is to keep the overall average amount of tasks, repartitioned to each agent, as balanced as possible. To do so we rate each task with a cost

expressing the *quality of a task*. Such qualitative measure is introduced to give each agent a coarse portrait of the global search exploration status. Agents use these information to employ the best[1] decisional processes in the load balancing strategy.

5.1.3 Tasks and Task Queues

The basic unit of interaction among agents is referred to as a *task*. A task contains the information related to the decisional choices the worker agent has performed during a partial exploration of the search tree and it can be viewed as a path from the from the root node to a node rooting an unexplored subtree.

To describe a task, we adopt the minimal information, to be used by a worker, to locate a specific unexplored subtree, i.e. the list of the variable selected, together with their type, labeling choice, and explored domains, that are made during a the generation a partial consistent solution.

In the context of concurrent search space exploration, a challenge is to resume the search of a constraint solver on an arbitrary unexplored subtree, ensuring a low overhead. This can be seen as the problem of efficiently handling a path reconstruction.

The structure of the tasks generated in FIASCO, allows a solver to reconstruct the path from the root node to the node rooting the subtree to explore,

[1] According to the perceived knowledge of other agents task cost.

simply by enforcing the chain of choices for the variables values, as specified in the task description. Note that these labeling choices are performed without enforcing the bound consistency checks and by delaying the propagation of the constraints woken up, as late as possible: e.g. right before starting the actual exploration of the subtree.

We define two types of tasks, based on the amount of additional information and actions required to an agent in order to start the execution of the task (i.e., construction of the corresponding subtree). This concept is referred as *task heaviness*.

Definition 5.1.1 (Heavy Task)**.** A task that is missing the runtime information (constraint store, value-trail stack, constraint status, variable domains, etc.) on the path from the root node, is defined *heavy task*.

An heavy task requires a reconstruction of the path from the root node, before the exploration of the subtree associated can begin.

Definition 5.1.2 (Light Task)**.** A task for which the runtime information on the path from the root node is already available is defined *light task*.

Figure 5.2 shows a schematic representation of the task heaviness in terms of the information contained in each class of tasks. In particular, an *heavy task* contains exclusively the description of the explored path, while a *light task* supplement it with the internal constraint solver information.

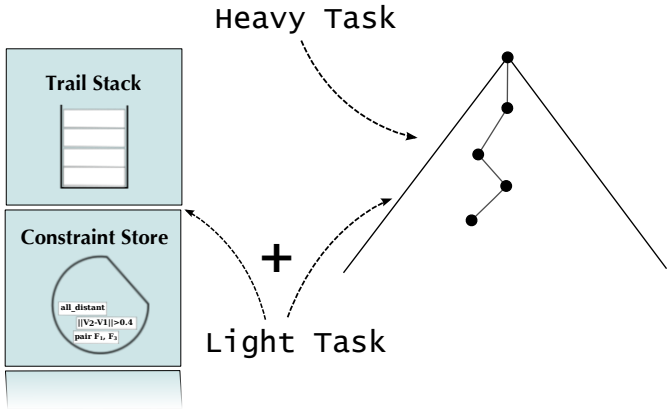

Figure 5.2: Heavy and Light task description.

To efficiently handle the different type of tasks, every agent in the pool has access to two task queues, differentiated in the nature of task they contain. Let denote with Q_H and Q_L the heavy and the light task queues respectively. Q_H represent a set of nodes received from other agents, while Q_L is represented by a structure that describes the ongoing exploration of a subtree.

5.1.3.1 Task Description

Due to the irregularity of the search tree structure, we implemented a dynamic load balancing strategy that uses distributed information on the exploration state of each worker agent.

In order to exploit the information used in the dynamic load balancing strategy, we introduce a qualitative description of a task. The main idea is to capture information characterizing the difficulty of a task, and how long it is believed a task will keep a worker busy. In last analysis, these information are

used by every agent in $\mathscr{A}$ to select the best area of the tree where to relocate (when needed).

In the following paragraphs we discuss the individual components used to qualitatively describe a task.

Load. We introduce two descriptors, l_w and l_s, representing, respectively the level of the subtree from which an agent has started/resumed its exploration, and the level from which an agent could give away some task to serve incoming requests.

According to the position of l_s in the search tree w.r.t. its height, we are able to guess the heaviness of the task in terms of the number of nodes to be explored in order to find a solution.

This concept is referred in our description as the *load* of a task[2]. The *task load* applies to the intuition that the more l_s is close to the root node, the largest (unless branch and bound) is the subtree described by such task. Analogously, when l_s is close to the maximum height of the tree, inverse consideration hold.

Difficulty. Let define W_M and W_m as the maximum and minimum height of the tree explored by the worker, within a fixed window history (i.e., a certain number of visited nodes). Observing the difference between the maximum and minimum

[2]Note that a task is the collection of the information to be given away to serve incoming task requests.

level $W_M - W_m$, in a fixed window size, we are able to capture the pace at which an agent proceeds in the vertical exploration of tree. We use an exponential moving average to smooth out short-term fluctuations and highlight longer-term trends, defined by the followings:

$$W_M(t) = \alpha\ W_M + (1 - \alpha)\ W_M(t - 1) \tag{22}$$

and

$$W_m(t) = \alpha\ W_m + (1 - \alpha)\ W_m(t - 1) \tag{23}$$

where $W_M(t)$ and $W_m(t)$ are the average maximum and minimum levels explored at a time t, and $0 \leq \alpha \leq 1$. In our experiments we found that a good value for alpha is 0.3.

Availability We introduce the concept of *availability* to describe how rapidly the available (light) tasks will be consumed by the current agent. With this descriptor we intend to capture the probability that the tasks present in the L_Q will be still available to serve an incoming task request.

Let N_a be the set of nodes that can potentially be given away to serve a single incoming request, i.e. the non explored siblings of the node at level pointed by l_s. Note that the distance between l_s and $W_m(t)$ describes the distance between the next level from which it is possible to take tasks to give away and takes into account the average number of backtracking moves performed during the

exploration of current subtree. Intuitively, this quantity represents an estimation on how quickly the nodes in N_a will be consumed by current *worker* agent.
The concept of *availability* of nodes to give away, is defined by the following:

$$\max(0,\ W_m(t) - l_s) \tag{24}$$

The more l_s is far apart from $W_m(t)$ the longer we expect the choices in N_a to be available to serve incoming task requests.

Combining the arguments presented above, we are able to model a qualitative description of the task, currently being executed by some agent, with respect to the number of nodes it will be able to give away to fulfill an incoming request. Let $\kappa : \langle \mathbb{N} \times \mathbb{R} \times \mathbb{R} \rangle \to \mathbb{R}$, the function $\kappa(l_s, W_m, W_M)$ is defined by:

$$|N_a|\Big(w_l\big(1 - \frac{l_s}{H}\big) + w_d\big(\frac{\max(0, W_m - l_s)}{H}\big) + w_a\big(\frac{\max(0, W_M - W_m)}{H}\big)\Big) \tag{25}$$

where w_l, w_d, w_a represent respectively the weights for task load, difficulty and availability, and H is the maximum height of the search tree. Later in the text we will use an abuse of notation for the task cost related to an agent A_i, by the expression $\kappa(A_i)$. Note that all the factors in the equation (25) are normalized to the size of the problem (i.e. the number of decisional choices made at each branch of the tree).

This concept is also referred as *task cost* in the following Sections.

5.2 The multi-thread model

The multi-thread model adopted in our parallel solution, relies on two types of threads: workers, and schedulers. For every agent $A_k \in \mathscr{A}$, we indicate with $W_k(S_k)$ its *worker* (*scheduler*) component.

5.2.1 Worker

In the FIASCO multi-thread model, each *worker* thread is in charge of exploring the portion of the search tree to it assigned. Based on the nature of the task, when a *worker* resumes its search exploration, one of the following scenario may occur:

1. If a *light task* is selected, the *worker* simply starts the search procedure (described in Section 4) from the level rooting the node ending the path described by the selected task.

2. If an *heavy task* is selected, before being able to start the exploration of the subtree, the *worker* needs to employ a task reconstruction procedure (Algorithm 5) to explicit the labeling choices expressed in the task description.

The overall structure of the path reconstruction procedure is described in Algorithm 5. Given an heavy task $T \in Q_H$, the algorithm selects every pair of variables and associated values (V, v) from the variable list $T.V_{LIST}$, stored in the task description (line 6). The variable type (point, fragment or pair) is identified by

additional information carried in the task representation. The values for the selected variable are assigned without requiring bounds consistency checks (line 7), since the path being explored is fruit of an earlier investigation of some constraint solver in $\mathscr{A}$. It follows that the instantiation of each value is local consistent among all the choices of the path T.

In line 8, the constraint associated to the instantiated variables are silently activated. In other words, the constraints c involving the variable V are added to the constraint store Q, by taking into account that there is no need to verify the presence of a support for c in the domain of every variable it involves—the constraint propagation is delayed.

The condition of line 2, ensures the path described by the task T to be constructed; when such condition is verified, the rule iteration step is performed—by the AC-3 algorithm discussed in Procedure 3—ensuring the propagation of all the awaken constraints and the correct initialization of the constrained search space. A call to the search routine (line 4) resumes the *worker* exploration on the subtree described by the received task.

After the search exploration is resumed, we guarantee the presence of some light tasks in Q_L, unless the subtree being explored results to be too small—i.e. the level rooting such subtree is deeper then the threshold $\Theta_{\texttt{max\_expand}}$.

This light task queue population is efficiently handled by updating the worker level (l_w, introduced in Section 5.1.3.1). Figure 5.3 shows how this proce-

Algorithm 5: FIASCO's path reconstruction procedure.

```
1 procedure: path_reconstruction(task T);
2 if T.V_LIST = ∅ then
3 |   AC-3(𝒞, T.V_LIST);
4 |   return search(𝒞, F_LIST);
5 select (Var, val) from T.V_LIST;
6 instantiate values val on Var;
7 Q ← Q ∪ {(V, c) | c ∈ 𝒞, V ∈ T.V_LIST} T.V_LIST ← T.V_LIST ∖ {Var};
  F_LIST ← F_LIST ∖ {Var};
8 path_reconstruction(T);
```

dure is executed when a new (heavy) task T is started. Before going ahead with the exploration of the subtree described by T, the state of l_w is increased of a value parametric to the position of *worker* in the tree after the task reconstruction. The value of l_s (scheduler level) points at the root of the subtree of T, and denotes the position from which an agent could give away some task to satisfy incoming requests. In terms of computational complexity, the effect of generating light tasks, and pushing them to Q_L, is executed in $O(1)$. Note that while the horizontal

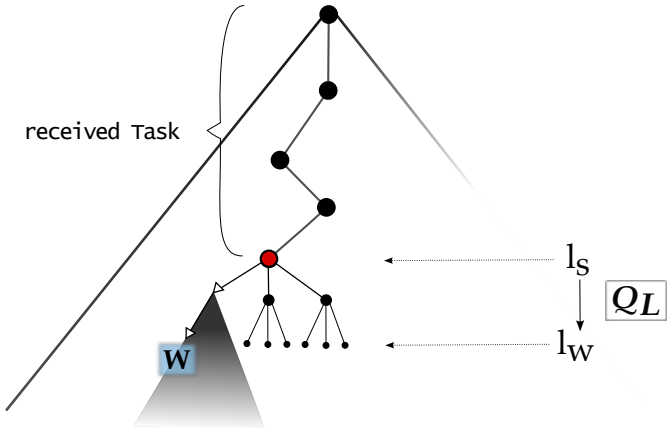

Figure 5.3: Light task queue population.

exploration of the search tree advances, the light tasks in Q_L are consumed by the *worker*. In the general picture *scheduler* and *worker* compete to acquire light tasks, in case a task request must be satisfied and $Q_H = \emptyset$. Therefore all accesses to the light task queue are performed in mutual exclusion.

A *worker thread*, other than complete the exploration of the search space assigned, is in charged to update the *cost difficulty* component used in the cost calculation—see Equation (25).

This process is carried out by collecting information about the values W_M and W_m (defined in Section 5.1.3.1) at a constant interval rate (set at 1000 nodes exploration). In addition, for every $K > 0$ pairs of values collected, the average moving averages expressed by the (22) and (23) are computed and their values made available for cost calculation.

The evaluation for the quality of a task, that includes the computation of other two components (*load* and *availability*), is performed by the *scheduler thread* to do not overload the *worker* with unnecessary computations.

5.2.2 Scheduler

The use of a fully decentralized schema, and the consequential adoption of a dynamic task rescheduling among the agents, arises the necessity of a robust and efficient communication system. The communication among agents lie on a protocol that is managed by the scheduler.

A general outline of the scheduler is given in Algorithm 6. The procedure handles incoming communications asynchronously, and processes them according to the type of message and the actual status of the worker (idle–jobless/busy).

Algorithm 6: A general overview of the scheduler algorithm

```
1  while ¬ global termination do
2      if worker.status=idle ∧ (Q_H ≠ ∅) then
3          Q_H ← Q_H ∖ T;
4          reactivate the worker on task T;
5      decode incoming message (if any) according to their msg.type;
6      if STATINFO timer elapsed then
7          compute the task cost and update the statinfo table;
8          send msg.type=statinfo to next agent in the ring;
9      if worker.status=idle then
10         if global_termination then
11             collect local best-k solution and send msg.terminate to
               next Agent in the ring;
12     if Msg buffer queue M ≠ ∅ then
13         process message according to msg.type;
14 return global best-k solution(s);
```

Among other tasks the scheduler is in charge of fill the (heavy) task queue by collecting tasks from other agents in $\mathscr{A}$, and reactivate a *worker* exploration. The information on the status of exploration of the search space of the current worker are reticulated among the agents, to help the scheduler in the load balancing strategy. Termination is guaranteed by a suitably modified *Dijkstra termination detection* algorithm in a token-ring fashion [Mis83, Mat87, DS80]. Upon termination, the $k-$best solutions found by every agent are circulated in a ring model among the agents in $\mathscr{A}$, and processed, in turn, to return the global k-best solutions. A more

detailed discussion on the communication schema and the scheduling procedures is treaded in the next Section.

5.3 Scheduling and Communication

Cluster-level parallelism is exploited allowing concurrent constraint solvers to work on different branches of the search tree. Due to the irregularity of the structure of the tree, a dynamic load balancing result necessary to ensure high efficiency in relocating agents from low quality area of the search space. A dynamic rescheduling of tasks among agents, involve an intensive communication system to exchange the information necessary to make the best decision about the worker relocation.

In the multithread model, an additional level of communication is required in order to allow worker and scheduler thread to exchange details about their local states.

In the next sections we will discuss the communication details and the scheduling techniques used in both level of parallelism.

5.3.1 Thread communication protocol.

In this section we discuss the protocol for the communication between worker and scheduler. The model adopted makes use of variables, local to the scope of the agent, to allow the exchange of information between the two parts. The access to these shared variables is regulated with the classical mutual exclu-

sion access formalism. The amount of mutual exclusion tests performed in such communications has been carefully calibrated.

According to the status of the exploration of the search space, we label a worker as `idle`, if it is not currently involved in the process of the search space exploration, and $Q_L \cup Q_H = \emptyset$; a worker is `jobless` if it is likely, according to the task cost evaluation, that the search space exploration will impending terminate and therefore $Q_L = \emptyset$; finally a worker is said `busy` if it is actively exploring some branch of the search tree and it is none of the other states.

The knowledge about the worker state allow the scheduler to undertake specific actions to guide the overall process of the global search space exploration employing targeted cluster-based communication.

The definition of a worker state `jobless/idle` is twofold: on one side it activate the scheduler in querying a task requests, and on the other side it report the worker unavailability to generate new task for incoming requests. The worker state active, allow the scheduler to employ a sub-task generation process, to fulfill incoming task requests. A discussion on the sub-task generation and load balancing is provided in Section 5.4.

In a multi-core environment, this protocol allows high modularity: multiple workers can be controlled by a single scheduler. According to such considerations, FIASCO can be adapted to best fit different hardware specifications.

5.3.2 Cluster level parallelization

In order to exploit an efficient parallelization at cluster level, we organize agents in a hybrid topology by identifying two classes of communications:

- *token-based* communications uses messages adopting a ring topology, where data is transmitted sequentially from one ring agent to the next one;
- *arbitrary* communication do not constrain the message traffic. Data can flow within any pair of agents in the pool.

For the first class of messages a control token mechanism is necessary to restrict the access to the media. In this framework the action of sending a message is regulated by the token passing strategy. Only agents holding the control token can send the associated message.

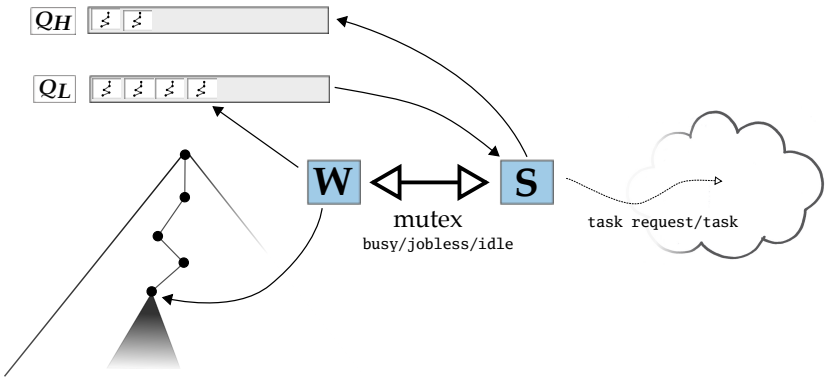

Figure 5.4: Thread and cluster communication within an agent

The cost of message passing involves the use of an agent internal resources as well as the occupation of the communication channel. In order to reduce the

message traffic in the system during the execution of the parallel problem, we carefully design a convenient communication protocol and an efficient communication strategy that uses messages to exchange information and requests among agents.

Figure 5.3.2 depict an overview of the two level communication protocol introduced above. Thread based communication acts locally to an agent A, to guarantee updated information between its worker W and scheduler S , while the cluster based communication allow message exchange among agents to exchange local information and manage task requests. In the following Section we describe the protocol used for an efficient message handling.

5.3.2.1 Messages

Our parallel system uses 6 type of messages to exploit various type of communications. We provide here a brief description of the message type with respect their content and use.

- `statinfo msg`. A message of type `statinfo` contains the *Statistical Information Table* (see Section 5.4.2). Such messages are reticulated among the agents arranged into a ring topology, with the purpose of exchanging information on the exploration status of the search space. The perceived knowledge about the global status of the tree exploration helps, in last analysis, in selecting the more convenient agent to contact for a task request,

when relocation is required.

- `terminate msg`. A message of type `terminate` is sent in a ring fashion mode whenever the global termination conditions have been detected (the termination conditions are discussed in the following Section). Prior sending a termination message, an agent collects the top$-k$ conformations generated by its associated worker, and encode them into the message data.

  An agent receiving a termination message, selects the best$-k$ solution from the comparison of the conformation received and its own best$-k$ solutions, and forward the termination message into the ring. Successively the agent can safetely terminate its execution.

- `task_req msg`. A message of type `task_req` is sent from an agent A_i to an agent A_j whenever the agent A_i needs to relocate its worker into an area of the tree being explored by the agent A_j. The selection of the tree branch, where to attempt to relocate the agent A_i, is performed by taking into account the perceived[3] quality of the subtree being consumed by the agent A_j.

- `task msg`. A message of type `task` is generated in response of a task request, if the current agent is able to produce a subtask to serve the requesting agent

[3]The word "preceived" here, is used to stress that the information an agent has access to, might not be up to date.

A_i.

A message of type `task_request` contains an heavy task, that will be used by the receiving agent to reconstruct the path from the root node to the node rooting such task, in order to relocate its worker in the desired area of the search tree.

- `no_task msg`. A message of type `no_task` is sent in response to a `task_req` message, whenever the receiver agent is not able to generate a new task to fulfill the request (i.e. its the worker status is either `idle` or `jobless`).

- `ttoken msg`. A message of type `ttoken` is sent to control the actions related to the reticulation of the termination messages. A `ttoken` message contains the token number and color, and notifies the receiving agent about its state of termination token holder. Such messages are sent from token holder agent to the next agent in a ring fashion, whenever the local termination property is satisfied (see section 5.3.2.2).

5.3.2.2 Termination detection.

Recall that FIASCO's parallel framework is implemented on a pure decentralized system, therefore the termination of the parallel execution needs a careful treatment. The termination detection scheme used in this work is a variation of the standard token wave Dijkstra algorithm [DS80]. Let us introduce the following definitions.

Definition 5.3.1 (Local termination property)**.** An agent A_k satisfies the local termination property if and only if:

$$\forall i \in \{1, \ldots, n\} \ \texttt{status}_k(W_i) = \texttt{idle}. \tag{26}$$

where $\texttt{status}_k(W_i)$ denotes a function that returns the believed W_i status according to the agent A_k observations in its Statistical Information Table. That is, the local termination for agent A_i is detected whenever no task requests can be employed according to the current agent knowledge about the state of $\mathscr{A}$.

Definition 5.3.2 (Global termination property)**.** The global termination property is satisfied if and only if:

$$\forall i \in \{1, \ldots, n\} \ \texttt{status}(W_i) = \texttt{idle}. \tag{27}$$

Note that the local termination is a local (w.r.t. the agent) property, i.e.

$$local\_termination(A_i) \not\Rightarrow (\forall A_j \in \mathscr{A} \, local\_termination(A_j)),$$

whilst, the global termination is satisfied globally by the agents of the pool:

$$global\_termination(A_i) \Rightarrow \big(\forall A_j \in \mathscr{A} \ global\_termination(A_j)\big).$$

The general outline of the algorithm used to detect termination among agents, is showed in Algorithm 7 that is part of the scheduler procedure (outlined in Algorithm 6). We assume that the algorithm is running on agent A_k.

Every agent in $\mathscr{A}$ is colored either to black or white. Agents who satisfy the local termination property are colored to white, conversely are colored to black.

At the beginning a token is assigned the value 0 (`ttoken.num`) and every agent is colored to black.

When an agent detects the local termination property (Line 3), it becomes white (Line 4) and it generate a white token with value 1 (line 9). The token is passed in the ring and its value incremented any time a process receiving the token satisfies the local termination property. Note that more that one token can be generated in the ring, therefore our policy imposes an agent to accept a termination token only if has value greater then the previous accepted one (Line 5). This guarantees that only one valid token reticulate among the agents. Since the information contained in the statistical information table might not faithful reflect the actual global state of the agents pool, an agent satisfying the local termination property can be reawakened by receiving new more updated information. To take account of this condition, a dual-pass ring termination strategy is employed. If a white agent receives a white token containing value $2n$ it can detect the global termination property (Line 12–14). At this stage the agent sends a termination message, that is reticulated in ring two times: the first time, the message has the effect of void any other termination token present in the ring (Line 16), and the second time it trigger the receiving agent to collect the results produced by the worker (encoded in the termination message) and forward the termination message into the ring (Line 20–22). If more then one termination message is generated at the first iteration, we consider valid the one generated by the agent

with the smallest id (Line 17–18). After forwarding the solutions to the next process in the ring an agent can safety terminate. The set of best solutions is returned by the last agent alive (see Algorithm 6).

Algorithm 7: Termination detection algorithm for agent A_k

```
1  local_termination ← 1;
2  if (∀ W_i ∈ 𝒜) status_k(W_i) = idle then
3  |   local_termination ← 1;
4  |   W_i.color ← white;
5  if recv(ttoken.num) > ttoken.num then
6  |   ttoken.num ← recv(ttoken.num);
7  if local_termination ∧ (ttoken.id = 0 ∨ recv(ttoken.id) = k) then
8  |   if ttoken.color = white then ttoken.num ← ttoken.num + 1 ;
9  |   ttoken.color ← white ;
10 |   send msg.type = ttoken to A_{k+1};
11 if W_k.color = black then ttoken.color = black;
12 if recv(ttoken.num) = 2n then
13 |   term_no ← 1 ;
14 |   send msg.type = terminate_1 to A_{k+1};
15 if (recv(terminate_1.Id) then
16 |   void ttoken;
17 |   if recv(terminate_1.Id) > term_Id then
18 |   |   void recv(terminate_1);
19 |   else
20 |   |   if recv(terminate_1.term_no) < n then
21 |   |   |   term_no ← term_no + 1;
22 |   |   |   send msg.type = terminate_1 to A_{k+1};
23 if (recv(terminate_1) ∧ term_no = n) ∨ (recv(terminate_2) then
24 |   collect best-k solutions;
25 |   send msg.type = terminate_2 to A_{k+1};
26 |   global_termination ← 1;
```

5.4 Load balancing

In this section we discuss the main techniques used to produce an efficient load balancing in the parallel system outlined above.

5.4.1 Partitioning the search space for the initial task

Consider a set of n agents $A_1, \ldots, A_n \in \mathscr{A}$ available for a concurrent exploration of a search space S. In order to correctly initialize the agents in $\mathscr{A}$, assigning them a specific portion of the search space, we require the satisfaction of the following properties:

1. No overlapping task shall be produced.

2. Every leaf of the search tree shall be reachable by some agent in $\mathscr{A}$.

Note that the two proprieties above can, more succinctly, be described in terms of partitioning the search space S into a set of sets of tasks $\mathbb{T} = \{\vec{T_1}, \ldots, \vec{T_n}\}$. Noting that $|\mathbb{T}| = |\mathscr{A}|$, hence it is possible to define a bijection $p : \mathscr{A} \to \mathbb{T}$, that assigns each agent to a specific set of tasks. By definition of partition, $\bigcap_{i=1}^{n} T_i = \emptyset$ and $\bigcup_{i=1}^{n} T_i = S$; it follows that the properties (1) and (2), can be implicitly guaranteed prior existence of such a function p. In last analysis, we can reduce the former problem to the problem of finding a bijection p to map agents to tasks.

Since a task is represented by a path leading from the root node of the search tree to the node rooting the subtree to explore, the function p can be

defined by devising a parallel, coordinated exploration of the tree to lead each agent with a unique set $\vec{T_i} \in \mathbb{T}$. Note that an enumeration for the set of agents and the set of nodes in the tree is straightforward. It follows that the existence of a bijection from $\mathscr{A}$ to $\mathbb{T}$ guarantees that the process of partitioning the search space among agents can be performed with no need of communication.

The initial exploration of the search tree is executed concurrently by every agents in $\mathscr{A}$ following a modified depth-first search. In this phase, we label each node with "$[a, b]$", with $a, b \in \mathbb{N}$ and $a \leq b$. Such label denotes the set of agents $A_a, \ldots, A_b$ that need to be initialized with some task (prior starting the search exploration). Such tasks will be generated by recursively expanding the current node.

At the beginning the entire set of agents is mapped into the search tree root node, that has label $[1, n]$. As the search exploration unfolds, the agents are equally repartitioned among the nodes of the tree. The expansion of a node u with label $[a, b]$ produces nodes $v_1, \ldots, v_k$, and the set of agents with id's in $[a, b]$ is repartitioned in k nodes as follow:

i. If $k \leq (b-a)$ (more agents than nodes are found) each agent is mapped into a node v_i by assigning $\lfloor \frac{(b-a)}{k} \rfloor$ agents to every node v_i and the remaining agents $L \equiv [(b-a) \mod k]$ nodes to the first $v_1, \ldots, v_L$ nodes.

ii. If $k > (b-a)$ (fewer agents then nodes are found) the set of tasks is dis-

tributed equally among the agents by assigning $\lfloor \frac{k}{(b-a)} \rfloor$ task to each node, and the remaining $L \equiv [k \mod (b-a)]$ tasks to the first $A_a, \ldots, A_{a+L}$ agents.

At the end of this parallel step every agent is mapped with a distinct set of nodes[4], and at the same time it is guaranteed that the assigned tasks have the shortest possible path from the root node—i.e. they are potentially the largest tasks that may have been generated by such exploration. Note, moreover that this procedure generate only light tasks, and as soon as this partitioning procedure terminates every agent is ready to begin the tasks exploration.

Figure 5.5 illustrates an example of the parallel partitioning procedure where $|\mathscr{A}| = 10$. The root node has label $[1, 10]$ to indicate that all 10 agents will share such node in the initial task set. The DFS is executed by all the agents in parallel; after the first expansion, case (i) applies—the number of agents is bigger the number of available nodes. From left to right, agents $A_1 \ldots, A_4$, $A_5 \ldots, A_7$ and $A_8 \ldots, A_{10}$ will be sharing the same path. Expanding the children nodes, produces an assignment of tasks to the agents light task queue. In particular a single task is given to all the Agents but A_5 and A_8, which will be initialized with two tasks (the ones ending at nodes $[5, 5]$ and $[8, 8]$ respectively) according to case (ii).

[4]The set of tasks initially assigned to an agent is represented by siblings nodes

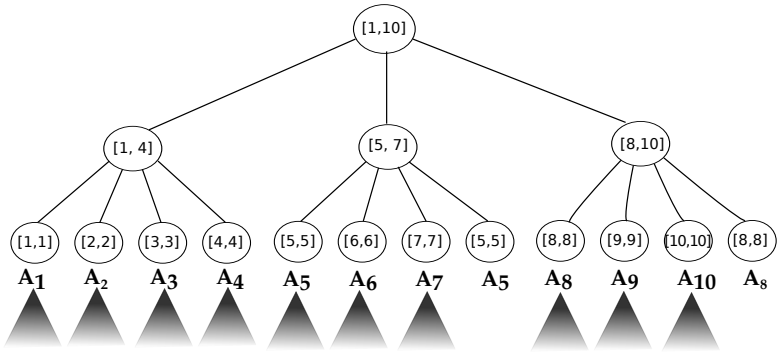

Figure 5.5: Parallel partitioning of the search space

5.4.2 Statistical Information Table

Exploring a search space in a distributed and fully decentralized environment involve a significant amount of communication to ensure a correct load balancing. Indeed, without a clear knowledge of an arbitrary agent search exploration state, many task request might be addressed to agents that are exploring a poor area of the search tree and therefore cannot fulfill the incoming requests. To reduce the amount of superfluous communication we introduce the concept of a *Statistical Information*.

The idea is that partial information about the global search exploration state can direct agents to commit better choices in terms of selection process for a task request. Intuitively, when relocation is necessary, a good load balancing strategy would seek to relocate the considered agent in a promiscuous zone of the search space, where some other agent is actively involved. This description is mainly captured by the cost function of a task introduced in Section 5.1.3.1.

The *statistical information* are held by each agent A_k in the form of table (referred as T^k) constituted of n rows. Each row $T^k(i)$ is associated to the agent $A_i \in \mathscr{A}$, and contains the information necessary to exploit the load blanching strategy. Every row of the table $T^k(i)$, carries a flag (`idle`) referred to the worker W_i's exploration state; an identification number (`id`) of agent A_k; the id (`succ_id`) of the agent that A_k expect to contact for the next task request (whenever a relocation results necessary); the `cost` evaluating the A_k's task quality, according to equation (25); a termination code (`term_code`) used to detect the termination properties. In addition, each $T^k(i)$ contains a list of the A_i's most recent agents contacted for a task request, referred as the `window_history`, and the set of agents (`agents_to_serve`) that has decided to contact agent A_k whenever they need to relocate their associated worker in some other area of the search tree.

The *Statistical Information* table is circulated as a token among the agents (organized as a ring for this purpose) through a message of type `statinfo`. A `statinfo` message requires the cost of current task exploration to be up to date. When an agent receives a `STATINFO` table, it updates its own row, and forwards it into the ring. The whole process is regulated by a timer to prevent floating the channel with a big amount of `statinfo` messages. Once an agent receives a `statinfo` message it starts a countdown, set at the value T^k.`timer`, contained in the table. At the end of the countdown, the table is forwarded to the next agent of the ring.

The rate at which the information are circulated is parametric to the search space exploration status of the agents in $\mathscr{A}$. The idea is motivated by the observation that when the parallel search starts most of the workers will be involved in solving some tasks, and, in general, agent relocation is not expected. It follows that a small amount of information is sufficient to handle this phase. As the exploration of the search space unrolls, the active tasks will implicitly become smaller; in this phase, agents relocation is expected at a higher frequency. To take account these variations, we dynamically update the rate at which the statistical information are circulated in the ring. At the beginning of the concurrent search, the value of the `T.timer` is fixed to $\frac{1}{n}$ seconds (with $n = |\mathscr{A}|$). During the parallel process, any time an agent, willing to relocate, receives two consecutive messages of type `no_task`, it will decrease the value of the T`.timer` of a constant factor ϵ_t. This choice is motivated from the fact that the agents to contact for task requests are selected according to the information contained in the Statistical Information table. When two consecutive negative replies are received, it can be supposed that the statistical information, used for the agent selection for task requests, are out of date. When an agent receive the Statistical Information table, it will select as countdown the lowest value between its internal countdown and T`.timer`. This strategy ensures the reactivity of the system following the evolution of the space exploration. Note that it is sufficient one reticulation of the Statistical Information tables to update the rate at which the information are spread among the

agents.

Since the termination information is also carried in such tables, at the end of the concurrent search, only negative answers can be generated to any task request. Therefore the T.`timer` will be rapidly decremented to its lowest limit (set as the estimated time necessary to compute the task cost information[5].

5.4.3 A load balancing strategy

In a complete decentralized system an efficient task rescheduling becomes necessary to balance out the irregularity of the search tree. The concept introduced in the previous Sections allows to exploit a rened load balancing strategy. Two key features that such strategy guarantees are: to promptly relocate agents that terminate a local exploration, and to minimize the number of communications among agents. These two features are in often in contrast: a reasonable assumption, in a decentralized system, is that the more information is available to the agent, the more refined would be the relocation of the requesting agent.

In last analysis the load balancing problem can be expressed as the problem of finding the best position of the tree where to relocate a requiring agent, so to globally guarantee the minimization of the overall agent's idle time. We use the Statistical Information to seek this goal, and in particular we use the qualitative task description of the active agents, to select the possible agent to contact for a

[5]Recall that the task cost take account of the calculation of an average moving average described in section 5.1.3.1

task request.

In the hereinafter discussion, we use the following notation. Let T^k be a *Statinfo Table* held by agent A_k (that identifies the agent to relocate). We use the notation $T^k(i)$ to indicate the information relative to the agent A_i, held by the *Statinfo Table* of A_k.

The preference order, w.r.t. the best agent to contact for a task request, is expressed by sorting the $T^k(i)_{i\in\{1,\ldots,n\}}$ for each agent $A_k \in \mathscr{A}$, in a descending order according with respect to the task cost. During the sorting process, we also take account the agent *neighborhood factor*. Consider the case when A_k's relocation is needed and the cost of two candidate choices for the next task request, A_i, A_j, differ of some $\epsilon > 0$. The agent in the closest A_k's ring topology neighborhood will be preferred for a task request. To illustrate the rationale behind this choice, assume that the workload of every agent in the pool is highly balanced, and the statistical information are traveling fast allow sharing similar statistical data. In a such scenario, if a set of agents require relocation, only the top ranked agent (or a small subset of top ranked agents) will be selected to fulfill such requests. As a result, many agents will attempt to contact the same small set of the selected agents. To avoid such a bottleneck, the *neighborhood factor*, helps in taking into account the requiring agent's neighbors, by assigning them a higher priority when a the cost function is computed—the neighborhood priority decreases linearly with the neighbor distance. This formalism is expressed by the

following: $(\forall A_i \in \mathscr{A})(i \neq k)$,

$$\kappa(A_k) = w_1\big(\kappa(A_k)\big) + w_2\big(\frac{1}{|k-i|+1}\big) \tag{28}$$

where the $\kappa(A_k)$ is referred to the equation (25), and w_1, w_2 are weights associated to the task cost and neighborhood factor.

Whenever an agent receive a message of type `statinfo` the *Statinfo Table* is updated and the successor function selects the next process to contact for a task request. Note that this step is completed before the table is forwarded back into the ring.

Prior describing the strategy used to selected the "best" agent to contact in case of relocation, we need to introduce few definition.

Definition 5.4.1 (Successor)**.** A successor function δ is a mapping from the set of agents to the set of agents, $\delta : \mathscr{A} \to \mathscr{A}$, that assigns every agent to the "best" agent to contact in case of task request.

Observe that the successor function is not surjective: an agent A_k can be the successor of more then one agent in $\mathscr{A}$. This concept is expressed by the following definition:

Definition 5.4.2 (ATS)**.** Given an agent $A_k \in \mathscr{A}$, the set of agents $A_i \in \mathscr{A}$ s.t. $\delta(A_i) = A_k$ is defined the set of the set of the Agents To be Served by A_k:

$$ATS(A_i) = \{A_i \mid \delta(A_k) = A_i\}$$

We introduce two properties that the successor function should satisfies:

Definition 5.4.3 (Anti-reflexive property)**.** The successor of an agent A_i cannot be A_i:

$$\forall A_i \in \mathscr{A},\ \delta(A_i) \neq A_i$$

Definition 5.4.4 (Anti-symmetric property)**.** The successor of the successor of an agent A_i shall not be A_i. Or equivalently, if the successor of an agent A_i is A_j, hence A_j's successor shall be different from A_i:

$$\forall A_i, A_j \in \mathscr{A},\ (\delta(A_i) = A_j) \implies (\delta(A_j) \neq A_i)$$

The latter property can be relaxed in case no other agent choice results feasible in $\mathscr{A}$ (e.g. when $|\mathscr{A}| = 2$).

The successor function is computed according to the Algorithm 8. The procedure refers to the cost calculation for agent A_k and requires the T^k to be sorted in descending order with respect to the $\kappa(A_k)$. The default successor is chosen by selecting the first element of the sorted T^k that is not already present in the A_k's window history (Line 1). The while loop (Lines 4–16) is aimed at checking the satisfaction of the following conditions over the chosen successor agent:

i. The antireflective and antisymmetric properties for the successor of A_k are verified in Lines 5 and 7 respectively.

ii. The conditions in Line 10 ensures the current agent A_k not to be in the ATS's set of the successor $\delta(A_k)$ selected, and that the size of the agent to serve set of $\delta(A_k)$ to be smaller then a threshold Θ_{ATS}.

iii. In Line 12 we ensure that the $\delta(A_k)$ is not among the last $L-$agents selected as successor by A_k in the previous request.

Note that if no feasible candidate can be found in the whole $\mathscr{A}$, the antisymmetric property is relaxed, and properties (*(ii)* and *(iii)* dropped)—the first agent that was not satisfying the antisymmetric property is selected (Line 9).

Observe, moreover, that the order to which these conditions are verified reflect their importance in terms of failure in achieving a valid successor.

If conditions (*i, ii, iii*) are satisfied the algorithm updates the selected process in the statinfo table T^k, together with its window history and the ATS set (Lines 22–23). In case no the agents in $\mathscr{A}$ is able to fulfill these requirement, the first agent with the highest cost not in the window history or not having A_k as successor if returned.

It is important to note that, the key principle of this load balancing strategy is given by the information carried in the Statistical Information Tables. Therefore, we take advantage of any communication between two agents, A_s (sender) and A_r (receiver), to send updated information (of the sender Statistical Information). Whenever a communication message needs to be employed, (for a task

Algorithm 8: Successor function algorithm for agent A_k

Data: T^k sorted in descending order w.r.t. the cost function

```
1  sel ← min_i{i | A_i ∉ T^k(k).wh};
2  sel_fail ← sel;
3  check ← 0; count ← 1;
4  while check ∨ count ≥ |𝒜| do
5      if k = sel then
6          sel ← sel + 1 mod |𝒜|;
7      else if k = δ(T^k(sel)) then
8          sel ← sel + 1 mod |𝒜|;
9          sel_fail ← sel;
10     else if A_sel ∈ ATS(T^k(sel)) ∨ |ATS(T^k(sel))| ≥ Θ_ATS then
11         sel ← sel + 1 mod |𝒜|;
12     else if A_sel ∈ T^k(k).wh then
13         sel ← sel + 1 mod |𝒜|;
14     else
15         check ← 1;
16     count ← count + 1;
17 if count ≥ |𝒜| then
18     if k = sel then sel ← sel + 1 mod |𝒜|;
19     sel ← sel_fail;
20 else
21     T^k(k).succ ← sel;
22     T^k(k).wh ← T^k(k).wh ∪ {A_sel};
23     ATS(T^k(sel)) ← ATS(T^k(sel)) ∪ {A_k};
```

request, a task response, etc.), we append to the original message, the sender information $T^s(s)$. The receiving agent A_r updates its Statistical Information table by substituting the field $T^r(s)$ with the most updated received one.

The idea in this approach is motivated by the fact that the length of the messages is almost irrelevant, in terms of message-passing cost, when compared to the channel occupation cost while sending the message. For this reason we adopt the policy of always send the maximal amount of (updated) information.

5.4.4 Some Implementation details

FIASCO parallel system has been implemented on a traditional Beowulf cluster, using InfiniPath MPI and C++ under GCC 4.2.2. Each agent is implemented as an MPI process, and the thread protocol is implemented with the option MPI_THREAD_FUNNELED, to ensure that only the main thread (*scheduler*) will make MPI calls (all MPI calls are funneled to the main thread). FIASCO make uses of Open MPI, version 1.2.7, hence we do not allow a dynamic process handling.

Every agent is attributed an incremental id starting at 0 and the system topology, with respect to message passing, is an hybrid topology: all to all for task request, and ring based for Statistical Information passing and token handling.

All communications are handled asynchronously by the scheduler thread, allowing the computation to proceed, while waiting for messages from other agents, or handling the physical resources necessary to send and receive messages. Such protocol ensures the maximum performances, and reduce the idle intervals.

5.5 Experimental results

In this Section we present some experimental results targeted at testing the performances and the scalability of the parallel system. We run exhaustive searches according to the model described in Section 4.

In Section 5.5.1 we test the quality of the parallel system analyzing the load

balancing, when light work conditions and high fragmentation occurs. For this purpose we select a test of target proteins, for which a complete enumeration of the conformational space can be handled within a reasonable amount of time (in the order of few minutes). We test our system with up to 64 concurrent solvers. Scalability results are discussed in Section 5.5.2, where we test the system for those proteins for which, the time allowed in the sequential implementation was not sufficient to explore the complete conformational search space. We discuss the quality of the results and computational time gain, when compared to the sequential results. We refer to FIASCO_seq for the tests performed using the sequential implementation of FIASCO, and FIASCO_par, for the test performed over the parallel version of the solver.

We show that the parallelization of FIASCO, results to be effective, allowing us to tackle longer proteins (100–300 amino acids), and to improve the quality of the results—when compared to the sequential version—in the analysis of shorter targets.

Methods. The setting applied to the solver in order to conduct the experiments, are the same of those used to test the sequential version (described in Section 4.5.1).

For the task cost calculation (see Section 5.1.3.1) we found that good values for the load, difficulty and availability weights defined in Equation (25) are: $w_l =$

0.30, $w_d = 0.1$ and $w_a = 0.60$.

The neighboring factor, described by the Equation (28), has been set up with weights: $w_1 = 1$ and $w_2 = 2$.

We set the threshold $\Theta_{\texttt{max\_expand}} = \lfloor \frac{7}{10} H \rfloor$, where H denotes the height of the search tree. Recall that $\Theta_{\texttt{max\_expand}}$ indicates the maximum depth of the tree from which a task can be generated—by removing it from the light task queue of a working agent—to serve incoming requests.

For what concerns with the Statistical Information Tables, we set the size for the window history set (wh) and the agents to serve set (ATS) to 2, if the number of agents in $\mathscr{A}$ is greater then 8. Such information is relaxed otherwise.

The initial delay time an agent needs to wait in order to send a `statinfo` messages to the next agent of the pool, is set to be 0.001 seconds and the time increment, caused by receiving two consecutive negative responses at a task request, is: 0.00005 seconds.

5.5.1 Performances

In this Section we analyze the performances of the parallel system. We target our tests to evaluate the load balancing and the quality of the message passing strategy adopted.

For this experiment, we use a test set composed of four proteins, suitable modified in order to allow a complete enumeration of the conformation search

space in a small amount of time (in the order of 20–1000 seconds). The idea is to test the system in conditions of high fragmentation and load balancing and when light work is conduced by the agents in $\mathscr{A}$.

An analysis of concurrent solvers in bigger search spaces is treated in the next Section.

For each combination of protein and number of agents employed, we performed 20 runs and recorded the average values of computational time, idle time per process, number of task requests and number of successful rescheduling.

In Figure 5.6 we report the parallel speedup obtained using a number of agents ranging from 2 to 64. The black solid line represent the theoretical linear speedup. As it can be observed the reported speedup are close to the linear one, when a contained number of agents is used. The speedup slowly degrade as the number of the employed agents increases. This behavior is due to the condition of high fragmentation characterizing each test.

In another experiment we investigate in the amount of time spent by each agent in the process of relocating in a different branch of the search tree. The results are illustrated in Figure 5.7, where we plot in a logarithmic scale, the percentage of the average idle time per process. From such results it is evident that a very short time it is consumed in the process of rescheduling—considered as the interval marked by the instant a worker become idle to the instant in which it restart the search. These results confirm the strength of our rescheduling strategy.

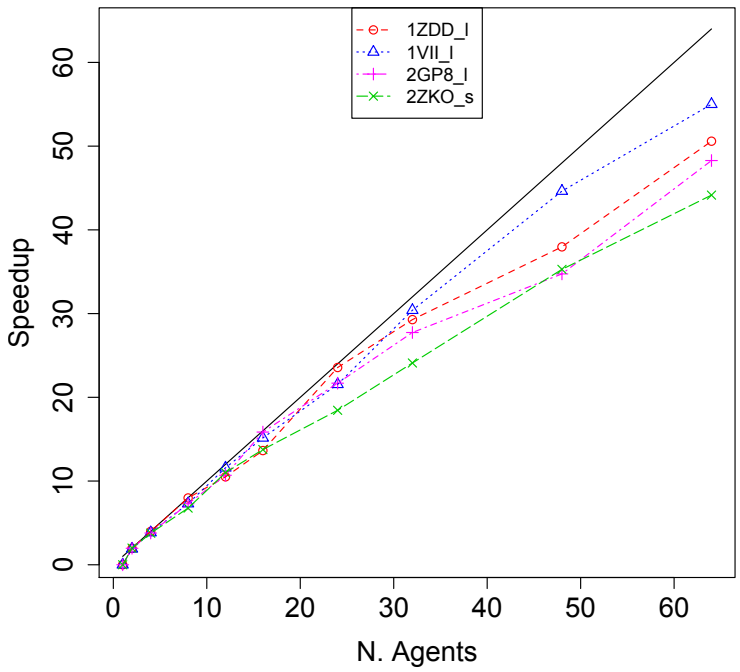

Figure 5.6: Parallel Speddup

We would like to stress that in the context of a concurrent exploration, a decentralized load balancing strategy is essential to smooth out the irregularity of the search tree. Indeed there is no knowledge about the size of each subtree in which a relocation could take place—the effects of the propagation are not predictable apriori.

In Table 5.1, for completeness, we report the detailed results obtained from the experiments discussed above. In the first column we describe the number of agents (#A) used in each computation, and for each protein we report the time (in seconds) recorded to exhaustively complete the exploration of the search space,

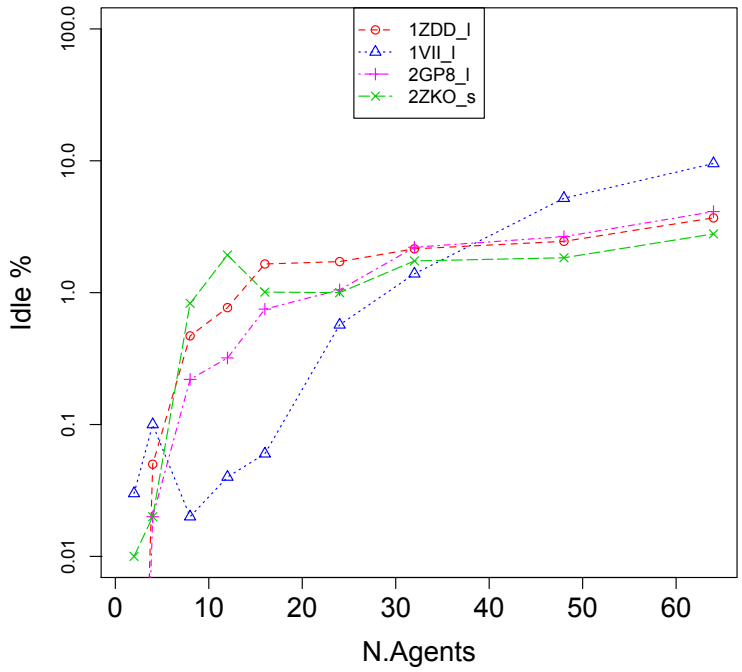

Figure 5.7: Parallel idle time

the percentage of idle time per agent (% Idle), the parallel speedup achieved (P.S.), the total number of task requests during the parallel computation (#T.R.) and the percentage of successful rescheduling (%S.R.)—defined as the ratio $\frac{\#msg.task}{\#msg.task\_req}$.

Note that, in average, each agent requires a relocation 4 to 7 times in each computation. In this context it is possible to appreciate the accuracy of our rescheduling strategy while selecting the right agents to contact for task requests. Observing the percentage of successful rescheduling (i.e. the number of task requests that have being fulfilled in the first reply), it is possible to note that the global status of the search exploration is well captured by the statistical

	1VII_s					2GP8_l				
#A	Time	% Idle	P.S.	#T.R.	%S.R	Time	% Idle	P.S.	#T.R.	%S.R
1	26.51	0	0	0	–	88.94	0	0	0	–
2	13.93	0.03	1.90	2	1.00	45.14	0.00	1.97	5	1.00
4	6.91	0.10	3.84	12	0.93	23.48	0.02	3.79	10	1.00
8	3.63	0.02	7.30	8	1.00	12.20	0.22	7.29	29	0.97
12	2.29	0.04	11.58	11	1.00	8.32	0.32	10.69	49	1.00
16	1.75	0.06	15.15	15	1.00	5.61	0.75	15.85	51	1.00
24	1.23	0.57	21.55	23	1.00	4.1	1.05	21.69	64	0.99
32	0.87	1.21	30.40	27	1.00	3.21	2.21	27.74	86	1.00
48	0.59	5.20	44.63	46	0.98	2.56	2.66	34.74	92	0.99
64	0.48	9.54	55.00	45	0.98	1.84	2.19	48.28	108	1.00
	1ZDD_l					2ZKO_s				
#A	Time	% Idle	P.S.	#T.R.	%S.R	Time	% Idle	P.S.	#T.R.	%S.R
1	91.12	0	0	0	–	1856	0	0	0	–
2	46.57	0.00	1.96	3	1.00	933.8	0.01	1.99	5	1.00
4	22.87	0.00	3.98	11	1.00	492.1	0.02	3.77	15	1.00
8	11.81	0.00	7.98	38	1.00	274.1	0.83	6.77	80	0.99
12	8.68	0.01	10.50	33	0.99	168.7	1.93	11.00	84	1.00
16	6.68	0.49	13.64	26	1.00	134.9	1.01	13.76	250	1.00
24	3.86	3.41	23.56	60	0.98	100.6	1.00	18.45	277	1.00
32	3.11	2.15	29.28	91	099	80.33	1.74	23.10	289	1.00
48	2.40	2.45	37.97	152	0.99	52.61	1.84	35.28	332	1.00
64	1.81	3.69	50.59	187	0.96	43.01	2.79	43.15	418	0.99

Table 5.1: Parallel experimental results

information reticulated among the agents.

From the above observations, it follows that the reasons for the speedup degradation, observed when the number of agents increases consistently, have to be investigated in the way a constraint solver relocation is handled, or in other words, in how efficiently a solver relocate to a specific branch of the tree in order to resume the search. This will be object of future investigations.

5.5.2 Scalability

The goal of this section is to analyze the scalability of our system. We analyze a subset of 6 target proteins from the test set used in section of 4.5.3 and 2 additional large proteins ($n > 100$). The test set is evaluated using 128 concurrent processes, and we impose a computation time limit of 2 hours. We collected the best-10 solution returned by each agent, and among these, we select the one that minimizes the RMSD measure.

In order to establish a comparison between FIASCO_seq and the FIASCO_par, we introduce an efficiency measure index (`gain`), defined as the ratio between the time necessary to generate the best solution in the sequential version, and the time spent by the parallel system in producing a solution of equal or better quality (in terms of RMSD).

Table 5.2 summarizes the experimental results collected from the test set, and it is divided in two sections: the first, is dedicated to the comparison of the parallel with the sequential version of the solver, while the second reports the best values generated in each run.

In the "First Improvement" section of Table 5.2 we report the RMSD, the energy value and the computational time associated to the first conformation generated that matches or exceeds the quality of the prediction given by the sequential test. Moreover, for every prediction we report the `gain` factor. In the "Best Solution"

Protein		First Improvement ($2h$) limit				Best Solution ($2h$) limit		
PID	N	RMSD	Energy	Time (s)	`gain`	RMSD	Energy	Time (s)
1E0M	37	6.14	-233.04	17.68	4,165	5.06	-275.05	4,216.7
1ENH	56	5.58	-264.71	419.5	13	5.06	-275.12	4,614.14
2IGD	60	7.09	-244.86	2.42	49,514	5.84	-310.82	305.97
1H40	69	6.92	-304.21	0.476	43,332	4.24	-383.71	559.75
1AK8	76	9.17	-268.14	17.32	6,602	9.17	-268.14	17.32
1DGN	89	5.76	-285.69	37.5	102.66	5.76	-285.69	37.5
						Best solution (1d) limit		
3L2A	129					3.78	-2,552.77	26,709
3EMN	295					2.65	-7,315.15	19,275

Table 5.2: Computational time gain and qualitative results

columns we describe the RMSD, the energy value and the computational time, associated to the best solution generated by the test. The last two columns of Table 5.2 reports the analysis of two large proteins for which it is given a one day computational time limit. In particular for the target 3EMN_X it has been performed an RMSD rather then energy optimization (refer to Section 6.1.1 for details).

Observe that the computational time improvements of FIASCO_par implementation w.r.t. the sequential one are promising: exploiting concurrency allows, in general, to generate solutions, that are qualitatively comparable with the ones generated in its sequential counterpart, gaining different order of magnitude in computational time.

It is interesting to note that, in the most of the cases, the speedup observed, are superior that a simple linear progression w.r.t. the number of concurrent agents used in the search. The reason for such a speedup needs to be

investigated analyzing the differences between the sequential and the concurrent behavior of the solvers with respect to the areas of the search tree explored during the computation.

Let's first focus on the behavior of the solver in FIASCO_seq. Given any order over the solutions space, the search space is exhaustively analyzed producing solutions, for the defined CPS, exploring the leftmost paths first (w.r.t. the given path enumeration). It follows that, in every local minima reached, the solver will generate many solutions that are not able to substantially improve the quality of the already generated conformations. On the other hand in the concurrent search, given enough agents, a good differentiation—w.r.t. the fragment used to assembly a conformation—of the explored solutions may be provided. It is important the note that different areas of the tree are associated with different degree of flexibility of specific conformation sub-structures. Moreover, according to the agent partitioning strategy (see Section 5.4.1) the concurrent solvers branches at the higher parts of the search tree; it follows that every time a new solution is generated by some agent in $\mathscr{A}$, there is a high chance that it differs from solutions generated by other agents with respect to its sub-structural composition.

Therefore, in the comparison of the two implementation, producing qualitative comparable results in an incomplete exploration of the search space, rewards the one able to generate the most different set of conformations.

Consider now the prediction for the case of target protein 1ENH. The

outcome of the parallel test execution results to be poor (approximately 13x) in terms of gain measure, when considering the number of agents used for the computation. We shall analyze the reasons behind such result. Note that the best solutions reported (both for FIASCO_seq and FIASCO_par) are generated in about only the 1% of the total execution time allowed. This implies that such conformations are located in roughly the first 1/100 portion of search space explored (assuming the absence of many fail nodes at higher rightmost levels of the tree).

It is likely that the RMSD landscape for such configurations, presents a consistent local minima that is reached in the earlier stages of the exploration—areas further to the left in the search tree. In this respect, considering the way we partitionate the search space, it is to be expected that the parallel and the sequential exploration would converge to such minima at similar phases. To confirm such hypothesis, we have tracked the exploration conducted by each agent in $\mathscr{A}$, and find out that the solution for the target 1ENH, reported in Table 5.2, is generated by an agent with low id (A_6)[6]; moreover when the solution was generated we recorded no task rescheduling for such agent. Such hypothesis was also confirmed by physically inspecting the sequence of decisional choices made by the agent generating such solution.

[6]Recall that in our search space partitioning strategy the smaller is the agent id, the (roughly) leftmost is the area of the tree it will be in charge to explore.

The last observation we would like to rise is related to the energetic values found among the solutions generated by the two implementation of FIASCO. In fact the energy values returned by the parallel exploration are always smaller then the one outputted by its sequential counterpart.

First, we shall take into account that the computational time limit imposed to two systems, is greatly different (2 days for the sequential version and 2 hours for the parallel exploraiton). In fact when sufficient time is given, the behavoiur of the search carried by the concurrent solvers will be similar, in terms of energy minimization, to the one carried by a sequential search (e.g. for the target 3L2A). However, it is interesting to note, that even when such time limit difference are taken into account, FIASCO_par was able to generate better conformations in terms of RMSD measure, associated to lower energy values, when compared to the ones generated by the sequential tests.

In order to understand such behavior we need to inspect the way the conformational search space is explored in the two versions of the system, focusing on the objective function evaluation. Remark that the system outputs a candidate solution, every time the energy value associated to the conformation found exceeds the best energy value recorder during the whole process. Optimizing the energy values, while proceeding in a leftmost exploration of the conformational search space, causes the number of solution generated per unit of time to drastically decrease. Clearly, while the search exploration proceeds, the optimization

function value becomes more and more refined, which implies that the probability of finding new conformations with better energy associated usually decreases. The implication here is that the energy minimization does not (exclusively) relies on a particular branch explored, but more on the chain of events that have lead the search in a particular status,

This behavior makes the structure comparisons unfear in a context in which it is difficult to describe structure characterization trough an energy function.

A possible direction of investigation would be to consider a more "targeted" use of the energy model. Different energy functions may be adopted for different structures of the protein—assuming that knowledge of such structured is given in input—or different energy weights could be proposed in various region of the protein. In our model, for example the torsional contributions weights may be lowered for the amino acids modeling coil regions (such us loops)

Another idea in which future investigation may be focusing, deal with optimizing protein sub-structure first with targeted energetic functions and hence focus on the whole conformation using a different optimization function to optimize the weaker interaction that stabilize the protein structure in its globally.

CHAPTER

SIX

ONGOING AND FUTURE WORKS

In this Chapter we present the current stage of the work in progress and propose different ideas and intuitions for the future directions.

In Section 6.1 we discuss the current work in progress and present preliminary results on two case of study. We apply our work to two sets of unknown proteins one associated to the *Ebola* virus and one derived from the study of the inner ear of the *Xenopous laevis*.

In Section 6.2 we propose different ideas aimed at achieving further computational speedup and at improving the quality of the predictions inquiring new constraint and propagators, energy-based branch and bounds, novel heuristics, and employ the use of specialized structure evaluation functions targeted at modeling characteristic protein regions.

6.1 Current works and Special studies

FIASCO is an ongoing work, and our current focus is in refining the protein representation model. The reduced backbone-centroid model is being replaced with a full atom backbone-centroid representation to predict hydrogen bonds with more accuracy and allow the use of a more refined energy function. Moreover, careful experiments on the fragment set in use are being performed. The aim of these tests is to provide concrete evidence about the structural coverage, at low RMSD threshold, provided using the current fragment set. We are also planning to reduce the approximations introduced by the amino acid partitioning introduced to map the 20 amino acid into 9 classes.

On another side, we are working on the refinement of the pair model. The idea is to pre-compute the possible interaction between any compatible pair of special fragment. The intuition is derived by the studies on super-secondary models[Sin05, KM97, OTdB07, CC98, MBHC95, CH02, BKV00]. It is believed, in fact, that at the earlier stages of the protein folding, when interactions within local shapes are stabilized, such structures tend to group together to form well recognized pattern, called super-secondary structures. We design a model for the helix-helix interactions and early results show interesting behavior in reaching local minima (in the RMSD minimization landscape) sharply faster. This implies that the placements of the blocks, modeled by the pair interactions, guide the

search avoiding many moves that are not biologically meaningful.

6.1.1 Cases of Study

In this section we report preliminary results on two case of study, in which we focus on the prediction of unknown proteins structures sequenced using microarrays, and in the prediction of suitable modified protein structures, targeted at drug design. The experiments are an ongoing work, and are conducted in close collaboration with experts in the fields of biology and computational biology; we tested our solver analyzing the *Reston Ebola Virus VP35 Interferon Inhibitory Domain* [LSF$^+$10] and the *voltage-dependent anion channel protein, Xenopus laevis inner ear* [OTT$^+$11a, UCC$^+$08b].

We show that our system is suitable to handle the cases in which specific homology or structural information on the targets is available.

Reston Ebola Virus VP35 Interferon Inhibitory Domain Ebola viruses causes lethal hemorrhagic fever in primates (including humans). The *Reston Ebola virus* (REBOV) is the only known Ebola virus that is nonpathogenic to humans[GJ03]. The VP35 protein, is an inhibitory domain, critical for immune suppression, and in [LSF$^+$10] Leung et al. characterized such domain observing minor differences from the lethal ZEBOV Ebola virus—overall backbone RMSD of 0.64 Å[LSF$^+$10].

Taking into account the structural similarities of the two proteins, and the low

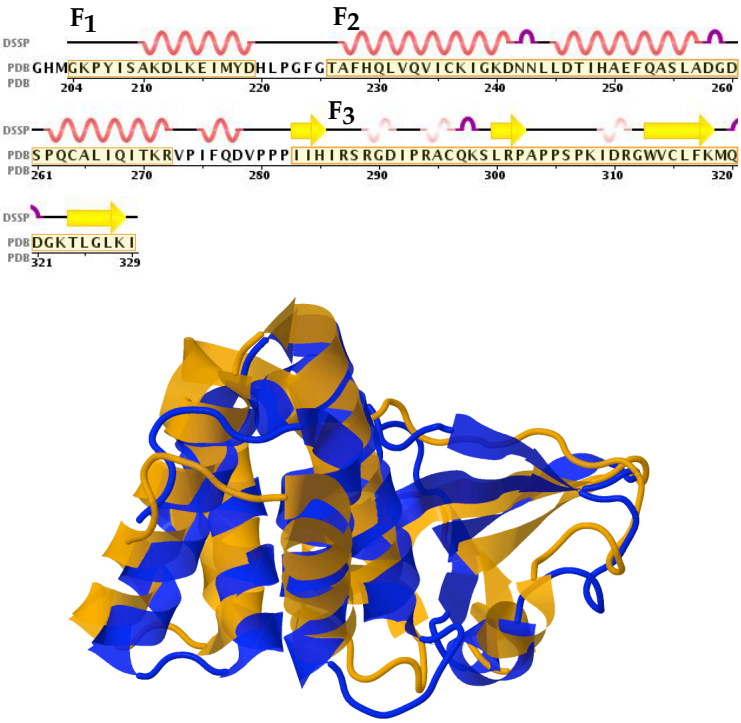

Figure 6.1: REBOV VP35 prediction. The main geometrical user defined constraints (top) and structure prediction (bottom). The native structure is displayed in blue (darker) and predicted structure in orange (lighter).

tolerance for sequence variability, the study of the REBOV VP35 IID protein is highly important in the domain of drug design.

We provide a reconstruction of the REBOV VP35 protein (3L2A), showing that our tool is suitable at exploiting predictions where sub-structures may be arbitrarily substituted with ad-hoc built geometrical conformation, or using small structures form the Assembly-db. Figure 6.1 shows the block structure (top) given in input as special fragment for the protein structure analysis. The figure shows the primary sequence of the protein 3L2A, together with the secondary structure prediction. The special fragments are indicated as F_1, F_2, F_3, and processed as

a set of geometrical constraints. Note that in the constraint modeling phase we can emphasize those protein regions in which higher variability (characterizing differences between REBOV and ZEBOV) is desired. For example we can allow point mutations in loop regions to observe structural differences from the set of solutions so generated.

In Figure 6.1 (bottom) we report the best prediction produced, in terms of energy minimization, superimposed to the native structure. The prediction error is within 3.78Å, and it is obtained using 128 cores in a computation of 24-hours computation. The prediction can be further improved by the use of Molecular Dynamics refinement as shown in [DDFP11]. In the future works we plan to extend the protein representation model to a full atom backbone, to be able to generate more refined solutions and use a more precise energy functions.

Note that FIASCO is inherently able to generate different structures satisfying the required constraints imposed on the protein model. This ability makes it highly suitable to explore possible conformational variations in those regions believed to control the inhibitory activity. This characteristic makes FIASCO a suitable tool for biomedical investigations, related to drug design.

3EMN_X: voltage-dependent anion channel protein, Xenopus laevis inner ear. In this section we present preliminary results of our effort in adapting the FIASCO solver for the analysis of membrane proteins essential for in-

ner ear functions. The inner ear is the sensory organ responsible for detecting sound and sensing changes in both linear and angular acceleration. Understanding changes in gene expression as the inner ear develops is a crucial step in determining genes required for the development and function of this organ [OTT+11b, UCC+08a, JSP+07].

The novel protein proposed for the analysis is sequenced from genes involved in ion binding and/or transport in the developing and mature inner ear of the *African clawed frog, Xenopus laevis*.

In the earliest stages of this experiment we conducted homology searches, consisted in comparing the unknown sequence with all the known structures stored in the PDB, using BLAST[AMS+97], which resulted in 3 proteins with homology higher then 97%: 3EMN_X (e-value 3e-134), 2K4T_A (e-value 1e-133), 2JK4_A (e-value 7e-133). The e-value, represent the probability to get the retrieved alignment by generating a random sequence, and it indicates that the results obtained are very reliable.

Moreover we verified the consistency of the core residues using multiple sequence alignment (ClustalW)[TGH02], to eventually substitute the residues that differ in the alignments, with structural templates from the original sequence.

For the geometrical constraint modeling, we selected the best results returned by the protein alignment analysis w.r.t. the target protein. The structures determined are associated to protein sequences: 3EMN_X, 2K4T_A and 2JK4_A.

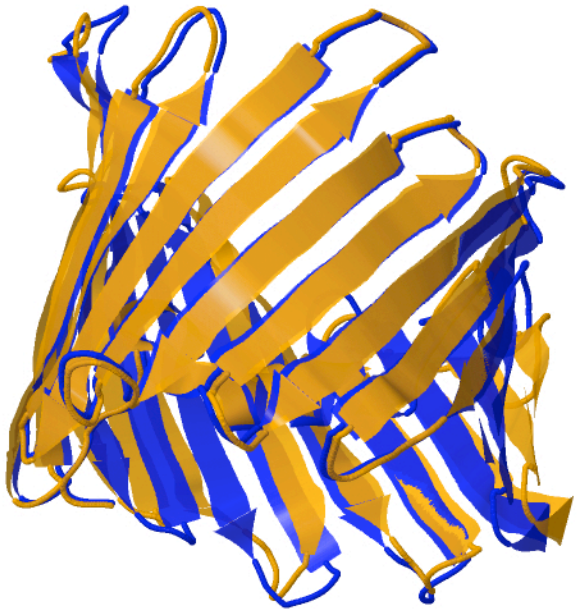

Figure 6.2: 3EMN_X prediction. The native structure is displayed in blue (darker) and predicted structure in orange (lighter).

The fragment set used, ranged from 4 to 100. For this experiment we select to optimize the RMSD measure (w.r.t. native structure 3EMN_X) since a high homology between the two sequences was detected ($\geq 99\%$) with low probability of errors.

Figure 6.2 shows the best prediction obtained using three fragment blocks, modeling the above constraints. In blue (darker color) we show the native structure of protein 3EMN_X (found to have the 99% of similarity with the target sequence) and in orange (lighter color) the predicted structure. The prediction was computed using 128 cores, yielding an accuracy of 2.65Å.

6.2 Intuitions, ideas and future directions

In this Section we propose different ideas aimed at improving our solver in terms of quality of the predictions and efficiency in generating them.

Energy branch & bounds. Experimental evidence shows that while searching for the best prediction, the solver generates many conformations that are not biologically meaningful. Recall that the meter adopted to evaluate the accuracy of a prediction, is the energy function. We plan to develop a branch-and-bound strategy that relies on the exploitation of the energy function evaluated on local consistent solutions. Such bound, would help in pruning those areas of the tree that do not contribute in producing a conformation associated with a lower energy value (w.r.t the best collected so far). Note that such bound can also be employed during the concurrent searches. In the parallel search exploration, the energy values are locally improved by agents working in different regions of the tree. Allowing the bounds information to reticulate among the concurrent solvers would help in guiding each agent in performing a "more" global-oriented search optimization.

Novel propagators. Employing efficient propagators (possibly) enhances the performances of the system. We plan to supplement the constraint solver with a new propagator which allows a better exploitation of the geometrical and distance

constraints. In particular, the novel propagator would help to identify inconsistent geometries for a chain of fragments to be placed. Such inconsistencies derive from the assembly of fragments leading in a very constrained region of the search space. Simulating k-fragment assembly steps, we can identify the possible points reached by the evolution of the peptide structure. Such information can be used to remove the chain of fragments that happen to the reach regions already occupied by some other protein block (placed in some previous assembly step).

Exploiting specialized energy models. The experimental results analyzed in Chapter 4 and Chapter 5 show that the energy model adopted in this work, is accurate in capturing the behavior of local interactions shaping secondary structures and macro blocks. Although the parts of the proteins characterized by a higher variability are not always well described, influencing the overall solution.
e have noticed that—even though in small amounts—the indiscriminate optimization of the energy function, causes an unconditional packing of many areas of the protein, including those regions characterized by a higher flexibility (e.g. loops and random coils). In general, it is reasonable to think that the packing of these regions does not follows the same rules of the one adopted by other more statistically recurrent structures. Indeed, these regions are more suitable to accommodate geometrical variations, to optimize the interaction of secondary structures.
It seams plausible that different protein regions fold in a slightly different way,

mainly according to their local structure.

A possible direction of investigation would be to consider a more "targeted" use of the energy model. Different energy functions may be adopted to model specific protein structures and blocks of structures—assuming that knowledge of such structures is given in input—or different energy weights could be adopted in various region of the protein. The idea is to shift from a global optimization of the energy function, to a more localized one, so to accommodate the characteristic of different protein regions. In this way we can exploit the structural knowledge, currently used to model geometrical constraints, as part of the interaction model.

New heuristics in loop modeling. The "function" of the protein loops can be simplistically described as that of connecting regions of secondary structures. Due to the difficulties of predicting the loop structure, given its amino acids sequence, we plan to investigate on a loop closure model that relies on statistical considerations and on the topology of the super-secondary structures that can be connected by such a loop [Lau04, FFOF06, TBOdB09].

Ad-hoc constraint generation suggestion system. For what concern the user defined constraint generation, experiments underline that the way protein regions are partitioned (to produce special fragments) plays an important role in the result of the final conformation. It is our intention to automatize the system to suggest the splitting sites for candidates special fragments. This suggestion system

would help in producing conformations that are structurally closer to the protein native state, yet, possibly, reducing the computational workload by avoiding some unnecessary degrees of freedom, in those area of the protein in which homology information can be used to generate special fragment candidates.

CONCLUSIONS

In this Thesis we presented a novel framework for the prediction and analysis of protein structures. Our approach relies on the use of constraint solving techniques to implement a fragment assembling methodology used to construct the protein tertiary structure. The fragments are built from a statistically significant set of short peptides extracted from the Protein Data Bank, as well as longer fragments obtained through user observations, homology studies, or secondary structure predictions.

The assembly process is modeled to met the requirements expressed by the imposed constraints, which in turn describe biological meaningful information devised from various knowledge and observations. The main constraint in use casts the local structural behavior and it is model using the concept of fragment. Other constraints capture geometrical restrictions, estimated distances and relative block positions and entropic considerations, derived from a suitable energy model.

We develop an ad-hoc imperative framework, seeking high modularity in the

way constraints are handled and the effect of their application propagated. We carefully implemented crucial data structures to handle efficient consistency and propagation of constraints. In addition, to scale up to large protein structure prediction, we introduced a parallel implementation, that guarantees an efficient exploration of the search space of possible conformations. In doing so, we exploit an MPI-based cluster distributed system, in a multi-thread environment.

Current results show that our system is suitable to tackle proteins up to 300 amino acids for which homology information is given. In addition, we show that the our system is able to produce predictions within a marginal error range, for short peptides for which a weak or no homologies information are supported.

Compared with other systems (e.g. Rosetta), the model proposed in this work has the advantage to present high modularity enabling the simple handling of ad-hoc constraints to model specific requirements, different search strategies and energetic models.

REFERENCES

[Aea02] Bruce Alberts and et al. *The Molecular Biology of the Cell*. Garland Science, fourth edition edition, 2002.

[AMS+97] Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997.

[Apt03] K. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[AYL89] N. L. Allinger, Y. H. Yuh, and J-H. Lii. Molecular mechanics. the mm3 force eld for hydrocarbons. *J. Am. Chem. Soc.*, 111:8551–8556, 1989.

[Bac98a] R. Backofen. Constraint techniques for solving the protein structure prediction problem. In *Proc. of CP 1998*, volume 1520, pages 72–86. LCNS, 1998.

[Bac98b] R. Backofen. Using constraint programming for lattice protein folding. In *Pac Symp Biocomput*, pages 389–400, 1998.

[Bac04] R. Backofen. A polynomial time upper bound for the number of contacts in the hp-model on the face-centered-cubic lattice (fcc). In *J. of Discrete Algorithms*, volume 2, pages 161–206, 2004.

[BBC+11] M. Best, K. Bhattarai, F. Campeotto, A. Dal Palù, H. Dang, A. Dovier, F. Fioretto, F. Fogolari, T. Le, and E. Pontelli. Introducing fiasco: Fragment-based interactive assembly for protein structure prediction with constraints. *WCB 2011 colocated to CP2011*, 2011.

[BBO+83] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus. Charmm: A program for macromolecular energy minimization and dynamics calculations. *J. Comp. Chem.*, 4:187–217, 1983.

[BKV00] M Bansal, S Kumar, and R Velavan. Helanal: a program to characterize helix geometry in proteins. *Journal of biomolecular structure & dynamics*, 17(5):811, 2000. hjjghj.

[BMF03] M. Berrera, H. Molinari, and F. Fogolari. Amino acid empirical contact energy denitions for fold recognition in the space of contact maps. *BMC Bioinformatics*, 8(4), 2003.

[Bro99] R. J. Brooker. *Genetics: Analysis and Principles*. Addison-Wesley, 1999.

[BW01a] Backofen and S. Will. Fast, constraint-based threading of hp sequences to hydrophobic cores. In *LNCS*, volume 2239, page 494508, 2001.

[BW01b] R. Backofen and S. Will. Optimally compact nite sphere packings hydrophobic cores in the fcc. In *Proc. Annual Symposium on Combinatorial Pattern Matching (CPM 2001)*, volume 2089, pages 257–272. LCNS, 2001.

[BW02] R. Backofen and S. Will. Excluding symmetries in constraint-based search. *Constraints*, 3(7):333–349, 2002.

[BW03] R. Backofen and S. Will. A constraint-based approach to structure prediction for simplied protein models that outperforms other existing methods. In Springer Verlag, editor, *Proc. of ICLP 2003*, 2003.

[BWBB99] R. Backofen, S. Will, and E. Bornberg-Bauer. Application of constraint programming techniques for structure prediction of lattice proteins with extended alphabets. *Bioinformatics*, 15(3):234–242, 1999.

[Cam11] F. Campeotto. Predizione della struttura di una proteina componendo frammenti: limiti computazionali e sviluppo di un predittore (in italian). Master's thesis, University of Udine, 2011.

[CC98] Y Cui and R Chen.... Protein folding simulation with genetic algorithm and supersecondary structure constraints. ... *Structure Function and Genetics*, Jan 1998.

[CGP+98] P. Crescenzi, D. Goldman, C. Papadimitriu, A. Piccolboni, and M. Yannakakis. On the complexity of protein folding. *Journal of Computational Biology*, 5(3):423–465, Fall 1998.

[CH02] X De La Cruz and E Hutchinson.... Toward predicting protein topology: An approach to identifying β hairpins. *Proceedings of the ...*, Jan 2002.

[Cri58] F.H.C. Crick. On protein synthesis. *Symp. Soc. Exp. Biol. XII*, XII:139–163, 1958.

[Cri70] F.H.C. Crick. Central dogma of molecular biology. *Nature*, 227:139–163, 1970.

[Dal06] A. Dal Palù. *Constraint Programming Approaches to the Protein Structure Prediction Problem.* PhD thesis, University of Udine, 2006.

[DDFP10] Alessandro Dal Palù, Agostino Dovier, Federico Fogolari, and Enrico Pontelli. Clp-based protein fragment assembly. *Theory and Practice of Logic Programming, special issue dedicated to ICLP 2010*, 10(4):709–724, 2010.

[DDFP11] A. Dal Palù, A. Dovier, F. Fologolari, and E. Pontelli. Clp-based protein fragment assembly. In *Proceedings of IJCAI 2011*, 2011.

[DDP06] A. Dal Palù, A. Dovier, and E. Pontelli. Global constraints for discrete lattices. In *In 2nd International Workshop on Constraint Based Methods for Bioinformatics (CP2006)*, volume 2, 2006.

[DDP07] A. Dal Palù, A. Dovier, and E. Pontelli. A constraint solver for discrete lattices, its parallelization, and application to protein structure prediction. *Softw. Pract. Exper.*, 37(13):1405–1449, 2007.

[Dil85] K. A. Dill. Theory for the folding and stability of globular proteins. *Biochemistry*, 1985.

[DS80] E.W. Dijkstra and C.S. Scholten. Termination detection for diffusing computation. *Inform. Process. Lett.*, 11(1):1–4, 1980.

[EK87] R. Elber and M. Karplus. Multiple conformational states of proteins: a molecular dynamics analysis of myoglobin. *Science*, 235:318–321, 1987.

[FAMCMS74] L. M. F. A. Momany, Carruthers, R. F. McGuire, and H. A. Scheraga. Intermolecular potentials from crystal data. iii. determination of empirical potentials and application to the packing congurations and lattice energies in crystal of hydrocarbons, carboxylic acids, amines, and amides. *J. Phys. Chem*, 78:1595–1620, 1974.

[FFOF06] N Fernandez-Fuentes, B Oliva, and A Fiser. A supersecondary structure library and search algorithm for modeling loops in protein structures. *Nucleic acids research*, 34(7):2085, 2006.

[FPD+07] F. Fogolari, L. Pieri, A. Dovier, L. Bortolussi, G. Giugliarelli, A. Corazza, G. Esposito, and P. Viglino. Scoring predictive models using a reduced representation of proteins: model and energy definition. *BMC Structural Biology*, 7(15), 2007.

[GJ03] T.W. Geisbert and P.B. Jahrling. Towards a vaccine against ebola virus. In *Expert Review of Vaccines*, volume 2, pages 777–789. Expert Reviews, 2003.

[GPC+01] G. Gupta, E. Pontelli, M. Carlsson, M. Hermegildo, and K. Ali. Parallel execution of prolog: a survey. *ACM TOPLAS*, 23(4):472–602, 2001.

[HT92] J. D. Honeycutt and D. Thirumalai. The nature of folded states of globular proteins. *Biopolymers*, 32:695–709, 1992.

[HTS+04] T. X. Hoang, A. Trovato, F. Seno, J. R. Banavar, and A. Maritan. Geometry and symmetry presculpt the free-energy landscape of proteins. In *Proc. Natl. Acad. Sci., USA*, 2004.

[Jac98] SE. Jackson. How do small single-domain proteins fold? *Fold. Des.*, 3(4):R81–R91, 1998.

[JSP+07] M.S. Jensen, T.L. Sorensen, J. Petersen, J.P. Andersen, and B. Vilsen. Crystal structure of the sodium-potassium pump. *Nature*, 450:1043–1049, 2007.

[JTR98] W. L. Jorgensen and J. Tirado-Rives. The opls potential functions for proteins. energy minimizations for crystals of cyclic peptides and crambin. *J. Am. Chem. Soc.*, 110:1657–1666, 1998.

[KB95] T. Kiefhaber and R.L. Baldwin. Intrinsic stability of individual alpha-helices modulates structure and stability of the apomyoglobin molten globule. *J. of Mol. Biol.*, 252:122–132, 1995.

[KB99] Krippahl and P. Barahona. Applying constraint propagation to protein structure determination. In LCNS, editor, *Proc. of CP 1999*, volume 1713, pages 289–302, 1999.

[KB02] L. Krippahl and P. Barahona. Psico: Solving protein structures with constraint programming and optimisation. In *Constraints*, volume 7, pages 317–331. Kluwer Academic Press, July/October 2002.

[KB03] L. Krippahl and P. Barahona. Propagating n-ary rigid-body constraints. In *Proc. of CP 2003*, volume 2833, pages 452–465. LCNS, 2003.

[KHE04] J. Kubelka, J. Hofrichter, and W.A. Eaton. The protein folding 'speed limit'. *Curr. Opin. Struct. Biol.*, 14(1):76–88, 2004.

[KKD+03] K. Karplus, R. Karchin, J. Draper, J. Casper, Y. Mandel-Gutfreund, M. Diekhans, and R. Hughey Source. Combining local-structure, fold-recognition, and new fold methods for protein structure prediction. *Proteins*, 6(53):491–497, 2003.

[KM97] W Kohn and C Mant. . . . α-helical protein assembly motifs. *Journal of Biological Chemistry*, Jan 1997.

[Lau04] F Laurent. Use of a structural alphabet for analysis of short loops connecting repetitive structures. *doaj.org*, Jan 2004.

[LD89] Kit Fun Lau and Ken A. Dill. A lattice statistical mechanics model of the conformational and sequence spaces of proteins. *Macromolecules*, 22:3986–3997, 1989.

[LD90] Kit Fun Lau and Ken A. Dill. Theory for protein mutability and biogenesis. In USA, editor, *Proc. Natl. Acad. Sci*, volume 87, pages 638–642, 1990.

[LDA+00] S. Lovell, I. Davis, W. Arendall, P. de Bakker, J. Word, M. Prisant, J. Richardson, and D. Richardson. Structure validation by c_α geometry: ϕ, *psi* and c-deviation. *Proteins*, 2000.

[Lev68] C. Levinthal. Are there pathways for protein folding? *J. Chim. Phys.*, 65:44–45, 1968.

[Lig74] A. Light. *Proteins: Structure and Function.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1974.

[LKJ+04] J. Lee, SY Kim, K. Joo, I. Kim, and J. Lee. Prediction of protein tertiary structure using profesy, a novel method based on fragment assembly and conformational space annealing. *Proteins*, 4(56):704–714, 2004.

[LNC08] Albert Lehninger, David L. Nelson, and Michael M. Cox. *Lehninger Principles of Biochemistry*. W. H. Freeman, fifth edition edition, 2008.

[LSF+10] D.W. Leung, R.S. Shabman, M. Farahbakhsh, M. Prins, K.C. Borek, D.M. Wang amd E. Mhlberger, and C.F. Basler amd G.K. Amarasinghe. Structural and functional characterization of reston ebola virus vp35 interferon inhibitory domain. *J Mol Biol*, 399(3):347–357, Jun 2010.

[Mac75] A.K. Mackworth. Consistency in networks of relations. Technical Report 75, Dept. of Computer Science, Univ. of B.C. Vancouver, 3 1675.

[Mac77] A.K. Mackworth. On reading sketch maps. In Cambridge MA, editor, *In Proceedings IJCAI77*, pages 598–606, 1977.

[Mat87] Friedemann Mattern. Algorithms for distributed termination detection. In Heidelberg Springer, Berlin, editor, *Distributed Computing*, volume 2, pages 161–175, 1987.

[MBHC95] A.G Murzin, S.E Brenner, T Hubbard, and C Chothia. Scop: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247(4):536–540, 1995.

[Mis83] Jayadev Misra. Detecting termination of distributed computations using markers. In York NY USA ACM, editor, *In Proceedings of the second annual ACM symposium on Principles of distributed computing (PODC '83)*, pages 290–294, 1983.

[MSM00] Cristian Micheletti, Flavio Seno, and Amos Maritan. Recurrent oligomers in proteins: An optimal scheme reconciling accurate and concise backbone representations in automated folding and design studies. *PROTEINS: Structure, Function, and Genetics*, pages 662–674, 2000.

[MSVH09] Laurent Michel, Andrew See, and Pascal Van Hentenryck. Transparent parallelization of constraint programming. *INFORMS Journal on Computing*, 21(3):363–382, Summer 2009.

[Neu97] Arnold Neumaier. Molecular modeling of proteins and mathematical prediction of protein structure. *SIAM Rev.*, 39(3):407–460, 1997.

[OTdB07] B Offmann, M Tyagi, and A.G de Brevern. Local protein structures. *Current Bioinformatics*, 2(3):165–202, 2007.

[OTT+11a] A. Oshima, K. Tani, M.M. Toloue, Y. Hiroaki, A. Smock, S. Inukai, A. Cone, B.J. Nicholson, G.E. Sosinsky, and Y. Fujiyoshi. Asymmetric configurations and n-terminal rearrangements in connexin26 gap junction channels. *J.Mol.Biol.*, 405, 2011.

[OTT+11b] A. Oshima, K. Tani, M.M Toloue, Y. Hiroaki, A. Smock, S. Inukai, A. Cone, B.J. Nicholson, G.E. Sosinsky, and Y. Fujiyoshi. Asymmetric configurations and n-terminal rearrangements in connexin26 gap junction channels. *J.Mol.Biol*, 405:724–735, 2011.

[OW83] M. Ohgushi and A. Wada. 'molten-globule state': a compact form of globular proteins with mobile side-chains. *FEBS Lett*, 1(28):21–24, Nov 1983.

[Per99] L. Perron. Search procedures and parallelism in constraint programming. In Springer Verlag, editor, *Principles and Practice of Constraint Programming (CP)*, 1999.

[Rea09] S. Raman and et al. Structure prediction for casp8 with all-atom renement using rosetta. *Proteins*, 77(S9):89–99, 2009.

[RSMB04] CA Rohl, CE Strauss, KM Misura, and D Baker. Protein structure prediction using rosetta. *Methods Enzymol*, 383:66–93, 2004.

[RvBW06] F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.

[Sin05] M Singh. Predicting protein secondary and supersecondary structure. *Handbook of Computational Molecular Biology ...*, Jan 2005.

[SKHB97] Kim T. Simons, Charles Kooperberg, Enoch Huang, and David Baker. Assembly of protein tertiary structures from fragments with similar local sequences using simulated annealing and bayesian scoring functions. *J. Mol. Biol.*, 268:209–225, 1997.

[TBOdB09] M Tyagi, A Bornot, B Offmann, and A.G de Brevern. Analysis of loop boundaries using different local structure assignment methods.

Protein science: a publication of the Protein Society, 18(9):1869, 2009.

[TGH02] Julie D. Thompson, Toby. J. Gibson, and Des G. Higgins. *Multiple Sequence Alignment Using ClustalW and ClustalX*. John Wiley and Sons, Inc., 2002.

[UCC+08a] R. Ujwal, D. Cascio, J.P. Colletier, S. Faham, J. Zhang, L. Toro, and P. Ping. The crystal structure of mouse vdac1 at 2.3 a resolution reveals mechanistic insights into metabolite gating. *PNAS*, 105(46):17742–17747, 2008.

[UCC+08b] R. Ujwal, D. Cascio, J.P. Colletier, S. Faham, J. Zhang, L. Toro, P. Ping, and J. Abramson. The crystal structure of mouse vdac1 at 2.3 åresolution reveals mechanistic insights into metabolite gating. In *Proc.Natl.Acad.Sci.USA*, pages 17742–17747, 2008.

[Vea01] J. Craig Venter and et al. The sequence of the human genome. *Science*, 291:1304–1351, 2001.

[WV85] L T Wille and J Vennik. Computational complexity of the ground-state determination of atomic clusters. *J. of Physics A: Mathematical and General*, 18(8):419–422, 1985.