A Declarative Concurrent System for Protein Structure Prediction on GPU

Federico Campeotto[a,b], Agostino Dovier[b], and Enrico Pontelli[a]

[a] *New Mexico State University, Las Cruces, New Mexico, USA*

[b] *University of Udine, DIMI, Udine, Italy*

October 10, 2014

Abstract

This paper provides a novel perspective in the *Protein Structure Prediction (PSP)* problem. The PSP problem focuses on determining putative 3D structures of a protein starting from its primary sequence. The proposed approach relies on a *Multi-Agent System (MAS)* perspective, where concurrent agents explore the folding of different parts of a protein. The strength of the approach lies in the agents' ability to apply different types of knowledge, expressed in the form of *declarative constraints*, to prune the search space of folding alternatives. The paper makes also an important contribution in demonstrating the suitability of a *General-Purpose Graphical Processing Unit (GPGPU)* approach to implement such MAS infrastructure, with significant performance improvements over the sequential implementation and other methods.

1 Introduction

Structural biology is the branch of molecular biology and biochemistry that deals with problems related to the molecular structure of DNA and proteins, and how such structures affect behavior and function. Our specific focus in this work is concerned with structural studies of *proteins*. Proteins are macro-molecules that are critical to the regulation of vital functions in all biological processes. The prediction of the 3D structure of a protein from a sequence of amino acids is one of the oldest and most challenging a problems in bioinformatics [2]. Such prediction is of fundamental importance, since structural properties are critical in determining the functions of proteins and in understanding how proteins interact to make cellular processes happen [47, 4, 1]. The huge number of conformations in which a protein can potentially fold, together with the lack of an accurate energy function that can guide the folding process, make the *Protein Structure Prediction (PSP)* problem a difficult task, even for small or medium-length proteins (i.e., with less than 100 amino acids).

In this paper, we tackle the PSP problem using a perspective that builds on the methodologies inherited from the *Multi-Agent Systems (MAS)* domain. The proposed MAS approach is used to concurrently explore and then assemble foldings of local segments of the protein. Distinct agents are in charge of retrieving, filtering, and coordinating local information about parts of a protein, aiming to reach a global consensus. Relationships among substructures are described and exploited in terms of *constraints*—where a constraint is a high-level and declarative specification of required mutual relationships among entities. In our case, the proposed constraints deal with spatial relationships among parts of proteins being configured and assembled. A strength of constraint-based methods is their *elaboration tolerance,* that allows the incremental addition of new knowledge about the protein (e.g., properties of the amino acids, knowledge about specific substructures) without the need of redesigning the solving mechanisms. Thus, any new knowledge about a protein can be readily integrated and used to prune the space of potential conformations. Furthermore, constraint-based methods offer the power of *propagation* of any decision made during construction of a protein conformation, immediately removing infeasible branches of search space as each decision is performed. Indeed, the interest towards constraint-based methods for structural bioinformatics has grown in recent years—the reader is referred to [5, 19] for recent surveys.

This MAS-based model is effectively implemented using a *General Purpose Graphical Processing Unit* (GPU) architecture, leading to significant gains in terms of execution time, compared to a sequential implementation.

GPU architectures offer parallelism at a low cost and they elegantly accommodate the PSP multi-agent infrastructure proposed here. It must be observed that, in spite of the high number of cores available in a typical GPU, performance improvements w.r.t. a sequential implementation is not automatically guaranteed, due to the features and bottlenecks of typical GPU architectures (e.g., memory access, dependence of the code executions of the various threads running in the same streaming processor).

In this paper, we present a solver that performs a constraint-based local search, distributed among several agents. The computation proceeds until a local minimum is found. Experimental results confirm that the local minimum reached captures with good precision the actual shape of the protein being studied. Agents use GPU cores to explore large portions of the search space and to propagate constraints on the ensemble of structures produced by each agent. The solver is capable of achieving excellent performance and demonstrates speedups over a sequential solver on a large pool of benchmarks.

A preliminary version of this paper has been presented at the 2013 RCRA meeting and in a IEEE ICPP short paper [12]. This paper greatly expands on the previous presentations, by providing the first complete description of the solver and a precise analysis of its performance and properties.

2 Related Work and Background

2.1 Proteins and Structures

In this section, we summarize some relevant concepts from structural and molecular biology. We refer the reader to the existing literature for a more in-depth discussion of these concepts (e.g., [16, 19]).

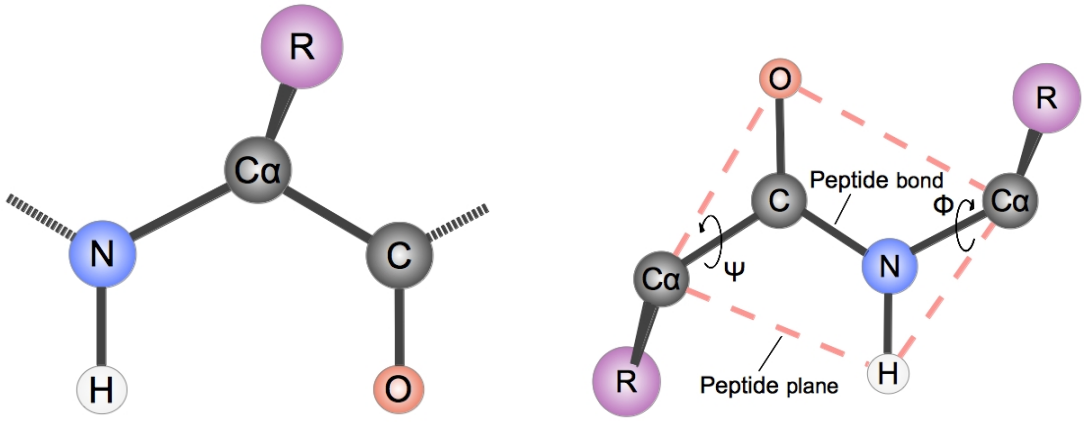

Figure 1: Representation of an amino acid (left), and the peptide plane (right), with the peptide bond that binds two amino acids, and the phi (ϕ) and psi (ψ) dihedral angles

A protein is a polymer composed of a sequence of simple molecules, called *amino acids*. This sequence, referred to as the *primary sequence*, folds into a unique three-dimensional stable structure, that represents the conformation with *minimum* free energy; such structure determines the biological functions of the protein. We refer to this structure as the *tertiary structure* of the protein. The *Protein Structure Prediction (PSP)* problem is defined as the problem of finding the tertiary structure of a *target* protein given its primary sequence.

There are 20 types of amino acids. Each one can be represented by a common structure, called the *backbone*, constituted by two Carbon, one Hydrogen, one Oxygen, and one Nitrogen atoms (see Fig. 1 on the left). The distinction between amino acids originates from an additional group of atoms (containing from 1 to 18 atoms), called the *side-chain*. In a simple model, this group can be represented by a single large *centroid* (*group R*). The centroid is linked to the central carbon atom *Cα*, and its size and distance from *Cα* can be statistically determined for each amino acid type.

Two consecutive amino acids are linked together by a *peptide bond*, that connects the *C-O* group of the first amino acid to the *N-H* group of the following one. Due to the double bond character of the peptide bond, these four atoms, together with the *Cα*-atoms immediately preceding and following them, lie on the same plane, known as *peptide plane* (Figure 1 on the right). Therefore, the peptide bond backbone has only two

degrees of freedom per amino acid group (see also Figure 5 later in the paper): **(1)** The rotation around the N-$C\alpha$ bond (ϕ angle), and **(2)** The rotation around the $C\alpha$-C bond (ψ angle). A sequence of amino acids joined by peptide bonds forms the backbone chain of the protein, whose 3D coordinates describe the complete tertiary structure. Sequences of ϕ and ψ angles, defined by backbone atoms $CNC\alpha C$ and $NC\alpha CN$, respectively, determine exactly the tertiary structure (as 3D coordinates of atoms), and vice-versa.

Specific ranges of the torsional angles determine also particular local structures, referred to as *secondary structures* of the protein. Secondary structure elements often assume regular and repetitive spatial shapes, being constrained by the hydrogen bonds formed by local inter-residue interactions. The most common examples are the α-helices and the β-sheets. Other parts of the polypeptide instead present less regular structures. They allow the folds between the structured parts, and they are usually identified with *loops* and *turns*, as shown in Figure 3.

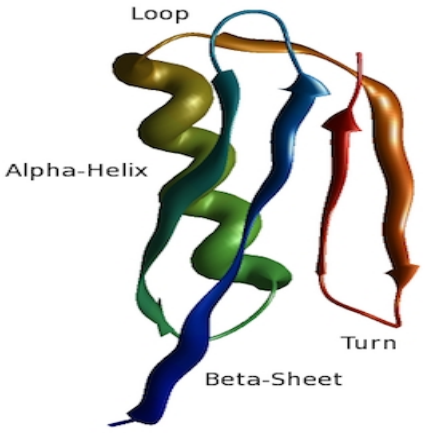

Figure 3: *α-helix*, *β-sheets*, *loops*, and *turns*

Proteins can be generally classified on the basis of their secondary structure elements: if only α-helices (resp. β-sheets) are present, we are in presence of a α (resp. β) protein; otherwise it is an $\alpha\beta$-protein.

In the literature, several geometric models for proteins have been proposed. One choice that influences the quality and the complexity of computational approaches to protein structural studies is the number of points that describe a single amino acid. In this paper, we use a simple model, where a protein composed of n amino acids is described by the 3D coordinates of the sequence of atoms $(N\text{-}C\alpha\text{-}C\text{-}O\text{-}H)^n$, and the side chain is represented by a single centroid. Centroid positions are calculated *a-posteriori* on the basis of the coordinates of the other atoms of the backbone. If each atom is identified by a triplet representing its 3D coordinates, then the tertiary structure of a target protein will be defined by a list of $[p_1, \ldots, p_{15n}]$ values, where $p_{i \in [1..15n]} \in \mathbb{R}$.

2.2 Protein Structure Prediction

PSP is a known challenging problem in bioinformatics, that prompted the development of several approaches in the literature. A class of attractive, but computationally expensive, approaches is the class of *ab-initio* prediction methods, that simulate the folding process by using knowledge about the chemical structure of the protein and the laws of physics. They usually require a large amount of computational resources; good results have been obtained from the adoption of distributed computing [6] and high-performance architectures [49, 33]. GPU technology has also been used for ab-initio prediction (e.g., [38]). In a recent proposal, [38] address the PSP problem using GPUs to perform molecular dynamic simulations, i.e., simulations of physical movements of atoms and molecules.

An alternative strategy to PSP is represented by the class of *comparative modeling* approaches; these methods assume that a limited set of structural motifs can represent the majority of the existing protein structures. Hence, previously determined structures can be used as templates to predict unknown targets. In the Rosetta algorithm [46], small fragments of known proteins are combined together in order to recreate the structure with a minimum of free energy (i.e., the native structure). This idea has proven to be promising, making Rosetta one of the most successful comparative modeling approaches. In I-TASSER [54, 40] a comparative modeling approach is improved with a hierarchical strategy. Given a target sequence, I-TASSER first generates some protein templates through "threading" techniques. Afterwards, it assembles template fragments in order to generate a set of candidate structures, from which it extracts the structure with minimum energy. The I-TASSER server (`http://zhanglab.ccmb.med.umich.edu/I-TASSER`) was ranked as the best server for protein structure prediction in recent CASP competitions—an international series of competitions aimed at establishing the current state-of-the-art in protein structure predictions.

Last but not least, extensive research has been conducted in predicting locations, types and conformations of secondary structure elements—the interested reader is referred to [39] for a recent survey.

Declarative techniques have been extensively employed to model the PSP problem. Using *Constraint Programming (CP)*, it is easy to describe spatial properties of the unknown protein in terms of geometric constraints. In these models, it is common to superimpose the protein structure on a discretized representation of the three dimensional space, often organized as a crystal lattice structure. Successful results have been obtained for both the simple model with only two amino acids (the *HP* model), by solving a constraint optimization

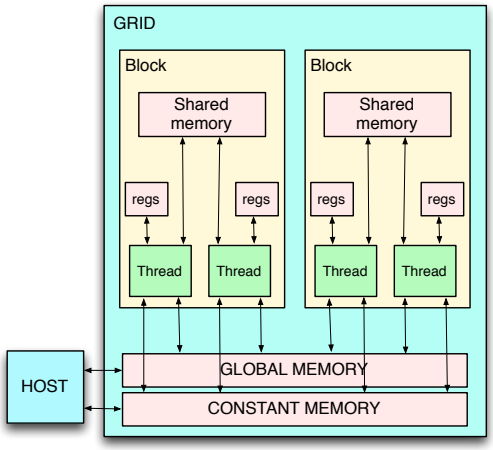

Figure 4: CUDA Logical Architecture

problem [3], and for the complete model [21] that uses all the different types of amino acids. A promising idea has been adopted in [22], where the 3D conformation of a protein is predicted via protein fragments assembly, modeled in terms of finite domain constraints and implemented using a dedicated constraint logic programming system.

The use of *multi-agent systems* has been relatively limited in the context of the PSP problem. An agent-based framework for protein structure prediction, using machine learning techniques, is presented in [37]. Several portfolio approaches, that combine results produced by several protein prediction servers, are presented in literature (e.g., [26]). In [11], the authors use *Concurrent Constraint Programming* techniques, where a blackboard model [15] supports communication among agents. Another multi-agent framework that uses a blackboard architecture is presented in [30], while in [18] agents are used as a reinforcement learning approach for solving the PSP problem in the 2-amino acid HP model.

Finally, we would like to mention that the use of GPU architectures has shown its advantages in several other bioinformatics applications, such as computational proteomics [32], DNA sequencing [43], and molecular docking [48], as well as in the context of Constraint Programming and SAT solving [14, 20].

2.3 GPU Computing in a Nutshell

Modern graphic cards (*Graphics Processing Units*) are multiprocessor devices, offering hundreds of computing cores and a rich memory hierarchy for graphical processing (e.g., DirectX and OpenGL). Efforts like NVIDIA's *CUDA—Compute Unified Device Architecture* [42] aim at enabling the use of the multicores of a GPU to accelerate general applications—by providing programming models and APIs that enable the full programmability of the GPU. In this paper, we consider the CUDA programming model. The underlying conceptual model of parallelism supported by CUDA is *Single-Instruction Multiple-Thread (SIMT)*, a variant of the SIMD (Single-Instruction Multiple Data) model. In SIMT, the same instruction is executed by different threads that run on identical cores, while data and operands may differ from thread to thread. CUDA's architectural model is summarized in Figure 4.

Different NVIDIA GPUs provide varying numbers of cores, their organization, and amounts of memory. The GPU is constituted by a series of *Streaming Multi-Processors (SMs)*; the number of SMs depends on the specific class of GPUs—e.g., the Fermi architecture provides 16 SMs. In turn, each SM contains a number of computing cores (each with a fully pipelined ALU and floating-point unit); the number of cores per SM may range from 8 (in the older G80 platforms) to 32 (e.g., in the Fermi platforms). Each GPU provides access to on-chip memory (for thread registers and shared memory) and off-chip memory (L2 cache, global memory and constant memory).

CUDA introduces a logical view of computations, allowing programmers to define abstract parallel work and to schedule it among different hardware configurations (see Fig. 4). A typical CUDA program is a C/C++ program that includes parts meant for execution on the CPU (referred to as the *host*) and parts meant for

parallel execution on the GPU (referred as the *device*). A parallel computation is described by a collection of *kernels*. Each kernel is a function to be executed by several threads. Threads spawned on the device to execute a kernel are hierarchically organized to facilitate the mapping of the threads to the (possibly multi-dimensional) data structures being processed: threads are organized in a 3-dimensional structure (called *block*), and blocks themselves are organized in 2-dimensional tables (called *grids*). CUDA maps blocks (coarse-grain parallelism) to the SMs for execution; each SM schedules the threads in a block (fine-grain parallelism) on its computing cores in chunks of 32 threads at a time (called *warps*), thus allowing group of threads in a block to use the computing resources while other threads of the same block might be waiting for information (e.g., completing a slow memory request). Threads have access to several memory levels, each with different properties in terms of speed, organization (e.g., banks that can be concurrently accessed) and capacity. Each thread stores its private variables in very fast registers (anywhere from 8K to 64K per SM); threads within a block can communicate by reading and writing a common area of memory (called *shared memory*). Communication between blocks and with the host is realized through a large *global memory* (up to several gigabytes).

The kernel, invoked by the host, is executed by the device and it is written in standard C-code. The number of running blocks (gridDim) and the number of threads of each block (blockDim) are specified in the call executing the kernel, with the following syntax:

$$\mathsf{Kernel} \lll \mathsf{gridDim}, \mathsf{blockDim} \ggg (\mathsf{param}_1, \ldots, \mathsf{param}_n);$$

In order to perform a computation on the GPU, it is possible to move data between the host memory and the device memory. By using the specific identifier of each block (blockIdx—providing x, y coordinates of the block in the grid), its dimension (blockDim) and the identifier of each thread (threadIdx—providing x, y, z coordinates for the thread within the block), it is possible to differentiate the data accessed by each thread and the corresponding code to be executed. For example, the following code fragment shows a kernel and the corresponding call from the host. Each element of a two dimensional matrix is squared, and each thread is in charge of one element of the matrix. The matrix A is represented by a pointer in the device's global memory. CUDA provides functions (e.g., cudaMemCopy) to transfer data between the host and the device's global memory.

```
int main() {
   ...
    dim3 thrsBlock(n,n);
    sqMatrix<<<1,thrsBlock>>>(A);
    ...
```

```
__global__ sqMatrix(float *Mat){
    int i=threadIdx.x;
    int j=threadIdx.y;
    Mat[i][j] = Mat[i][j]*Mat[i][j];
}
```

While it is relatively simple to develop correct CUDA programs (e.g., by incrementally modifying an existing sequential program), it is challenging to design an efficient solution. Several factors are critical in gaining performance. The SIMT model requires active threads in a warp to execute the same instruction—thus, diverging flow paths among threads may reduce the amount of actual concurrency. Memory levels have significantly different sizes (e.g., registers are in the order of dozens per thread, while shared memory is in the order of a few kilobytes per block) and access times; different cache behaviors are applied to different memory levels (e.g., constant memory is a cached read-only global memory) and various optimization techniques are used (e.g., accesses to consecutive global memory locations by contiguous threads can be *coalesced* into a single memory transaction). Thus, optimization of CUDA programs requires a thorough understanding of the hardware characteristics of the GPU being used.

2.4 Constraint Satisfaction and Optimization Problems

A *Constraint Satisfaction Problem (CSP)* is a triple $\mathcal{T} = \langle X, D, C \rangle$ where $X = \{x_1, \ldots, x_m\}$ is a set of variables, $D = \{D_1, \ldots, D_m\}$ is the corresponding set of variable domains.[1] For the sake of simplicity, we will use the notation D^x to denote the domain of the variable x. $C = \{C_1, \ldots, C_h\}$ is a finite set of constraints. A constraint is a relation $C_i \subseteq D_{i_1} \times \cdots \times D_{i_{k_i}}$ for some set of indices $\{i_1, \ldots, i_{k_i}\} \subseteq \{1, \ldots, m\}$; let $scope(C_i)$ be $\{x_{i_1}, \ldots, x_{i_{k_i}}\}$. A *solution* to a CSP $\mathcal{T}$ is a m-tuple $\langle s_1, \ldots, s_m \rangle$ such that $s_j \in D_j$ for $1 \leq j \leq m$ and for each $1 \leq i \leq h$ $\langle s_{i_1}, \ldots, s_{i_{k_i}} \rangle \in C_i$. If $\mathcal{T}$ has a solution it is said to be *consistent*, otherwise it is inconsistent.

A *Constraint Optimization Problem (COP)* is a pair $\mathcal{Q} = \langle \mathcal{T}, E \rangle$, where $\mathcal{T}$ is a CSP and $E : D_1 \times \cdots \times D_m \to \mathbb{R}$ is a *cost function*. A solution $\vec{s}$ of the COP $\mathcal{Q}$ is a solution of $\mathcal{T}$ such that $E(\vec{s}) = \min_{\vec{x} \text{ solution of } \mathcal{T}} E(\vec{x})$.

A solution of a CSP is often found by alternating two steps: **(1)** Selection of a variable x_i and assignment $x_i = d$ for a value $d \in D_i$ (*labeling*), and **(2)** Propagation of the information $x_i = d$ for removing elements from

[1] In this paper, we restrict our attention to *finite domain* variables.

the domains of the remaining variables (*constraint propagation*), possibly detecting inconsistencies—when a domain becomes empty. Solving a COP implies computing $E(\vec{s})$ for each solution of the CSP, and imposing additional constraints, in order to avoid the computation of solutions that do not improve the best value found so far. Phase **(1)** implements non-deterministic choices, often explored using a backtracking procedure, while phase **(2)** is typically deterministic and implemented by fast filtering algorithms. GPUs can be efficiently used in **(1)** and in **(2)**: parallelism can be exploited in order to concurrently explore different variable-value assignments, and constraint propagation can be distributed on multiple cores.

A particular search technique frequently employed in solving a COP $\mathcal{Q} = \langle \mathcal{T}, E \rangle$ is the *Large Neighborhood Search* (LNS) ([50, 44]). The basic idea is as follows. Once one solution of $\mathcal{T}$ is found, instead of proceeding with some form of backtracking, a subset of the variables (chosen randomly, often with some general or problem oriented strategy) are uninstantiated. A new (small) COP minimizing the overall objective function—or simply improving it—with the assignment of these variable is solved, and the procedure is repeated starting from the last solution. The computation is usually stopped by a timeout or by a number of consecutive non-improving steps. The technique is proved to be very effective in scheduling problems (see, e.g., [51]). In this work, we use LNS to explore the potential foldings of a protein structure: first we define a tentative initial solution for the tertiary structure of the target protein (e.g., the straight configuration), then we run small COPs in order to fold secondary structure elements and to model loops and turns that join them together.

3 Problem Formalization

Given a primary sequence $\vec{a} = a_1 \cdots a_n$ of a protein of length n (each a_i is a symbol representing an amino acid), we model the PSP problem as a COP $\mathcal{Q} = (\langle X, D, C \rangle, E)$, where X, D, C, E are described in the following subsections.

3.1 Variables and domains

$X = \mathcal{X} \cup \mathcal{P}$ is a set of finite domain variables, where $\mathcal{X} = \{x_1, y_1, \ldots, x_n, y_n\}$, and $\mathcal{P} = \{p_1, \ldots, p_{15n}\}$ (hence, $m = |X| = 17n$). The variables x_i and y_i (for $1 \leq i \leq n$) are associated to the torsional angles ϕ and ψ of the i^{th} amino acid, respectively.

The variables $p_{5(i-1)+3t+1}, p_{5(i-1)+3t+2}, p_{5(i-1)+3t+3}$ $(i = 1, \ldots, n$ and $t = 0, \ldots, 4)$ are associated to the x, y, z coordinates of the t^{th} atom of the i^{th} amino acid a_i. More precisely, if $t = 0$ then it is an N-atom, referred as n_i; if $t = 1$ then it is the $C\alpha$-atom, referred as $C\alpha_i$; if $t = 2$ then it is a C-atom, referred as c_i; if $t = 3$ then it is an O-atom, referred as o_i; and if $t = 4$ then it is a H-atom, referred as h_i. We will denote the above list of 15 coordinates as $\vec{\mathcal{P}}_i = \langle n_i, C\alpha_i, c_i, o_i, h_i \rangle$ and we refer to these variables as *point* variables.

D^{x_i} and D^{y_i} will store sets of angles retrieved from a database of proteins using statistical information (we use DASSD ([25]). The domains for all the point variables are initially set all equal to the range $[-500000..500000]$, that has been experimentally proved to be large enough to accommodate all proteins tested in our benchmarks.

3.2 Constraints

C is a finite set of constraints over X. These constraints describe geometric properties that the final structure must satisfy to be a physically admissible structure. There is a strong correlation between $\mathcal{X}$ and $\mathcal{P}$, allowing us to infer the first from the second and vice versa. Keeping both types of variables explicit allows the programmer to easily add specific angle or spatial constraints. In particular, assignments of angle variables in $\mathcal{X}$ identify structures that are represented by ground points in $\mathcal{P}$. On the other hand, structures obtained by constraint propagation over variables in $\mathcal{P}$ identify a unique sequence of pairs of angles ϕ and ψ for the variables in $\mathcal{X}$.

Let us introduce the most relevant constraints used to encode the PSP problem.

3.2.1 `table` Constraints

Values for variables in $\mathcal{X}$ are retrieved from a statistical database. These values are given as pairs $\langle \phi, \psi \rangle$. Therefore, for each $i = 1, \ldots, n$, we consider a *table* constraint that associates $\langle x_i, y_i \rangle$ to the possible admissible pairs for those angles. As a result, during the search, the assignment of x_i and y_i is done simultaneously according to the table.

3.2.2 The `alldistant` Constraint

This constraint has been originally introduced by [24]. Given a list of $3k$ variables $\mathcal{P} = (p_1, \ldots, p_{3k})$, and a list of k positive values $\vec{d}$, the constraint $\texttt{alldistant}(\mathcal{P}, \vec{d})$ imposes a distance relation between each pair of 3D points identified by the variables in $\mathcal{P}$, i.e., $\forall i \in \{1, \ldots, k-1\}$ and $\forall j \in \{i+1, \ldots, k\}$:

$$\| \underbrace{(p_{(i-1)3+1}, p_{(i-1)3+2}, p_{(i-1)3+3})}_{A} - \underbrace{(p_{(j-1)3+1}, p_{(j-1)3+2}, p_{(j-1)3+3})}_{B} \| \geq d_i + d_j$$

where $\| \cdot \|$ is the Euclidian norm. d_i and d_j can be seen as the radii of two spheres centered in the points A and B, respectively. The constraint states that these two spheres cannot intersect.

We use this constraint to model the fact that the various atoms have a minimum distance each other. In particular, the value d_i is chosen as the radius of a sphere containing the atom of coordinates $(p_{(i-1)3+1}, p_{(i-1)3+2}, p_{(i-1)3+3})$. An additional `alldistant` constraint, that considers only the $C\alpha$ atoms of the amino acids, can also be added, imposing a minimum distance that depends on the the radii of sphere that contain their respective amino acids. These radii are computed from an average analysis in a database of known proteins.

3.2.3 The Single Angle (`sang`) Constraint

Given the list $\mathcal{X}$ of $2n$ FD variables and the list $\mathcal{P}$ of $15n$ point variables, and given $i \in 2, \ldots, n$, the *single angle* constraint $\texttt{sang}(i, \mathcal{X}, \mathcal{P})$ imposes a relation between the lists of points of the 15 variables $\vec{\mathcal{P}}_{i-1}$ and the subsequent 15 variables $\vec{\mathcal{P}}_i$ so as to satisfy:

$$\vec{\mathcal{P}}_i \in \{\mathbf{Rot}(\vec{\mathcal{P}}_{i-1}, \phi, \psi) \, : \, \langle \phi, \psi \rangle \in D^{x_i} \times D^{y_i}\}$$

where $\mathbf{Rot}(\cdot)$ is the roto-translation matrix needed to properly align the amino acid structure described by the angles $\langle \phi, \psi \rangle$ with the position of the previously placed atoms in $\vec{\mathcal{P}}_{i-1}$ (see Fig. 5). We use this constraint to model the relative positions of consecutive tuples of atoms related to amino acids $i-1$ and i according to the angles selected.

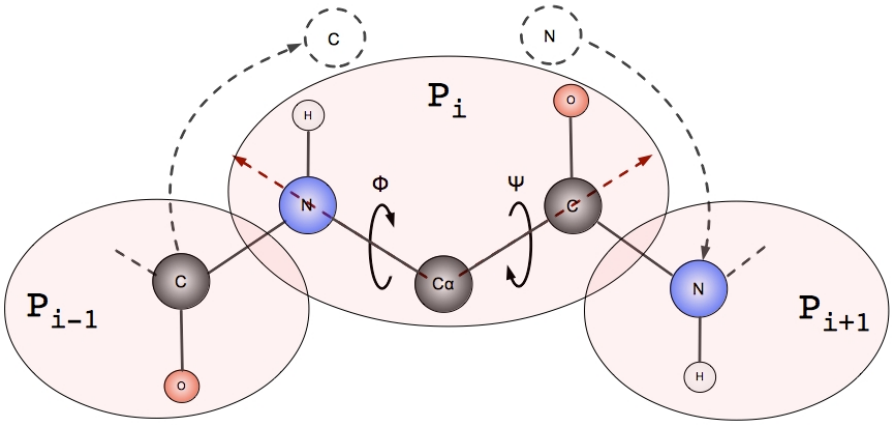

Figure 5: Rotation of the vector $C\alpha$–N (angle ϕ) or the vector $C\alpha$–C (angle ψ)

3.3 The Cost Function E

Since our aim is to find the tertiary structure that minimizes the free energy of the protein, we used as cost function a protein *energy function* already used in literature (e.g., the one adopted in [23]), composed of three components, described below:

1) *Hydrogen component:* Hydrogen bonding potentials are calculated from pairs of atoms N–H of amino acid i (n_i, h_i) and an O atom of another amino acid j (o_j), that are located within a certain distance threshold; an auxiliary statistical table $\texttt{tab}_h$ is used [35]. This contribution is calculated by a function $\texttt{Hydro}(X)$ defined as follows:

$$\texttt{Hydro}(X) = \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} hc(X,i,j)$$

where $hc(\cdot)$ returns the energy potential of one hydrogen bond:

$$hc(X,i,j) = \begin{cases} \texttt{tab}_h(\delta(X,i,j), \Theta(X,i,j), \Psi(X,i,j), \Gamma(X,i,j)) & \\ & \text{if } 1.75 \leq \delta(X,i,j) \leq 2.60 \\ 0 & \text{otherwise} \end{cases}$$

where: $\delta(X,i,j) = \|h_i - o_j\|$, $\Theta(X,i,j)$ is the bond angle $\widehat{o_j\, h_i\, n_i}$, $\Psi(X,i,j)$ is the bond angle $\widehat{C\alpha_j\, o_j\, h_i}$, and $\Gamma(X,i,j)$ is the torsional angle identified by $n_i, h_i, C\alpha_j, o_j$.

2) *Contact component:* it calculates the contribution of the contact of each pair of centroids of the side chain using the statistical table of contact energies $\texttt{tab}_c$ [8]. This component considers a threshold distance equal to the sum of the Van der Walls radii of the side chains of the amino acids involved.[2] If the distance is greater than such threshold, the potential decreases quadratically. The contact component $\texttt{Cont}(X, \vec{a})$ is defined as follows:

$$\texttt{Cont}(X, \vec{a}) = \sum_{i=2}^{n-1} \sum_{j=i+1}^{n-1} contact(\gamma(X,i,a_i), \gamma(X,j,a_j), a_i, a_j)$$

where $\vec{a}$ is the sequence of amino acids and the function $\gamma(X,i,a)$ returns the position of the side chain centroid of the amino acid i, which is dependent on the type of the amino acid a and the points $C\alpha_{i-1}, C\alpha_i, C\alpha_{i+1}$. Let us observe that the first and the last centroids of the structure are not taken into account since the tails of the protein do not contribute significantly to the energy value. The $contact(\cdot)$ function returns the contact potential computed in [8], retrieved by the indexing function $\texttt{tab}_c$:

$$contact(p,q,a,a') = \begin{cases} \texttt{tab}_c(a,a') & \text{If } \|p-q\| \leq VdW(a,a') \\ \texttt{tab}_c(a,a') \frac{VdW(a,a')^2}{\|p-q\|^2} & \text{Otherwise} \end{cases}$$

where $VdW(a,a')$ is the sum of the Van der Walls radii of amino acids a, a'.

3) *Correlation component*: Let Λ_i be the torsional angle determined by the $C\alpha$ atoms $C\alpha_i, C\alpha_{i+1}, C\alpha_{i+2}, C\alpha_{i+3}$. Two statistically computed tables $\texttt{tab}_{t_1}$ and $\texttt{tab}_{t_2}$ are are used as auxiliary functions. The first one retrieves information from one torsional angle Λ_i parametrized by the type of the amino acids a_{i+1} and a_{i+2}. The second table retrieves information from the pair of torsional angles Λ_{i-1} and Λ_i independently of the amino acids' types. The correlation component is computed as follows (see [29] for the physical backgrounds of this component):

$$\texttt{Corr}(X, \vec{a}) = \sum_{i=2}^{n-4} \texttt{tab}_{t_1}(\Lambda_i, a_{i+1}, a_{i+2}) + \texttt{tab}_{t_2}(\Lambda_{i-1}, \Lambda_i)$$

Finally, we set:

$$E(X, \vec{a}) = w_1 \texttt{Hydro}(X) + w_2 \texttt{Cont}(X, \vec{a}) + w_3 \texttt{Corr}(X, \vec{a})$$

Experimental training on the *top500* dataset (trying to reduce the standard deviation of the energies calculated w.r.t. each component) produces the following values: $w_1 = 8, w_2 = 0.1, w_3 = 7$ for predicting α/β structures, and $w_1 = 8, w_2 = 22, w_3 = 7$ for predicting turns and loops. All auxiliary tables used are available in `http://www.cs.nmsu.edu/fiasco/`.

The literature on precise energy functions for assessing protein foldings is extensive (see, e.g., [31, 45, 41]). Our approach is completely parametric w.r.t. the energy function.

4 A Multi-Agent System Architecture

A *Multi-Agent System (MAS)* is a system composed of several agents in an environment, each with a certain degree of autonomy, and collaborating with other agents to solve a common problem [52]. These agents have only a local and partial view of the global system, which is too complex to be handled by a single agent.

[2] The van der Waals radius of an atom is the radius of an imaginary hard sphere which can be used to model atoms.

The basic structure of an agent is shown in Figure 6 (left). Usually, an agent receives information from the external environment, it processes the information, and it uses a predefined strategy to determine which action to perform—the action has the potential to affect the environment, by making changes to it.

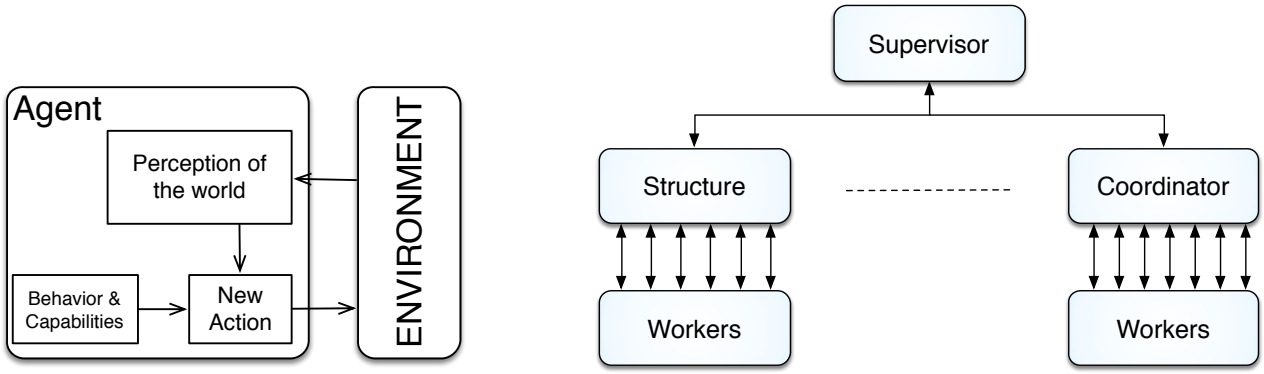

Figure 6: Basic structure of an agent and Multi-Agent System architecture

In this work, we use a multi-level MAS to compute the folding process. We define four types of agents: **(1)** A *Supervisor* agent, which is the highest level agent in the system, and it coordinates all of the other agents, **(2)** *Structure* agents, and **(3)** *Coordinator* agents, which are associated to the secondary structure elements of the target protein; both the structure and coordinator agents implement their own `search` methods; **(4)** *Worker* agents, which are associated to the $\mathcal{X}$ variables of the constraint model. The worker agents are in charge of propagating constraints and labeling the associated variables. The communication between agents passes through the *Supervisor* agent, since it is the only agent that has complete knowledge about the folding process during the complete search phase. Figure 6 (right) shows the architecture of the multi-agent system with the four types of agents. Let us consider a target protein of length n with s secondary structures elements involving p amino acids. s structure agents will be activated and coordinated by the supervisor agent. The supervisor agent will also activate 1 coordinator agent to take care of the remaining $s-1$ internal loops/turns and of the initial and final tails. More than one coordinator agents (e.g., one for each loop or turn) could be specified by the user to the supervisor agent as input parameter. n worker agents are created, one for each amino acid a_i, assigned as auxiliary agents for the corresponding structure/coordinator agent. Then, at each instant of the computation either p (if we are computing secondary structures elements) or $n-p$ (if we are coordinating structures looking for loops and turns) workers will run in parallel.

4.1 The Supervisor Agent

The *Supervisor* agent is the intermediary between the external world (e.g., user, external knowledge) and the other agents. It represents the highest level of abstraction in the framework and it holds global spatial information about the protein structure during the complete folding process. Its main task is to assign different sub-sequences of the primary sequence to the immediately underlying types of agents—the *coordinator* and the *structure* agents—and to supervise them, in order to guide the entire folding process towards a stable global configuration.

It is also responsible for creating the set of *worker* agents associated to the coordinator agents and to the structure agents. Moreover, it imposes both the search strategies and the constraints to be propagated on the other agents.

The supervisor agent sets a priority order among the agents. First, each "highly constrained" secondary structure (α-helix/β-sheet) is computed by the structure agents; afterwards, the coordinator agents are invoked to model the whole tertiary structure by moving *loops* or *turns*. In this work, we will restrict to a single coordinator agent. The supervisor agent must ensure also that the structures determined by each structure agent can be combined in order to obtain the global structure that can be effectively folded by the coordinator agent. To discard symmetric equivalent solutions, the secondary structure $a_i \cdots a_j$ with the lowest i among those computed is deterministically placed in the 3D space. The positions of all other points are obtained starting from it, as soon as the angles are assigned to variables x, y.

The location in the primary sequence and the type of the secondary structure elements is done by the supervisor agent using some of the capabilities of the secondary structure prediction algorithm *JNet* [17]. We remark that only the location is delegated to external knowledge, while the actual folding process is performed by the structure agents.

4.2 The Worker Agent

Worker agents are the lowest-level agents in the system. Each worker agent is associated either to a coordinator agent or to a structure agent and takes care of a particular amino acid a_i. In particular, it assigns all admissible values to the pair of variables x_i, y_i and executes consequent constraint propagation possibly rejecting inconsistent assignments. Precisely, propagation is performed to the portion of the constraint problem delegated to the structure/coordinator agent associated with it. A set of admissible structures (i.e., a set of structures that satisfy all the constraints the agent can see) is returned to the agent at the higher level.

This separation between high-level agents and worker agents allows us to not worry about how effectively the propagation is performed (e.g., choosing between CPU and GPU), but to solely invoke the propagation function of the worker agent from the structure or the coordinator agent, and choosing the best labeling according to an appropriate criterion and the type of constraint imposed on the variable associated to the worker agent.

4.3 The Structure Agent

This agent models secondary structure elements, such as α-helices and β-sheets. There is a structure agent for each secondary structure element of the target protein. Thus, each structure agent works exclusively on a specific part of the overall structure—i.e., it solves a folding problem on a limited subsequence of amino acids. It is not the task of the structure agent to maintains relations between the assigned subsequence and the rest of the structure (this will be handled by the coordinator agents).

The agent implements a *coordinate-wise gradient ascent* method (e.g., see [9, 10]). We call this search strategy *Iterated Conditional Mode* (*ICM*), implemented by Algorithm 2. This search strategy iterates to "greedily" refine a first feasible solution, as described in the next sections. This *greedy* strategy is suitable to secondary structure elements, since α-helices and β-sheets are "highly constrained" structures presenting a strong energy correlation between pairs of atoms; this implies a strong gap between geometrically stable structures and unfolded structures. Let us recall that our solution uses an independent system for secondary structure prediction to suggest type and locations of secondary structures.

4.4 The Coordinator Agent

The *coordinator agent* folds the protein by determining loops and turns that connect the secondary structure elements. x_i, y_i variables in already computed helices and sheets from the structure agents are already assigned; their related Cartesian variables $\mathcal{P}_i$ are instead unassigned: this allows to "move" structures as rigid objects. Since loops and turns are, in general, poorly structured, thus generating an intractable search space, the folding process is driven by a sampling of the search space. We compute the energy values of a set of structures to obtain more discriminant energy values. Coordinator agents adopt a search strategy that generalizes the one used by the structure agents and it is summarized in Algorithm 2 of Sect. 5.1. In particular, two sub-strategies have been implemented: a variant of the *Gibbs* [10] sampling search strategy, and a *Monte Carlo* search strategy (see Sections 5.2–5.3). Gibbs strategy is commonly used to solve the *Maximum A-Posteriori* (*MAP*) estimation problem. Recently, [36] have shown that COPs can be mapped to MAP estimation problems, therefore we have used it as a sampling algorithm to minimize $E(X, \vec{a})$ (Sect. 3.3).

5 General search schema

The structure and coordinator agents execute a large neighborhood search on a COP. The general schema of the LNS is implemented by Algorithm 1. The inputs provided to the algorithm are: the list $\mathcal{X}$ of $2n$ variables for angles, the list $\mathcal{P}$ of $15n$ point variables, and a list of *atom sizes* $\vec{d}$. Let us observe that each agent receives only a part of the whole protein, thus n is, in this case, not the number of amino acids of the whole protein, but only of a sub-part of it.

The first step of the algorithm (lines 3–7) is to post all the constraints required for the resolution of the problem—i.e., the `table` constraints that relates pairs of angles for variables x_i, y_i, the `sang` constraints that links point variables and corresponding FD variables, and an `alldistant` constraint over the set of point variables. Let us observe that this step is done only once, before the search begins.

The next step is to compute a first solution for the set of constraints (line 8)—this is a rather trivial task, e.g., by determining a solution that connects the most "straight" fragments; this allows us to satisfy the `alldistant` constraint in a trivial manner. The energy of this initial solution is computed in line 9.

Algorithm 1 Search($\mathcal{X}, \mathcal{P}, \vec{d}$)

```
1: - Constraints:
2: n ← |X|/2;
3: post table ⟨φ, ψ⟩ constraint on |X|;
4: post constraint: alldistant(P, d⃗);
5: for i ← 2 to n do
6:     post constraint: sang(i, X, P);
7: end for
8: S ← first_solution(X, P);
9: current_energy ← compute_energy(S);                ▷ Other parameters are omitted
10: - Search:
11: if (Agent = Structure) then
12:     repeat
13:         best_energy ← current_energy;
14:         (S, current_energy) ← icm(X, P);
15:     until (current_energy ≥ best_energy)
16: else if (Agent = Coordinator) ∧ (Search = Montecarlo) then
17:     repeat
18:         best_energy ← current_energy;
19:         (S, current_energy) ← mc(X, P);
20:     until (current_energy ≥ best_energy for k consecutive times)
21: else (Agent = Coordinator) ∧ (Search = Gibbs)
22:     S* ← S;
23:     best_energy ← current_energy;
24:     for i ← 1 to m do
25:         mc(X, P);                                 ▷ Prepare a new starting point (X, P)
26:         for t ← 1 to n_samples do
27:             (S, current_energy) ← gibbs(X, P);
28:             if current_energy < best_energy then
29:                 best_energy ← current_energy
30:                 S* ← S;
31:             end if
32:         end for
33:     end for
34:     S ← S*;
35: end if
36: return S;
```

The search phase starts at line 10. The code is slightly different in the case of structure agent and the case of coordination agent.

5.1 ICM

The search strategy adopted for the Structure agents is described by the loop in lines 12–15, that invokes the ICM algorithm (Algorithm 2) and it terminates when the energy value cannot be further improved. The ICM search strategy takes one pair of variables x_i, y_i at a time, it evaluates the energy for all their possible assignments, and they are assigned to the value that returns lowest energy. This task is performed by the

Algorithm 2 **icm**$(\mathcal{X}, \mathcal{P})$

1: **for** $i \leftarrow 1$ **to** n **do**
2: $wrk \leftarrow$ **get_worker_agent**(i);
3: $\mathcal{X} \leftarrow$ **choose_best_label**$(wrk, \mathcal{X}, \mathcal{P})$; ▷ Label x_i, y_i again
4: $\mathcal{S} \leftarrow$ **compute_structure**$[\mathcal{X}]$; ▷ Structure Updating
5: current_energy $\leftarrow$ **compute_energy**$(\mathcal{S})$;
6: **end for**
7: **return** $(\mathcal{S}, \text{current\_energy})$

worker agent i which runs the **choose_best_label** function (line 3). This function implements the strategy for selecting a new solution and it is performed in parallel on the GPU. Precisely, the variables x_i, y_i are assigned with all the elements (pair of angles) in their domains that satisfy the `table` constraint—variables $\vec{\mathcal{P}}_i$ are deterministically assigned by propagating the `sang` constraint.[3] All the resulting structures consistent with the `alldistant` constraint are energetically evaluated, and the one providing the minimum value is kept in $\mathcal{X}$.

Observe that the starting configuration is in the set of structure scanned by the ICM algorithm and, therefore, a solution is ensured to exists and the energy cannot increase. Although, in principle, two structures with the same minimum energetic value might emerge in the above step, the fine-grained energy model employed makes this situation highly unlikely; as a result, in practice the algorithm converges deterministically towards a local minimum. In the case in which two equivalent structures emerge, the first one encountered is chosen.

5.2 Monte Carlo

The Monte Carlo search strategy is implemented by the loop in lines 17–20 of Algorithm 1. Termination is forced after k consecutive loop iterations without any energy improvement, where k is a user-selectable parameter. The search space is sampled by the function **mc** described by the Algorithm 3.

Algorithm 3 **mc**$(\mathcal{X}, \mathcal{P})$

1: **for** $i \leftarrow 1$ **to** n **do**
2: $wrk \leftarrow$ **get_worker_agent**(i);
3: $\mathcal{X} \leftarrow$ **choose_best_label_random**$(wrk, \mathcal{X}, \mathcal{P})$; ▷ Label x_i, y_i again
4: $(t_{2i-1}, t_{2i}) \leftarrow (x_i, y_i)$;
5: **end for**
6: **if** (**solution_check**$(\vec{t}) =$ **true**) **then**
7: $\mathcal{X} \leftarrow \vec{t}$ ▷ Update consistently
8: **else**
9: $\mathcal{X} \leftarrow$ **assignment_from**$(\mathcal{S})$; ▷ Retrieve previous value (*)
10: **end if**
11: $\mathcal{S} \leftarrow$ **compute_structure**$[\mathcal{X}]$; ▷ Structure Updating
12: current_energy $\leftarrow$ **compute_energy**$(\mathcal{S})$; ▷ End of Coordination Agent
13: **return** $(\mathcal{S}, \text{current\_energy})$

The for loop scans through all the variables associated to the agent and it focuses on one pair of variables, x_i, y_i, at a time to determine a better structure. All the variables (related to loops and turns) assigned to the coordinator are uninstantiated. The worker agent implements the **choose_best_label_random** function (line 3). If x_i, y_i are not allocated to the coordinator agent, then the procedure halts immediately, leaving their values unchanged; otherwise, every admissible assignment to the pair x_i, y_i is attempted.[2] A fixed number of random assignments (samples) for the other variables x_j, y_j is attempted for each assignment of x_i, y_i. The values of x_i, y_i that is part of the sample returning the best energetic value is retained. In our experiments, we set the number of samples to 1,000. As before, the values to be assigned to the point variables $\mathcal{P}$ are obtained by propagation. The values t_{2i-1}, t_{2i} for x_i, y_i that allows us to obtain the best (i.e., minimal) energetic value are kept (lines 3–4). The list $\vec{t} = (t_1, \ldots, t_{2n})$ of these values constitutes a new possible solution. If this assignment

[3]For each new assignment the variables x_i, y_i and the variables $\vec{\mathcal{P}}_i, \vec{\mathcal{P}}_{i+1}, \ldots, \vec{\mathcal{P}}_{|X|}$ are uninstantiated.
[2]For each attempt all the variables assigned to the coordinator and all the variables $\mathcal{P}$ are uninstantiated.

(and the subsequent propagation to $\mathcal{P}$ driven by the `sang` constraint) satisfies the `alldistant` constraint (line 6—**solution_check**), the energy is evaluated and updated (line 7), otherwise, a new iteration of the loop in lines 17–20 of Algorithm 1 is started.

Variants of this method include also the possibility of worsening solutions with some fixed probability or with a probability decreasing over time (i.e., as in simulated annealing). The changes to the code to achieve these variants are minimal—these will be explored as future work. Moreover, if the test in line 6 fails, then some small changes in the solution will be attempted before deciding to restart from the previous solution. In particular, the algorithm considers other s solutions (default $s = 2$) sorted in ascending order of energy value before performing a new iteration of the loop in lines 17–20 (Algorithm 1). If there are any admissible assignment, the first one is used as new possible solution.

5.3 Gibbs sampling

Gibbs sampling mixes the ICM search strategy with the Monte Carlo sampling as follows. It uses two "meta" parameters: m (the number of *starting points*) that controls the number of iterations in lines 24–33 and *n_samples* (the number of sampling steps) that controls the number of iterations in lines 26–32. The coordinator agent invokes the function **gibbs** (line 27 of Algorithm 1) to sample one variable at a time based on the current assignment of the other variables. Each starting point is a *valid* random assignment of values to variables computed by invoking the **mc** function (see, Alg. 3). Algorithm 4 tries to improve the initial random point as follows. Initially, the current (random) structure is computed and energetically evaluated (lines 2–3). Then,

Algorithm 4 gibbs$(\mathcal{X}, \mathcal{P})$

```
1:  for i ← 1 to n do
2:      S ← compute_structure[X];
3:      current_energy ← compute_energy(S);
4:      wrk ← get_worker_agent(i);
5:      X ← choose_label_random(wrk, X, P);                ▷ Label x_i, y_i again
6:      if (Solution_check( X ) = true) then
7:          S* ← compute_structure[X];                     ▷ Auxiliary structure updating
8:          current_energy* ← compute_energy(S*);
9:          q ← min(1, exp(−current_energy*) / exp(−current_energy));
10:         r ← rand();
11:         if r ≤ q then
12:             X ← assignment_from(S);                    ▷ Reject the new state
13:         end if
14:     else
15:         X ← assignment_from(S);                        ▷ Retrieve previous value (*)
16:     end if
17: end for
18: S ← compute_structure[X];
19: current_energy ← compute_energy(S);
20: return (S, current_energy)
```

for each pairs of variables x_i, y_i, the corresponding worker agent invokes the function **choose_label_random** to randomly select a pairs of elements from their domains that satisfies the `table` constraint. If the selected assignment (and the subsequent propagation to $\mathcal{P}$ driven by the `sang` constraint) satisfies the `alldistant` constraint (line 6—**solution_check**), the new energy is evaluated on an auxiliary structure $\mathcal{S}^*$ (lines 7–8). The random choice is then accepted with a probability value q that depends on the ratio between the previous energy value and the current one (lines 6–14). If the random assignment does not satisfy the `alldistant` constraint the previous assignment is retrieved from $\mathcal{S}$ (line 15).

At each step of the Gibbs sampling algorithm the assignment is updated according to a *Metropolis* sampling process (i.e., a random assignment is accepted with probability that depends on the previous energy value—lines 9–10). Variants of this method includes also the possibility of changing the acceptance ratio in line 9

by multiplying the energy values by a *temperature factor* that varies over time. Moreover, the set of starting points can be partitioned in subsets of equal size, each represented by a different temperature factor.

5.4 The LNS general schema

Let us show how Algorithm 1, together with its auxiliary algorithms, implements a LNS schema. After one solution is computed (line 8 of Algorithm 1), a loop looking for improving solutions is entered.

In the case of the structure agent, the auxiliary procedure 2 generates sequentially n "neighborhoods", each obtained by releasing the assignments of the 17 variables corresponding to amino-acid i; neighbors are the set of assignments for such variables that satisfy all the constraints.

In the case of the coordination agent, we have two options: with Monte Carlo and with Gibbs. As far as Monte Carlo is concerned, the situation is similar to the structure agent case, with the difference that we do not optimize the neighbor at every step, but simply obtain an optimum of a sample of a subset of the neighbors. With a large number of samples (e.g. 1,000) this guarantees good results. As far as Gibbs is concerned, the situation is again similar: an improving solution (if any) is selected.

In this framework, GPU is used to speedup the exploration of large neighborhoods. The use of GPUs architectures is not new for speeding up LNS strategies. For example, a guideline for design and implementation of LNS strategies on GPUs is presented in [34].

6 Some Implementation Details

We implemented, in C++, a constraint solver that exploits parallelism on GPUs to explore the search space, following the previously described multi-level MAS model. As anticipated, locations of secondary structures are computed by a secondary structure prediction algorithm based on neural networks and sequence similarity alignments—specifically, the JNet application.[4] This descriptions can be provided by other secondary structure alignments tools (e.g., PSIPRED[5]) or by the user. In this section, we provide some implementation details about this solver.

6.1 Constraints

The `table` constraint is simply used for pairs of simultaneous assignments to variables x_i and y_i. Let us make some observations regarding how the other constraints can be efficiently handled on a GPU.

The `alldistant` constraint is checked on the whole set of point variables representing a protein structure. Using a sequential algorithm, the test of consistency of this constraint on a list of 3D coordinates of length n requires time $\mathcal{O}(n^2)$. In our implementation, we map this consistency check and constraint propagation to distinct cores of the GPU—by enabling each core to serve as representative of a different quintuple of atoms, that defines each amino acid of the structure. This allows us to reduce the complexity to $\mathcal{O}(n)$.

The propagation of the `sang` constraint relies on a kernel that is invoked with n threads, if the constraint is imposed on a list of $15n$ point variables. Each thread deals with a different quintuple of point variables, reducing the computation from $\mathcal{O}(n)$ to $\mathcal{O}(1)$ time. Further parallelism is obtained by calling the subroutine by all threads generating viable candidates within the implementation of the ICM search procedure.

6.2 Energy

The implementation of the energy function used in this work (see Sect. 3.3) as a CUDA kernel requires the introduction of two levels of parallelism:

1. Given a set of admissible structures, the energy value of each structure is calculated in parallel by a number of blocks equal to the size of the set, and
2. For a given structure, each energy field is calculated in parallel by a thread within the block.

[4] `http://www.compbio.dundee.ac.uk/www-jpred/advanced.html`
[5] `http://bioinf.cs.ucl.ac.uk/psipred/`

To obtain a linear time computation for the *contact* and the *hydrogen* potentials, we adopt the same strategy used for the `alldistant` constraint: if we consider a structure of length n, then we use n threads for both the *contact* potential and the *hydrogen bond* potential, while we use a single thread for the *correlation* potential. The total energy value is calculated in $\mathcal{O}(n)$ time. Further parallelism is obtained by calling the subroutine by all threads generating viable candidates.

6.3 Subroutines of the Search Algorithms

Subroutines are executed with references to an amino acid $i = 1, \ldots, n$. Let k be the number of pairs that can be assigned to variables x_i, y_i. Then the function **choose_best_label** is implemented by a CUDA kernel invoked with with a number of blocks equal to k and a number of threads equal to n.[6] Each kernel block $b = 1, \ldots, k$ considers an assignment $\langle \phi_b, \psi_b \rangle$ and each thread j computes the alignment for the j^{th} atoms $\vec{\mathcal{P}}_j$. Let us observe that the alignment of the atoms $\vec{\mathcal{P}}_j$ w.r.t. the previously placed atoms $\vec{\mathcal{P}}_{j-1}$ is deterministic, provided the positions of at least one list of points $\vec{\mathcal{P}}_l$, for $1 \leq l \leq n$ (see Sect. 3.2.3). Hence, the roto-translation matrices $\mathbf{Rot}(\cdot)$ can be computed independently by each thread once any 15-tuple of point $\vec{\mathcal{P}}_l$ variables is assigned. The cost of this subroutine is then $\mathcal{O}(1)$ time instead of $\mathcal{O}(kn)$ time of a sequential implementation.

A similar idea is applied for the subroutine **choose_best_label_random**. Let k be the number of pairs that can be assigned to variables x_i, y_i. Then a kernel CUDA is invoked a number of blocks $m \geq k$ equal to the number of random samplings to be performed.[7] The first k blocks performs exactly the same computation as the **choose_best_label** function—thus, simply exploring the impact of modifying only the angles associated to the x_i, y_i variables. The remaining $m - k$ blocks are subdivided in k groups, each corresponding to a different possible assignment of the variables x_i, y_i; these $m - k$ blocks are in charge of the actual random sampling. Each one of these blocks starts with the initial configuration of the variables x_i, y_i—determined by which of the k groups the block belongs to. The block then continues by determining the random assignments for all the other pairs of variables x_j, y_j, with $i \neq j$ (among those variables allocated to the coordinator agent), as described in Sect. 2. The roto-translation matrices $\mathbf{Rot}(\cdot)$ differ every time the alignment for a new list of point variables $\vec{\mathcal{P}}_{j+1}$ is computed based on the previous list of point variables $\vec{\mathcal{P}}_j$. The block makes use of two threads to speed-up such computation. In particular, we use one thread to perform the sequential alignment on the lists of point variables $\vec{\mathcal{P}}_1, \ldots, \vec{\mathcal{P}}_{j-1}$, while the second thread performs the sequential alignment on the lists of point variables $\vec{\mathcal{P}}_{j+1}, \ldots, \vec{\mathcal{P}}_n$, according to the two angles $\langle \phi, \psi \rangle$ randomly selected for the variables x_j, y_j. This subroutine runs in $\mathcal{O}(n)$ time using this organization in blocks and threads, instead of the $\mathcal{O}(k \cdot v \cdot m)$ required by a sequential implementation, where v is the number of variables assigned to the coordinator agent, and m is the number of random samplings.

As mentioned earlier, given a set of instantiated point variables, calculated by either the **choose_best_label** function or the **choose_best_label_random** function, the energy values of the corresponding structures are calculated in parallel on the GPU. Once a list of energy values (for each of the structures determined by the **choose_best_label**/**choose_best_label_random**) has been determined, we delegate the computation of the minimal energy to the host—using a simple linear scan of the list of energy values of the various structures. We do not use a logarithmic reduction to compute the minimal energy, since the cost of invoking a kernel CUDA plus the costs of the memory transactions between host and device would exceed the time required by the sequential scan.

As far as the Gibbs sampling strategy is concerned, the `for` of lines 24–33 of Alg. 1 is delegated to a CUDA kernel invoked with m blocks, where each block is assigned to a different starting point. Starting points are computed as described above by the **choose_best_label_random** function considering $k = 1$ and forcing the random assignment of the variables associated to the agent (i.e., it will be performed the same computation performed by one of the blocks $\geq (m - k)$ of the **choose_best_label_random** function). The for-loop that iterates on the number of samples (Alg. 1—line 26-32) and the for-loop that iterates on the number of worker agents (Alg. 4—lines 1-17) are kept as sequential loops due to the correlation of consecutive samples (i.e., consecutive samples define a Markov chain). Energy values are calculated in parallel on the GPU at each iteration of the two nested loops on all the set of (updated) starting points.

[6] The average number of the pairs of angles allowed is $k = 107$.

[7] The average number of the pairs of angles allowed is 389. We have tested values of k from 500 to 3000 and we have observed that the best behavior for the system is for values for k close to 1000.

6.4 General details about CUDA

We present some details related to the implementation of the CUDA kernels described in the previous sections. In particular, due to the features of the architectural model of CUDA, we must consider three main aspects that can affect the performance of the parallel computations: **(1)** The maximum number of threads per block; **(2)** The maximum number of threads that can be physically executed in parallel on each processor of the GPU (also known as the *warp* size); **(3)** The information stored on the device memory and the copies of data to and from the host memory.

In this paper we assume that we are using a (typical) hardware where the maximum number of threads per block is limited to 1024, and the size of a *warp* is 32. The first restriction could potentially limit the maximum size of the target protein to 1024, when we use one thread per amino acid (e.g., to implement the `alldistant` constraint). To avoid such restriction we can split the computation in multiple executions of the same kernel. For each invocation we simply consider a different window of 1024 consecutive amino acids, until we cover the whole protein. However, protein typical size is less than 1024, so this further stage is normally not needed.

The second restriction is important when we allow threads of the same warp to diverge to different computational branches. Since kernel instructions are issued not to each thread, but to each warp, to solve the divergences the compiler of CUDA generates code that will run sequentially one branch after the other, causing a delay in the execution of the entire warp. We solved this problem by splitting the parallel computation of different parts of the kernel code among warps of 32 threads. For example, considering the energy function instead of $2n + 2$ threads per block, we invoke the kernel with $2m + 64$ threads per block, where $m = \lceil \frac{n}{32} \rceil$. Hence, $2m$ threads are used to compute the *contact* potential and the *hydrogen bond* potential, and 64 threads are used to compute the *correlation* and *torsional* potential.

The final aspect regards the optimization of the memory usage in order to achieve maximum memory throughput. CUDA has different types of memory spaces. Each thread block has access to a small amount of a fast *shared* memory within the scope of the block. In turn, all threads have access to the same *global* memory. Global memory is slower than shared memory but it can store more data. Since applications should strive to minimize data transfers between the host and the device (i.e., data transfers with low bandwidth), we reserve in the global memory an array of size equal to the maximum number of structures expected for the sampling of the coordinator agent multiplied the size of a structure. Moreover, we reserve an array of Boolean values for the admissible structures and an array of doubles for the energy values. Each kernel receives the number of structures to be considered and the pointers to the arrays in the global memory, in order to properly overwrite them considering only the memory area affected by the kernel function. The memory transfers to and from the CPU are made at each labeling step by copying the array of structures produced by the propagation of constraints and the array of the energy values. Again, only the elements affected by the computation of the kernel are transferred into the host memory.

To optimize the computation on the device, we store all the structures to rotate, the structures on which to perform the consistency checks for the `alldistant` constraint, and those on which to calculate the energy values in the shared memory of the GPU—paying particular care to not exceed the maximum size available for the device in use. The shared memory can be a limitation when we manage a large number of structures or very long proteins. Nevertheless, this is not a problem if we consider that the size of our domains is about 300, and the proteins are typically 200-300 amino acids long. These numbers are compatible with the characteristics of CUDA (e.g., maximum number of threads per block), which makes this architecture particularly efficient on our model.

7 Results

We report some experimental results obtained from the GPU implementation of the multi-agent system (briefly, GMAS). We run our experiments on a CPU AMD Opteron 2.3GHz, 132 GB memory, Linux 3.7.10-1.16-desktop x86 64, and GPU GeForce GTX TITAN, 14 SMs, 875MHz, 6 GB global memory, CUDA 5.0 with compute capability 3.5. To evaluate the speedup gained from exploiting parallelism on the GPU, we implement a sequential version of the multi-agent system (briefly, CMAS). Since coordinator agents use randomized search strategies, we report the results averaged over 20 runs per protein, as well as their standard deviation (sd). Computational times are reported in second. To assess the quality of our predictions we use the Root Mean Square Deviation (RMSD) as an indicator of how close is the predicted structure w.r.t. the correspondent (known) structure deposited in the protein data bank (the lower the better). Energy and RMSD values are

averaged on the sets of results produced by both the CPU and the GPU implementations. "SUp" denotes Speedup in tables.

For following experiments we considered two different benchmark set of proteins selected from the *Protein Structure Database* ([7]). The first benchmark set, *BS1*, is composed by 27 proteins divided in 9 sets based on their length. In particular for $x = 1, \ldots, 9$ we we randomly selected an α protein, a β protein, and an $\alpha\beta$ protein of length $n \in \{10x + 1, \ldots, 10(x + 1)\}$.

The second benchmark set, *BS2*, is composed by 12 proteins of length that ranges from 125 to 200 residues, split into 4 subsets (where $n = 125, 150, 175, 200$). For each subset we randomly selected an α protein, a β protein, and an $\alpha\beta$ protein. The symbol *"*"* marks some proteins' *ID*s that represents difficult targets due to their *supersecondary structure* conformation. A *supersecondary* structure is a compact three-dimensional protein structure of several adjacent elements of secondary structure that is smaller than a protein domain or a subunit. For both benchmark sets we will consider a structure agent per secondary structure element and one coordinator agent (default options).

In all CMAS/GMAS experiments we set a global timeout of 5000 seconds. This situation is reported as "5000 (-)" in the tables. Let us observe that in these cases, the best solution found in the time allowed is returned.

In Section 7.4 we analyze the system on a long protein as a case of study; for this test we used more coordinator agents. In Section 7.3 for the comparison with I-TASSER and FIASCO we considered the benchmark sets presented in the publications concerning those tools. Globally the GMAS tool has been tested on 65 proteins. The CPU and the GPU versions of the multi-agent system, the set of proteins, the input files, and the best results computed by the MAS tool can be found at `http://www.cs.nmsu.edu/fiasco/`.

7.1 GPU vs. CPU

Structure agents and secondary structure elements: In Table 1 and 2 we compare GMAS and CMAS w.r.t. the times required by Structure agents to fold α-helices and β-sheets for the benchmark sets BS1 and BS2, respectively. SS denotes the total length of the secondary structures in the protein of length n. Let us observe that Structure agents use the ICM algorithm to fold secondary structures.

Protein ID	Type	SS/n	CPU	GPU	SUp
2CZP	α	11/15	0.065 (0.001)	0.011 (0.0)	5.9
1LE0	β	6/12	0.006 (0.001)	0.010 (0.0)	0.6
2H2D	$\alpha\beta$	9/18	0.003 (0.0)	0.007 (0.0)	0.4
2L5R	α	20/23	0.367 (0.001)	0.060 (0.0)	6.1
1E0N	β	13/27	0.103 (0.001)	0.037 (0.0)	2.7
1YYP	$\alpha\beta$	12/31	0.046 (0.001)	0.019 (0.0)	2.4
1ZDD	α	25/35	0.675 (0.001)	0.062 (0.0)	10.8
1E0L	β	12/37	0.054 (0.001)	0.035 (0.0)	1.5
1PNH	$\alpha\beta$	19/31	0.181 (0.003)	0.061 (0.0)	2.9
2K9D	α	33/44	0.973 (0.001)	0.075 (0.0)	12.9
1YWI	β	16/41	0.125 (0.001)	0.050 (0.0)	2.5
1HYM	$\alpha\beta$	18/45	0.575 (0.001)	0.072 (0.0)	7.9
1YZM	α	41/51	2.729 (0.001)	0.149 (0.0)	18.3
2CRT	β	27/60	1.014 (0.001)	0.176 (0.0)	5.7
2HBB	$\alpha\beta$	28/51	1.101 (0.011)	0.069 (0.0)	15.9
1AIL	α	59/69	6.986 (0.005)	0.395 (0.005)	17.6
1PWT	β	31/61	2.117 (0.001)	0.141 (0.0)	15.0
2IGD	$\alpha\beta$	40/61	4.267 (0.021)	0.345 (0.002)	12.3
1OF9	α	53/77	4.816 (0.010)	0.242 (0.002)	19.9
1SPK	β	27/72	0.994 (0.001)	0.111 (0.0)	8.9
1VIG	$\alpha\beta$	42/71	3.022 (0.001)	0.213 (0.002)	14.1
1I11	α	44/81	2.707 (0.001)	0.202 (0.0)	13.4
1TEN	β	48/87	7.258 (0.036)	0.556 (0.004)	13.0
1DCJ	$\alpha\beta$	43/81	2.943 (0.002)	0.204 (0.0)	14.4
1JHG	α	82/100	16.12 (0.003)	0.557 (0.003)	28.9
1WHM	β	38/92	7.309 (0.002)	0.342 (0.006)	21.3
2CJO	$\alpha\beta$	41/97	7.955 (0.001)	0.296 (0.005)	26.8

Table 1: CPU vs GPU: Secondary Structure predictions for the set BS1.

Protein ID	Type	SS/n	CPU	GPU	SUp
1A0B	α	98/125	40.15 (0.103)	1.223 (0.007)	32.8
1H10	β	56/125	24.12 (0.083)	0.730 (0.003)	33.0
1F98	$\alpha\beta$	74/125	18.96 (0.010)	0.574 (0.002)	33.0
2CJ5	α	116/150	71.78 (0.021)	1.155 (0.012)	62.1
1STB	β	72/150	36.46 (0.040)	0.748 (0.005)	48.7
1LEO	$\alpha\beta$	89/150	39.15 (0.157)	0.763 (0.001)	51.3
1BGD	α	113/175	80.73 (0.283)	1.311 (0.005)	61.5
1FNL	β	92/175	48.24 (0.024)	1.073 (0.008)	44.9
1T8A	$\alpha\beta$	130/175	95.44 (0.017)	1.592 (0.006)	59.9
1IB1	α	165/200	113.4 (0.108)	2.851 (0.002)	39.7
2GH2	β	100/200	63.60 (0.115)	1.264 (0.008)	50.3
1RR9	$\alpha\beta$	114/200	86.50 (0.023)	1.147 (0.005)	75.4

Table 2: CPU vs GPU: Secondary Structure predictions for the set BS2.

The speedups increase as SS increases. In particular, the higher speedups in most sets are obtained on Structure agents associated to proteins of type α. This is due to the highly constrained structure of the helices where the (rather greedy) search strategy reaches a local minima without iterating many times on the set of variables associated to the agent. β-sheets are less constrained structures and they require more iterations to converge to the local minima. For the GMAS implementation point of view, more iterations leads to more information exchanged between host and device (e.g., memory copies) resulting in a decrease in performance.

AB-Initio Prediction using Monte Carlo: Ab-Initio prediction is performed by the Supervisor agent by managing Structure agents and Coordinator agents. In Table 3 we report the comparison between CMAS and GMAS considering one Coordinator agent using the Monte Carlo search strategy on the benchmark set BS1. Running time are inclusive of secondary structure prediction (by structure agents), while the external computation using JNet of the possible segments where looking for secondary structures is not considered. For each of the following experiments regarding Monte Carlo we considered a sample set of 1000 structures, a number of improving steps $k = 4$,[8] and a time-out limit of 10800 seconds (3 hours). Speedups range from 4.5 to 17.3. For each protein we also report the best result found in 20 runs in terms of RMSD (column "Best). Let us observe that RMSD values are rather low: for proteins of length 100 we are in an average of 5Å.

AB-Initio Prediction using Gibbs: Coordinator agents can use the Gibbs sampling algorithm to model loops and turns and hence to fold the entire protein. In terms of time, the Gibbs sampling algorithm has a more stable behavior than Monte Carlo since the former runs for a fixed number of iterations, while the latter runs until a local minimum (or a given time-out) is reached. Computation of secondary structures are performed by structure agents as described in the previous experiment. In Table 4 we report the comparisons between CMAS and GMAS using the Gibbs sampling strategy for the Coordinator agent, considering a number of samples $n\_samples = 50$, and $m = 1000$ starting points.[9]

The standard deviations of running times are smaller than those obtained using Monte Carlo. Speedups are higher than in the previous experiments; this is due to the fact that Gibbs algorithm updates the initial set of starting points in parallel considering one structure at a time for each sampling step, while Monte Carlo produces a new sampling set for each variable associated to the agent at each iteration step.

Time vs. Number of Samplings: The time needed to fold the structures and their quality are two factors that are strongly correlated to the number of samples when using the Gibbs sampling search strategy. Although it is quite difficult to relate the upper bound on the number of samples with an upper bound on the quality of the results (e.g., see [36]), it is easy to study how the computational time varies w.r.t. the number of samples and the length of the proteins. In particular, since the coordinator agent invokes the `gibbs` function for a fixed number of times (see Alg. 1, lines 26–35), the computational time varies linearly w.r.t. the number of samples. We report this analysis in Figure 7 for the computationally most demanding protein of each subset, varying the number of samples from 1 to 50, and comparing GMAS with CMAS.

In order to study how the quality of the solution changes w.r.t. the number of samples we selected the longer protein of the benchmark set (i.e., 2CJO, with $n = 97$ but with 48 amino acids delegated to the Coordinator agent) and we varied the number of samples from 1 to 50. Table 5 reports the results in terms of RMSD and standard deviation w.r.t. the number of samples.

[8] Parameters found experimentally considering a trade-off between time and quality of predictions.

[9] The number of sampling steps has been found experimentally and it has been chosen as a good compromise between compu-

ID	n	CPU (sd)	GPU (sd)	SUp	RMSD (sd)	Energy (sd)	Best
2CZP	15	0.559 (0.128)	0.046 (0.010)	12.1	0.8 (0.0)	-490.5 (2.55)	0.8
1LE0	12	1.923 (0.176)	0.337 (0.020)	5.7	1.2 (0.2)	-344.2 (23.50)	0.5
2H2D	18	1.815 (0.118)	0.275 (0.073)	6.6	1.2 (0.0)	-263.3 (6.76)	1.2
2L5R	23	1.603 (0.222)	0.188 (0.028)	8.5	1.4 (0.0)	-1482.54 (0.00)	1.4
1E0N	27	63.80 (14.180)	10.73 (1.463)	5.9	2.4 (0.6)	-1945.00 (41.17)	1.5
1YYP	22	8.479 (2.262)	1.473 (0.482)	5.7	1.7 (0.3)	-1205.24 (31.23)	1.2
1ZDD	35	39.40 (7.801)	4.343 (1.263)	9.0	1.1 (0.2)	-2472.0 (27.79)	0.9
1E0L	37	20.76 (3.230)	1.816 (0.257)	11.4	2.7 (0.6)	-959.366 (70.40)	1.9
1PNH	31	24.15 (3.037)	2.865 (0.301)	8.4	2.7 (0.3)	-2611.61(40.20)	2.3
2K9D	44	87.46 (10.81)	13.67 (5.427)	6.3	1.5 (0.4)	-6162.405 (150.29)	0.9
1YWI	41	32.23 (4.690)	3.523 (0.959)	9.1	3.4 (0.3)	-1295.636 (14.75)	2.7
1HYM	45	81.10 (5.193)	9.309 (1.033)	8.7	3.5 (0.2)	-2258.224 (71.90)	3.0
1YZM	51	32.45 (4.361)	1.871 (0.279)	17.3	2.6 (0.4)	-4963.852 (63.70)	1.6
2CRT	60	1213 (166.45)	163.2 (20.70)	7.4	3.7 (0.6)	-9514.056 (271.25)	2.6
2HBB*	51	476.3 (37.86)	64.42 (6.383)	7.3	3.6 (0.5)	-6175.155 (217.69)	3.0
1AIL	69	88.10 (15.37)	6.546 (0.593)	13.4	3.2 (0.6)	-10443.04 (251.59)	2.3
1PWT	61	512.8 (45.63)	66.61 (12.23)	7.6	4.1 (0.6)	-7983.511 (238.57)	3.2
2IGD*	61	681.4 (50.13)	92.79 (12.55)	7.3	6.2 (0.8)	-7928.462 (218.74)	5.0
1OF9*	77	750.6 (82.15)	133.9 (32.54)	5.6	4.3 (0.7)	-17063.19 (480.68)	3.3
1SPK	72	1343 (149.27)	174.4 (28.00)	7.7	4.1 (0.4)	-8457.154 (334.07)	3.6
1VIG	71	977.1 (107.20)	164.8 (34.63)	5.9	4.9 (0.7)	-9295.76 (296.07)	3.7
1I11	81	296.7 (23.22)	38.70 (10.75)	5.6	4.4 (0.4)	-7852.049 (251.77)	3.8
1TEN*	87	2995 (466.99)	1014 (86.20)	2.9	5.2 (0.8)	-13955.71 (686.41)	4.1
1DCJ*	81	1590 (139.66)	310.7 (43.92)	5.1	4.5 (0.4)	-15398.04 (606.98)	3.7
1JHG	100	1168 (123.86)	187.0 (33.59)	6.2	5.7 (1.0)	-23687.56 (360.45)	4.0
1WHM	92	2796 (414.802)	611.5 (63.32)	4.5	4.5 (0.7)	-17220.98 (525.289)	3.3
2CJO*	97	9195 (547.22)	1988 (162.6)	4.6	4.6 (0.2)	-15785.77 (318.45)	4.2

Table 3: Time (sec.), quality, and energy values averaged on 20 runs for the set of proteins BS1. Coordinator agent uses *Monte Carlo* algorithm.

Protein ID	n.	CPU (sd)	GPU (sd)	SUp	RMSD (sd)	Energy (sd)	Best
2CZP	15	1.243 (0.001)	0.219 (0.002)	5.6	0.8 (0.0)	-490.461 (1.965)	0.8
1LE0	12	2.674 (0.001)	0.557 (0.009)	4.8	1.4 (0.1)	-308.881 (4.903)	1.3
2H2D	18	2.210 (0.018)	0.513 (0.009)	4.3	1.5 (0.3)	235.4091 (7.062)	1.1
2L5R	23	5.009 (0.154)	0.356 (0.006)	14.0	1.4 (0.0)	-1482.54 (0.0)	1.4
1E0N	27	69.69 (0.167)	3.379 (0.025)	20.6	2.7 (0.6)	-1724.791 (41.558)	1.7
1YYP	22	19.12 (0.14)	1.450 (0.009)	13.1	1.8 (0.4)	-1203.736 (11.868)	1.2
1ZDD	35	57.64 (0.09)	2.639 (0.013)	21.8	1.1 (0.1)	-2470.327 (26.436)	0.9
1E0L	37	27.06 (0.459)	1.895 (0.014)	14.2	2.5 (0.4)	-894.690 (55.465)	2.0
1PNH	31	33.99 (0.104)	1.793 (0.020)	18.9	2.3 (0.1)	-2538.936 (25.54)	2.1
2K9D	44	144.0 (1.282)	4.687 (0.020)	30.7	1.7 (0.7)	-5883.502 (102.288)	0.9
1YWI	41	41.48 (0.221)	2.251 (0.011)	18.4	3.4 (0.3)	-1168.549 (13.060)	2.9
1HYM	45	74.58 (1.250)	3.478 (0.049)	21.4	3.5 (0.5)	-2037.445 (39.651)	2.8
1YZM	51	56.04 (0.719)	1.800 (0.015)	31.3	2.8 (0.2)	-4993.436 (27.578)	2.4
2CRT	60	665.7 (10.23)	17.93 (0.320)	37.1	4.0 (0.7)	-8289.048 (166.152)	2.7
2HBB*	51	372.6 (1.628)	11.68 (0.116)	31.9	4.0 (0.2)	-5309.193 (101.03)	3.7
1AIL	69	156.0 (3.019)	3.889 (0.017)	40.1	3.5 (0.3)	-11275.98 (190.78)	3.2
1PWT	61	474.4 (0.345)	12.70 (0.611)	37.3	4.7 (0.6)	-6942.941 (146.546)	3.6
2IGD*	61	550.4 (2.870)	14.15 (0.231)	38.8	5.1 (0.9)	-7029.251 (86.943)	4.4
1OF9*	77	780.5 (10.87)	18.50 (0.429)	42.1	4.0 (0.8)	-14739.77 (335.616)	2.5
1SPK	72	700.3 (1.521)	18.42 (0.353)	38.0	4.6 (0.5)	7071.708 (80.60)	4.0
1VIG	71	750.0 (2.161)	23.00 (0.288)	32.6	5.1 (1.0)	-8015.638 (130.803)	3.4
1I11	81	433.1 (1.244)	10.11 (0.122)	42.8	3.9 (0.6)	-7544.959 (114.85)	2.8
1TEN*	87	1841 (2.742)	50.93 (1.611)	36.1	5.5 (0.5)	-12576.83 (413.913)	4.6
1DCJ*	81	1021 (1.246)	26.18 (1.077)	38.9	5.2 (0.6)	-13061.12 (174.81)	3.9
1JHG	100	1462 (1.344)	32.63 (1.255)	44.8	4.9 (1.3)	-21332.2 (258.721)	3.3
1WHM	92	1035 (1.281)	37.19 (0.262)	27.8	4.9 (0.6)	-13953 (237.5671)	4.3
2CJO*	97	2024 (2.149)	56.41 (1.274)	35.8	5.2 (0.6)	-12964.17 (170.607)	4.3

Table 4: Time (sec.), quality, and energy values averaged on 20 runs for the set of proteins BS1. Coordinator agent uses *Gibbs* sampling algorithm.

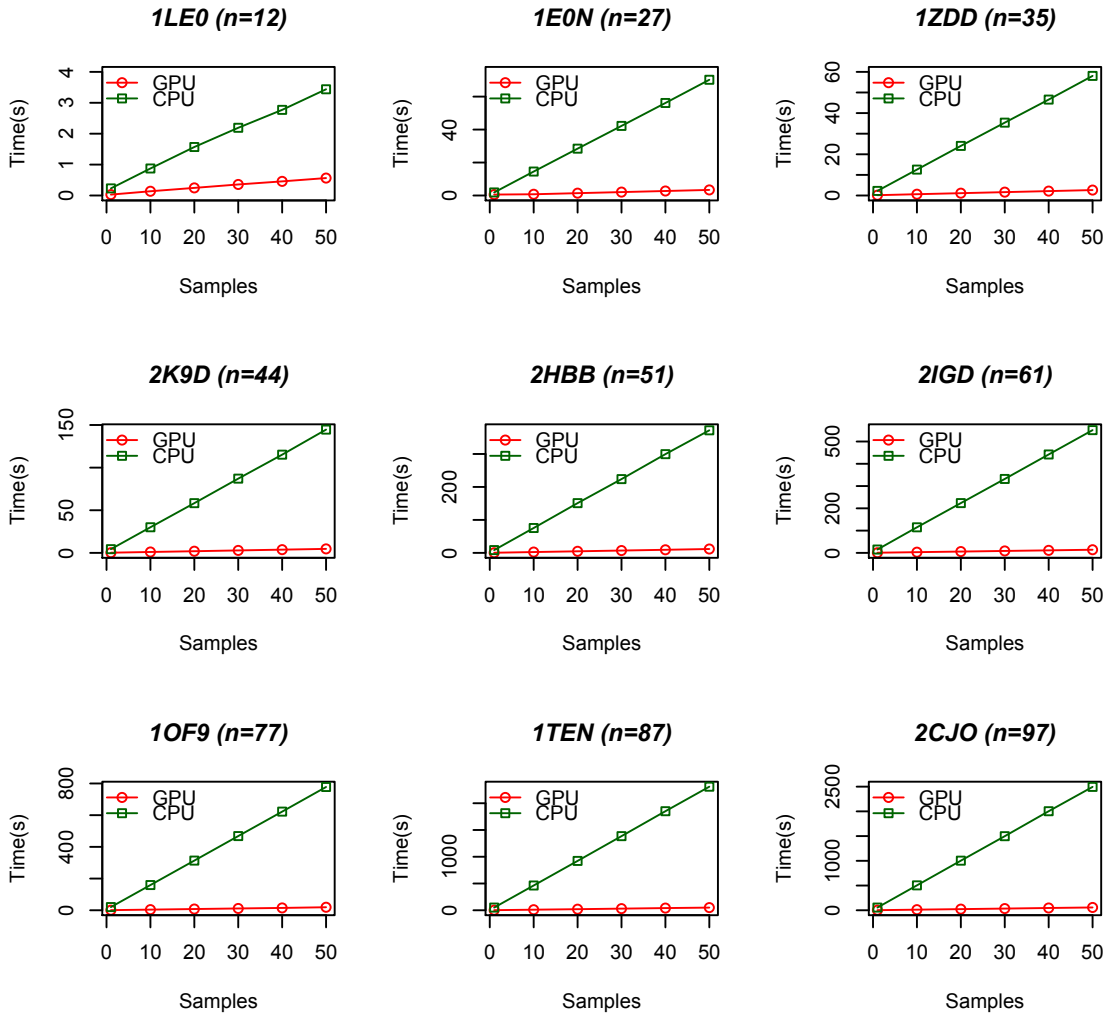

Figure 7: Time vs number of samples for the Gibbs sampling algorithm

Protein ID	Number of Samples					
	1	10	20	30	40	50
2CJO	6.3 (0.9)	5.8 (0.2)	5.5 (0.4)	5.4 (0.6)	5.4 (0.5)	5.2 (0.6)

Table 5: Quality w.r.t. number of samples for the Gibbs sampling strategy.

Our choice of setting 50 as upper bound on the number of sampling for the benchmarks set BS1 is motivated by the two following observations: (1) for $n \geq 30$ the improvements on the quality of the structures are relatively small, and (2) 2CJO can be implicitly considered as a representative protein for all the other targets in BS1, since it has the larger set of amino acids assigned to a coordinator agent (51 residues). Therefore, we have conjectured that this value is suitable for the whole set. Other, not reported, experiments have confirmed this hypothesis. Let us conclude this analysis by observing that the observed linearity guarantees that using less than 50 sampling steps would preserve the speedups.

Quality evaluation: using RMSD as Objective Function: The quality of the predicted structure strongly depends on the energy function adopted in the model. The energy function can be changed as a black box and without affecting the overall structure of the system. In this test, we evaluated the differences in terms of RMSD between the Gibbs sampling and the Monte Carlo algorithm using the RMSD w.r.t. the native known structure as objective function for both the structure and the coordinator agents. Of course this function cannot be used for still unknown proteins; however we experiment it in order to see if our tool is, in principle, able to compute the native structure if the "real" protein energy function would be know. Table 6 shows the results in terms of average and best RMSD computed for the benchmark set is BS1. Best results are reported in **boldfont**.

Protein ID	n	Gibbs (sd)	Best	MC (sd)	Best
2CZP	15	1.0 (0.0)	**1.0**	1.0 (0.0)	**1.0**
1LE0	12	1.0 (0.0)	1.0	0.9 (0.9)	**0.9**
2H2D	18	0.5 (0.0)	**0.5**	1.2 (0.0)	1.2
2L5R	23	1.4 (0.0)	1.4	1.1 (0.1)	**1.0**
1E0N	27	1.3 (0.2)	1.0	0.9 (0.0)	**0.8**
1YYP	22	0.9 (0.0)	0.9	1.5 (0.1)	**0.8**
1ZDD	35	1.6 (0.1)	1.5	1.5 (0.1)	**1.4**
1E0L	37	3.4 (0.1)	3.1	3.5 (0.3)	**2.8**
1PNH	31	2.8 (0.4)	**2.1**	2.9 (0.2)	2.7
2K9D	44	2.0 (0.3)	**1.5**	1.9 (0.1)	1.7
1YWI	41	2.5 (0.4)	**1.9**	2.5 (0.2)	2.2
1HYM	45	2.4 (0.3)	**1.6**	2.4 (0.2)	2.1
1YZM	51	1.5 (0.0)	1.5	1.5 (0.1)	**1.4**
2CRT	60	4.3 (0.3)	**3.9**	4.5 (0.2)	4.1
2HBB*	51	2.6 (0.3)	2.1	1.8 (0.3)	**1.4**
1AIL	69	2.0 (0.3)	**1.5**	2.8 (0.7)	1.7
1PWT	61	3.5 (0.3)	2.9	2.7 (0.4)	**2.1**
2IGD*	61	3.0 (0.2)	**2.6**	2.9 (0.3)	**2.6**
1OF9*	77	2.2 (0.3)	2.1	1.7 (0.1)	**1.5**
1SPK	72	4.3 (0.5)	3.8	4.2 (0.3)	**3.7**
1VIG	71	5.0 (0.5)	**4.2**	5.8 (0.4)	4.9
1I11	81	2.9 (0.4)	2.3	2.4 (0.1)	**2.2**
1TEN*	87	6.3 (0.4)	**5.8**	6.5 (0.4)	**5.8**
1DCJ*	81	4.6 (0.7)	**3.8**	4.3 (0.3)	**3.8**
1JHG	100	5.7 (0.5)	5.1	5.7 (0.6)	**4.3**
1WHM	92	5.2 (0.9)	**4.1**	4.8 (0.2)	4.5
2CJO*	97	4.8 (0.5)	4.1	4.2 (0.4)	**3.5**

Table 6: Quality evaluation: RMSD as objective function.

As expected, the average quality of the results is better than using the original energy function, since both search strategies try to minimize the spatial difference between the native structure and the target. Nevertheless, RMSD values are not close to zero since the variables' domains are created to be used for predicting unknown structures. Therefore, they do not necessary contain the true pair of angles of the amino acids of each target.

Let us observe that an objective function based on the RMSD value can be adopted for new search strategies. For example, minimization of the RMSD distance between (parts of) the target and a corresponding set of templates can be useful in template-based modeling techniques [27].

Adding constraints while preserving speedup: In this section we show that the system is highly modular and that we can introduce additional geometric constraints that allows to reduce the search space while preserving the speedup.

tational time and quality of predictions, as discussed in what follows.

We consider the case of modeling the side-chain of each amino acid. This can be done by defining a new constraint that relates the position of each $C\alpha$ atom with the group R defined on it. The `centroid (CG)` constraint enforces a relation among four triples of real variables $\vec{p}_1, \vec{p}_2, \vec{p}_3$, and $\vec{p}_4$. This relation establishes the value to assign to the variables $\vec{p}_4$ representing the coordinates of the side chain defined on the carbon atom $C\alpha_i$, given the bend angle formed by the carbon atoms $\vec{p}_1 \mapsto C\alpha_{i-1}, \vec{p}_2 \mapsto C\alpha_i$, and $\vec{p}_3 \mapsto C\alpha_{i+1}$, and the average $C\alpha_i$–side chain distance [28]. The coordinates of $\vec{p}_1, \vec{p}_2$, and $\vec{p}_3$ are already present in our modeling, while the coordinates of $\vec{p}_4$ are considered only here. Moreover, this constraint checks the minimum distance between side chains and all the other atoms of the structure in order to avoid steric clashes, as in the case of the `alldistant` constraint. Note that, using a sequential algorithm, it is possible to check the consistency of this constraint for a given assignment of values to the variables in P in time $\mathcal{O}(n^2)$. We adopt the same strategy used for the `alldistant` constraint to obtain $\mathcal{O}(n)$ time in the parallel implementation.

In Table 7 (resp., 8) we report the times for CMAS and GMAS on the protein set BS1 for the Monte Carlo (reps., Gibbs sampling strategies), using the `alldistant` constraint and the `CG` constraint.

Protein ID	n	CPU (sd)	GPU (sd)	SUp	RMSD (sd)	Energy (sd)	Best
2CZP	15	0.760 (0.158)	0.075 (0.017)	10.1	0.8 (0.0)	-487.009 (0.0)	0.8
1LE0	12	2.225 (0.247	0.489 (0.047)	4.5	1.4 (0.1)	-310.5714 (15.03)	1.3
2H2D	18	2.112 (0.340)	0.425 (0.042)	4.9	1.3 (0.2)	-258.0266 (4.655)	1.1
2L5R	23	2.072 (0.525)	0.281 (0.109)	7.3	1.4 (0.0)	-1482.54 (0.0)	1.4
1E0N	27	72.46 (9.620)	14.18 (2.458)	5.1	2.3 (0.4)	-1723.076 (49.3655)	1.7
1YYP	22	10.94 (2.793)	1.695 (0.114)	6.4	1.4 (0.2)	-1190.033 (36.27)	1.2
1ZDD	35	45.799 (7.169)	5.284 (1.378)	8.6	1.7 (0.2)	-2272.059 (55.53058)	1.5
1E0L	37	24.85 (3.285)	4.493 (0.271)	5.5	2.6 (0.6)	-882.369 (37.0802)	1.8
1PNH	31	27.25 (4.602)	3.370 (0.387)	8.0	3.4 (0.6)	-2518.261 (52.135)	2.1
2K9D	44	95.42 (9.547)	12.68 (1.798)	7.5	3.0 (1.3)	-5258.283 (114.113)	1.4
1YWI	41	29.07 (5.566)	3.940 (1.688)	7.3	3.1 (0.5)	-1152.215 (40.8021)	2.2
1HYM	45	97.96 (18.16)	10.29 (0.964)	9.5	3.5 (0.3)	-2113.299 (85.986)	2.9
1YZM	51	36.69 (10.11)	2.406 (0.330)	15.2	2.7 (0.3)	-4393.806 (39.639)	2.4
2CRT	60	1280 (110.4)	187.5 (22.14)	6.8	4.0 (0.3)	-8346.851 (264.8791)	3.5
2HBB*	51	562.2 (91.53)	67.66 (12.38)	8.3	3.9 (0.8)	-5471.265 (67.6792)	2.8
1AIL	69	132.5 (9.236)	7.575 (0.653)	17.4	3.8 (0.8)	-8880.202 (227.2679)	2.7
1PWT	61	554.5 (40.66)	67.75 (4.649)	8.1	4.1 (0.6)	-7034.326 (178.9961)	3.0
2IGD*	61	717.7 (135.3)	82.09 (12.19)	8.7	5.3 (1.0)	-7126.439 (224.9547)	4.1
1OF9*	77	768.2 (100.4)	140.0 (33.79)	5.4	3.5 (0.7)	-14429.41 (342.4358)	2.6
1SPK	72	1317 (103.3)	175.7 (9.452)	7.4	4.2 (0.5)	-7189.066 (264.136)	3.4
1VIG	71	1176 (184.4)	160.3 (34.90)	7.3	4.7 (0.9)	-8118.327 (181.7243)	3.3
1I11	81	322.7 (18.45)	49.91 (5.924)	6.4	4.0 (0.7)	-6870.417 (151.4574)	3.1
1TEN*	87	5330 (564.8)	1128 (148.1)	4.7	6.0 (0.8)	-13505.96 (204.369)	5.1
1DCJ*	81	1623 (209.8)	292.0 (52.30)	5.5	5.2 (0.7)	-12866.59 (445.7158)	4.4
1JHG	100	1417 (101.7)	179.8 (28.56)	7.8	6.1 (0.9)	-20293.91 (404.9333)	4.8
1WHM	92	2829 (250.4)	968.9 (189.5)	2.9	5.1 (0.8)	-14676.57 (484.0556)	3.8
2CJO*	97	8248 (748.3)	1967 (262.8)	4.1	5.4 (0.8)	-13916.12 (395.8187)	4.4

Table 7: Time (sec.), quality, and energy values averaged on 20 runs for some proteins of different length and type. Coordinator agents use *Monte Carlo* algorithm to explore conformations. CG constraint has been enabled.

7.2 Longer Proteins

After experimented the speedup of GMAS w.r.t. CMAS, in this section we show the results of GMAS on the set of larger proteins BS2. Table 9 shows the results in terms of time, RMSD, energy, and best RMSD when using the Gibbs algorithm for the Coordinator agent (1 Coordinator agent, 50 sampling steps). Times vary from 40.4 to 376.6 seconds, while the best RMSD values are always under 8.0Å. In Table 10 the CG constraint is added to the encoding. Quality of results are slightly decreased due to the poor approximation of the side chain with a single atom, while computational times are reduced since more structures have been pruned.

7.3 Comparison with other Systems

Comparison with Rosetta: We compared the quality of the proposed tool in terms of RMSD against the state-of-the-art *Rosetta* tool, initially presented in [46] and continuously evolved since then. For each

Protein ID	n	CPU (sd)	GPU (sd)	SUp	RMSD (sd)	Energy (sd)	Best
2CZP	15	2.006 (0.001)	0.314 (0.029)	6.3	0.8 (0.0)	-491.458 (1.022)	0.8
1LE0	12	4.114 (0.007)	0.806 (0.077)	5.1	1.4 (0.1)	-300.8306 (6.789)	1.3
2H2D	18	3.300 (0.039)	0.758 (0.060)	4.3	1.7 (0.7)	-244.392 (7.862)	0.9
2L5R	23	6.787 (0.070)	0.521 (0.047)	13.0	1.4 (0.0)	-1482.54 (0.000)	1.4
1E0N	27	93.61 (10.66)	4.964 (0.118)	18.8	3.0 (0.6)	-1517.871 (28.7422)	2.3
1YYP	22	25.71 (0.062)	2.100 (0.099)	12.2	1.7 (0.5)	-1183.739 (9.5)	1.1
1ZDD	35	81.41 (0.500)	3.384 (0.065)	24.0	1.6 (0.1)	-2378.678 (61.27)	1.5
1E0L	37	39.52 (0.782)	2.692 (0.017)	14.6	2.8 (0.6)	-849.1835 (35.66)	2.0
1PNH	31	47.02 (0.205)	2.451 (0.034)	19.1	2.8 (0.4)	-2440.618 (55.42)	2.1
2K9D	44	186.9 (15.24)	5.311 (0.069)	35.1	2.0 (0.4)	-5191.226 (247.499)	1.6
1YWI	41	52.96 (0.211)	3.203 (0.090)	16.5	3.0 (0.5)	-1072.859 (16.79)	2.3
1HYM	45	110.7 (0.198)	4.646 (0.093)	23.8	3.9 (0.7)	-1905.429 (62.30)	2.9
1YZM	51	70.85 (0.770)	2.575 (0.007)	27.5	2.6 (0.1)	-4533.8 (46.809)	2.5
2CRT	60	636.3 (4.732)	18.29 (0.331)	34.7	3.8 (0.6)	-7252.04 (247.893)	3.1
2HBB*	51	578.2 (4.624)	13.77 (0.208)	41.9	4.3 (0.4)	-4818.446 (54.610)	3.6
1AIL	69	226.8 (0.356)	6.403 (0.651)	35.4	3.7 (0.7)	-9842.159 (227.2839)	2.4
1PWT	61	524.7 (173.7)	11.60 (1.573)	45.2	4.2 (0.7)	-6173.182 (109.042)	3.3
2IGD*	61	361.7 (29.61)	16.94 (0.347)	21.3	5.4 (0.9)	-6185.173 (151.216)	4.0
1OF9*	77	1077 (23.35)	18.18 (0.615)	59.2	4.8 (0.9)	-12970 (212.057)	3.0
1SPK	72	554.2 (28.9)	15.08 (0.597)	36.7	4.9 (0.5)	-6151.49 (128.4445)	4.1
1VIG	71	644.6 (19.86)	22.70 (0.765)	28.3	5.0 (0.6)	-6995.402 (223.483)	3.9
1I11	81	461.9 (106.3)	13.56 (0.177)	34.0	4.0 (0.7)	-6747.406 (94.80)	2.8
1TEN*	87	3348 (15.58)	62.29 (0.354)	53.7	6.3 (0.9)	-11030.24 (311.69)	4.8
1DCJ*	81	1065 (392.9)	19.51 (1.196)	54.5	5.9 (0.9)	-11330.29 (271.17)	3.5
1JHG	100	1556 (63.96)	15.80 (0.347)	98.4	5.2 (0.8)	-19074.76 (579.9539)	4.0
1WHM	92	1879 (0.431)	30.18 (1.693)	62.2	5.2 (1.0)	-12364.63 (199.9923)	3.7
2CJO*	97	1795 (148.7)	45.30 (0.742)	39.6	5.8 (0.9)	-11197.55 (188.3018)	4.7

Table 8: Time (sec.), quality, and energy values averaged on 20 runs for some proteins of different length and type. Coordinator agents use *Gibbs* sampling algorithm to explore conformations. CG constraint has been enabled.

Protein ID	Time (sd)	RMSD (sd)	Energy (sd)	Best RMSD
1A0B	40.44 (0.99)	5.2 (1.1)	-23592.9 (1007.557)	3.5
1H10*	69.18 (3.278)	7.6 (1.0)	-14041.22 (1010.967)	6.9
1F98*	84.37 (3.410)	6.9 (0.9)	-21683.44 (760.408)	6.0
2CJ5	100.6 (4.546)	6.4 (0.8)	-35302.44 (1062.648)	5.2
1STB*	118.0 (5.452)	7.5 (1.1)	-19563.05 (910.6381)	5.6
1LEO	101.6 (8.208)	6.7 (0.9)	-32627.28 (909.583)	5.4
1BGD*	135.5 (7.360)	8.1 (1.5)	-47544.61 (1441.902)	5.2
1FNL*	376.6 (26.34)	8.4 (0.7)	-40839.9 (1424.246)	7.2
1T8A	214.8 (12.56)	7.9 (1.0)	-44482.65 (1107.889)	6.3
1IB1	208.7 (10.80)	8.4 (1.0)	-50007.91 (1047.775)	6.4
2GH2*	84.75 (0.851)	9.8 (1.5)	-38321.71 (1283.365)	7.5
1RR9*	298.8 (17.93)	8.4 (1.1)	-45074.02 (1959.957)	7.0

Table 9: Longer proteins $(125, 150, 175, 200)$: time and quality evaluation using Gibbs sampling.

Protein ID	Time (sd)	RMSD (sd)	Energy (sd)	Best RMSD
1A0B	32.33 (1.424)	7.8 (1.3)	-21292.37 (708.0099)	5.6
1H10*	49.76 (2.648)	7.3 (1.1)	-13354.67 (184.2466)	5.0
1F98*	51.56 (2.795)	7.1 (1.0)	-19554.71 (751.1157)	5.9
2CJ5	75.33 (7.619)	6.5 (1.2)	-30636.26 (218.7721)	5.6
1STB*	58.00 (3.399)	7.2 (1.5)	-17567.47 (862.4277)	4.5
1LEO	60.32 (5.187)	7.3 (0.8)	-28359.66 (658.0844)	6.1
1BGD*	83.57 (9.395)	9.0 (1.2)	-42451.74 (942.5947)	7.8
1FNL*	101.9 (15.99)	10.5 (1.7)	-34714.2 (2204.668)	7.9
1T8A	113.6 (4.960)	8.8 (1.7)	-39357.61 (862.8464)	5.5
1IB1	112.6 (10.07)	9.1 (1.2)	-42539.18 (1545.187)	7.3
2GH2*	118.5 (20.11)	12.4 (1.2)	-33937.24 (2317.069)	11.1
1RR9*	144.3 (18.84)	8.8 (1.2)	-38644.9 (1474.511)	6.9

Table 10: Longer proteins $(125, 150, 175, 200)$: time and quality evaluation using Gibbs sampling with the CG constraint enabled.

Protein ID	Time (sd)	RMSD (sd)	Energy (sd)	Best RMSD
1A0B	285.9 (61.19)	4.1 (0.4)	-25588.58 (717.2202)	2.7
1H10*	1693 (300.3)	5.8 (0.6)	-18914.5 (459.4581)	5.1
1F98*	2987 (598.4)	5.8 (0.9)	-26607.36 (721.812)	4.7
2CJ5	1583 (230.9)	7.0 (1.4)	-41436.0 (1558.731)	5.5
1STB*	5392 (625.9)	6.5 (0.2)	-25676.32 (1207.134)	6.2
1LEO	5145 (609.2)	5.8 (0.3)	-40749.62 (1038.819)	5.4
1BGD*	5000 (-)	6.2 (1.3)	-38185.23 (1199.362)	5.1
1FNL*	5000 (-)	6.0 (0.4)	-45450.2 (1014.597)	5.7
1T8A	5000 (-)	4.7 (0.7)	-39630.5 (922.132)	4.2
1IB1	4436 (543.1)	3.4 (0.2)	-43938.1 (907.0643)	3.1
2GH2*	5000 (-)	7.2 (0.7)	-49686.63 (503.952)	6.7
1RR9*	5000 (-)	5.8 (1.2)	-45133.57 (640.239)	4.6

Table 11: Longer proteins $(125, 150, 175, 200)$: time and quality evaluation using the Monte Carlo search strategy.

Protein ID	Time (sd)	RMSD (sd)	Energy (sd)	Best RMSD
1A0B	240.6 (44.56)	7.1 (1.8)	-22661.39 (242.603)	5.1
1H10*	1659 (247.8)	6.4 (0.5)	-13641.52 (5259.107)	5.9
1F98*	2287 (240.3)	6.3 (0.8)	-22783.95 (530.0175)	5.2
2CJ5	1547 (149.6)	13.5 (2.9)	-35356.96 (660.8399)	8.6
1STB*	4891 (372.8)	7.0 (0.6)	-21351.42 (440.864)	6.1
1LEO	4125 (402.6)	6.1 (0.9)	-33733.4 (164.6203)	5.2
1BGD*		6.6 (1.1)	-40615.83 (870.601)	5.4
1FNL*	5000 (-)	6.3 (1.0)	-44714.47 (382.754)	5.1
1T8A		4.6 (0.5)	-38881.8 (997.4146)	4.1
1IB1	3625 (538.4)	4.8 (0.3)	-43085.2 (934.407)	4.5
2GH2*	5000 (-)	9.2 (0.4)	-47081.13 (1051.74)	9.5
1RR9*	5000 (-)	6.3 (1.0)	-40543.97 (1206.53)	5.6

Table 12: Longer proteins $(125, 150, 175, 200)$: time and quality evaluation using the Monte Carlo search strategy and CG constraint.

protein of benchmark set SB1 and SB2 we built the dictionary for 3 and 9 amino acid long peptides previous sequence alignment using the PSIPRED online server (`http://bioinf.cs.ucl.ac.uk/psipred`). We followed the examples in the Rosetta distribution. Let us observe that our tool uses e a generic database of angles whereas Rosetta uses a database of fragments created based on the target sequence. To be as fair as possible in comparing the tools we run them considering their default settings. Moreover, we run the Rosetta tool on a host machine equipped with 8 processors Intel(R) Core(TM) i7-2600 CPU, 3.40GHz.

Table 13 show the results in terms of RMSD and time as given by Rosetta. For each protein we report also the RMSD corresponding to the best structure found by Rosetta and the RMSD corresponding to best structure found by GMAS among all the previous experiments using Monte Carlo (MC) and Gibbs sampling as well as the corresponding times. Best results are reported in **boldfont**.

The quality of the structures predicted by GMAS is in line with the results of Rosetta. On the other hand, Rosetta tends to be faster on longer proteins (e.g., ≥ 150 amino acids). This is due to several factors such as a better energy function that can lead sooner to a local minimum, the computation of the fragments database for each target, and the various heuristics encoded in that tool during the years.

Comparison with I-TASSER: We compared GMAS with another state-of-the-art protein structure prediction system, namely the Iterative TASSER (*I-TASSER*) tool [54, 40]. I-TASSER is a threading-based system that builds protein structures from primary sequences considering already known homologous proteins. Table 14 presents the comparison between I-TASSER and GMAS. For comparing I-TASSER with GMAS we considered the complete benchmark set of 16 proteins presented in [53] (benchmark 1). For each protein we report the best result found by I-TASSER (as reported in their work), as well as the best results found by GMAS using Monte Carlo without the GC constraint and Gibbs with the CG constraints, that previous experiments proved to be the best combinations. The running time of I-TASSER computations are those reported in the 2007 paper. However, we have re-launched some of them with our machine obtaining similar (slightly worse) results with their timeout of five hours. The RMSD results shows that our tool (considering our best result with the two options) has results comparable to I-TASSER (and obtained in shorter time).

		Rosetta			GMAS			
Protein ID	n	RMSD (sd)	Time (sd)	Best	MC	Time (sd)	Gibbs	Time (sd)
2CZP	15	0.4 (0.1)	2.700 (0.483)	**0.2**	0.8	0.046 (0.010)	0.8	0.219 (0.002)
1LE0	12	0.7 (0.2)	2.600 (0.516)	**0.3**	0.5	0.337 (0.020)	1.3	0.557 (0.009)
2H2D	18	1.3 (0.3)	3.428 (0.534)	**0.9**	1.1	0.425 (0.042)	**0.9**	0.758 (0.060)
2L5R	23	1.1 (0.0)	3.500 (0.547)	**1.1**	1.4	0.188 (0.028)	1.4	0.356 (0.006)
1E0N	27	2.7 (0.4)	7.100 (1.523)	2.0	**1.5**	10.73 (1.463)	1.7	3.379 (0.025)
1YYP	22	2.5 (0.4)	7.857 (0.690)	2.1	1.2	1.473 (0.482)	**1.1**	2.100 (0.099)
1ZDD	35	1.3 (0.4)	11.00 (0.942)	**0.8**	0.9	4.343 (1.263)	0.9	2.639 (0.013)
1E0L	37	3.5 (0.6)	11.70 (2.540)	2.3	**1.8**	4.493 (0.271)	2.0	1.895 (0.014)
1PNH	31	3.2 (0.8)	11.20 (1.549)	**1.8**	2.1	3.370 (0.387)	2.1	1.793 (0.020)
2K9D	44	2.8 (1.3)	12.30 (2.057)	1.6	**0.9**	13.67 (5.427)	**0.9**	4.687 (0.020)
1YWI	41	2.5 (0.4)	9.222 (1.715)	2.8	**2.2**	3.940 (1.688)	2.3	3.203 (0.090)
1HYM	45	4.2 (0.9)	15.10 (0.567)	3.1	2.9	10.29 (0.964)	**2.8**	3.478 (0.049)
1YZM	51	0.9 (0.4)	10.75 (1.035)	**0.5**	1.6	1.871 (0.279)	2.4	1.800 (0.015)
2CRT	60	4.5 (0.8)	20.75 (1.035)	3.5	**2.6**	163.2 (20.70)	2.7	17.93 (0.320)
2HBB	51	3.4 (0.7)	17.11 (1.452)	**2.2**	2.8	67.66 (12.38)	3.6	13.77 (0.208)
1AIL	69	4.1 (1.4)	26.33 (0.707)	**1.7**	2.3	6.546 (0.593)	2.4	6.403 (0.651)
1PWT	61	3.9 (0.7)	23.12 (0.640)	**2.8**	3.0	67.75 (4.649)	3.3	11.60 (1.573)
2IGD	61	5.6 (0.6)	20.75 (3.615)	4.7	4.1	82.09 (12.19)	**4.0**	16.94 (0.347)
1OF9	77	11.5 (0.1)	18.33 (1.032)	10.9	2.6	140.0 (33.79)	**2.5**	18.50 (0.429)
1SPK	72	4.6 (0.6)	25.12 (1.246)	3.7	**3.4**	175.7 (9.452)	4.0	18.42 (0.353)
1VIG	71	5.5 (1.2)	25.25 (1.281)	4.1	**3.3**	160.3 (34.90)	3.4	23.00 (0.288)
1I11	81	4.8 (1.1)	25.00 (1.322)	2.9	3.1	49.91 (5.924)	**2.8**	10.11 (0.122)
1TEN	87	6.8 (0.6)	34.50 (1.511)	5.7	**4.1**	1014 (86.20)	4.6	50.93 (1.611)
1DCJ	81	5.3 (0.9)	30.87 (1.807)	4.0	3.7	310.7 (43.92)	**3.5**	19.51 (1.196)
1JHG	100	5.0 (0.7)	38.25 (6.273)	3.8	4.0	187.0 (33.59)	**3.3**	32.63 (1.255)
1WHM	92	6.4 (0.5)	35.00 (1.772)	5.8	**3.3**	611.5 (63.32)	3.7	30.18 (1.693)
2CJO	97	4.7 (0.8)	38.62 (2.924)	**3.5**	4.2	1988 (162.6)	4.3	56.41 (1.274)
1A0B	125	6.0 (1.4)	45.25 (2.964)	4.4	**2.7**	285.9 (61.19)	3.5	40.44 (0.99)
1H10	125	7.1 (1.4)	47.50 (5.830)	5.4	5.1	1693 (300.3)	**5.0**	49.76 (2.648)
1F98	125	5.8 (1.0)	55.00 (3.681)	**4.5**	4.7	2987 (598.4)	5.9	51.56 (2.795)
2CJ5	150	7.1 (1.9)	62.77 (5.540)	**4.8**	5.5	285.9 (61.19)	5.2	100.6 (4.546)
1STB	150	6.3 (0.7)	62.80 (5.329)	4.8	6.1	4891 (372.8)	**4.5**	58.00 (3.399)
1LEO	150	6.5 (1.0)	69.50 (5.380)	**4.9**	5.2	4125 (402.6)	5.4	101.6 (8.208)
1BGD	175	7.8 (1.2)	59.10 (9.085)	5.8	**5.1**	5000 (-)	5.2	135.5 (7.360)
1FNL	175	6.8 (1.2)	60.70 (4.056)	**5.1**	**5.1**	5000 (-)	7.2	376.6 (26.34)
1T8A	175	6.8 (1.2)	86.40 (6.380)	5.0	**4.1**	5000 (-)	5.5	113.6 (4.960)
1IB1	200	7.8 (1.5)	129.0 (9.297)	6.4	**3.1**	4436 (543.1)	**6.4**	208.7 (10.80)
2GH2	200	8.0 (1.0)	101.9 (6.297)	6.9	**6.7**	5000 (-)	7.5	84.75 (0.851)
1RR9	200	7.1 (1.8)	86.40 (12.33)	4.8	**4.6**	5000 (-)	6.9	144.3 (18.84)

Table 13: Quality evaluation: best results of GMAS systems against Rosetta.

		I-TASSER	GMAS			
			MC		Gibbs	
ID	n	RMSD	RMSD	Time (sd)	RMSD	Time (sd)
1B72_A	49	3.1	**2.4**	20.56 (3.863)	3.1	7.437 (0.081)
1SHF_A	59	**1.7**	3.0	144.2 (11.63)	3.7	17.92 (0.096)
1TIF	59	7.0	3.9	45.11 (4.230)	**3.6**	9.350 (0.665)
2REB_2	60	4.7	4.1	17.78 (3.393)	**2.4**	7.023 (0.018)
1R69	61	**1.9**	3.5	124.5 (15.06)	3.5	15.58 (0.164)
1CSP	67	**2.1**	4.4	248.0 (46.72)	4.8	22.26 (1.124)
1DI2_A	69	**2.3**	4.8	41.23 (5.382)	5.4	13.09 (0.273)
1N0U_A4	69	4.4	4.7	128.3 (16.03)	**3.7**	19.63 (0.950)
1MLA_2	70	**2.7**	3.6	61.75 (14.83)	4.6	16.95 (0.241)
1AF7	72	4.2	**3.4**	60.39 (7.870)	**3.4**	14.92 (0.109)
1OGW_A	72	**1.1**	4.3	485.1 (66.16)	3.5	30.40 (0.518)
1DCJ_A	73	10.0	**3.7**	267.5 (42.02)	3.9	24.40 (1.267)
1DTJ_A	74	**1.7**	4.3	247.8 (23.99)	3.1	25.41 (0.172)
1O2F_B	77	5.2	3.6	228.5 (16.91)	**3.4**	22.82 (1.262)
1MKY_A3	81	4.5	**3.8**	545.7 (88.16)	4.9	35.20 (0.551)
1TIG	88	**4.4**	4.7	196.4 (30.92)	5.6	31.73 (0.301)

Table 14: Quality evaluation: best results of GMAS systems against I-TASSER.

Comparison with FIASCO: We compared GMAS with another CP-based tool FIASCO ([13] —Table 15). FIASCO is a C++-based constraint solver targeted at modeling a general class of protein structure studies that relies on fragment assembly techniques. The benchmark set used for these experiments is the same used in [13]. For GMAS we report the best results among 20 runs for each combination using Monte Carlo and Gibbs without the CG constraints, that previous experiments proved to be the best combinations. Let us observe that GMAS is a clear winner in this case.

		FIASCO		**GMAS**			
				MC		Gibbs	
ID	n	RMSD	Time	RMSD	Time (sd)	RMSD	Time (sd)
1ZDD	35	2.0	685.2	**0.9**	4.343 (1.263)	**0.9**	2.639 (0.013)
2GP8	40	6.2	376.8	**1.3**	1.916 (0.226)	**1.3**	2.070 (0.180)
2K9D	44	2.5	513.0	**0.9**	13.67 (5.427)	**0.9**	4.687 (0.020)
1ENH	54	8.2	1900	2.3	13.75 (5.731)	**1.3**	10.86 (1.301)
2IGD	61	10.5	1588	4.1	82.09 (12.19)	**4.0**	16.94 (0.347)
1SN1	63	5.5	889.2	4.1	47.35 (10.41)	**3.0**	46.40 (9.726)
1AIL	69	4.5	267.6	**2.3**	6.546 (0.593)	2.4	6.403 (0.651)
1B4R	79	6.1	504.6	4.1	462.1 (97.58)	**4.0**	479.8 (47.01)
1JHG	100	4.5	270.0	4.0	187.0 (33.59)	**3.3**	32.63 (1.255)

Table 15: Quality evaluation: best results of GMAS systems against FIASCO.

7.4 A Case of Study: 3BHI

In this section we consider a specific long protein as representative for a case of study to assess the capabilities of our solver on "hard" proteins. We selected the protein *3BHI* with $n = 276$, and its secondary structure contains 6 α-helices, 3 β-sheets, and 7 turns. First of all we compute the offsets of the secondary structure elements on the primary sequence using JNet that will produce a text file containing the desired alignment:

```
SSGIHVALVTGGNKGIGLAIVRDLCRLFSGDVVLTARDV...
----EEEE-------HHHHHHHHHHHHHH--EEEEE--H...
```

JNet requires few seconds to generate the alignment and hence it does not affect the overall computational time. However, this input file might be also generated by other tools or on-line servers (e.g., `http://bioinf.cs.ucl.ac.uk/psipred/`). Once the alignment has been generated we run the solver specifying the Gibbs algorithm as search strategy for the Coordinator agents, with 150 sampling steps, and the file containing the alignment as input file. We used the Gibbs option in the coordinator that proved to be the faster for long proteins in our previous experiments. We increased the number of samples to 150 since we experimentally observed a convergence of the quality of the solutions after 150 samples, running the system considering 50, 100, 150 and 200 samples. Results are reported in Table 16 - `Default` row.

Experiment	Time (sd)	RMSD (sd)	Energy (sd)	Best RMSD
`Default`	3885 (178.4)	12.15 (2.0)	-83854.05 (3060.509)	10.8
`Default-CG`	908.2 (60.83)	11.40 (1.4)	-74065.95 (2979.682)	10.1
`Multi-Coordinators`	1317 (2.146)	10.175 (2.1)	-86866.9 (4447.796)	8.3
`Multi-Coordinators-CG`	939.8 (3.024)	11.17 (1.2)	-76466.71 (5154.601)	9.8

Table 16: Case of Study: 3BHI (276 amino acids)

Then we have introduced the CG constraint: `Default-CG` row shows that time is reduced from 3885 to 908.2 seconds. Let us observe that also the quality of the solution is improved from 12.15Å to 11.40Å although the energy value is increased. This means that the energy function and, in particular, the weights of the energy components are not completely precise.

To further improve the prediction we used more than one coordinator agent, in particular one for each sequence of amino acids between a pair of consecutive secondary structures. Results are reported in column `Multi-Coordinators` column. With these new settings we improved the quality of the predictions (i.e., the structure is more compact since loops and turns are better simulated) and we reduced the computational time (since coordinator agents are associated to few variables). Finally we use both these facilities and results are reported in `Multi-Coordinators-CG` row. The time is further reduced but the average RMSD is increased due to the side chain centroids that forbid structures that are too compact.

7.5 Comparing different GPUs

In this section, we compare three different GPUs with different computational capabilities in order to evaluate how much the architecture affects execution times. We report the details of the various hardware used (CPU and GPU). (1) is the hardware used in previous experiments: TITAN: *Host* AMD Opteron Processor 6376, 2.3GHz, *Device* GeForce GTX TITAN, 2688 cores@875MHz (14SM); (2) Quadro: *Host* Intel Core i7-3770, 3.4GHz, *Device* Quadro 600, 96 cores@640 MHz (2SM); (3) Tesla: *Host* Intel Xeon, 2.4GHz, *Device* Tesla C2075, 448 cores@1.15 Ghz (14SM).

Table 17 compare these three different architectures on three different predictions using the MC algorithm and Gibbs sampling considering default settings, without the CG constraint. We report running time and speed-ups in any of the three machines. It emerges clearly that the approach benefits from better GPUs either in the running time or in the speed-up. Let us observe that we still have speedups also for the Quadro device that is the less powerful graphic card among the three devices, whit 2 SM and 96 cores (whereas the host has 3.4GHz w.r.t. 2.3 GHz of TITAN host and Tesla host). Speedups are in the order of $2 \sim 3$ when considering the Gibbs sampling algorithm that has been proven in the previous experiments to be the more effective for longer proteins.

Protein ID	n	**Time**									
		TITAN			Quadro			Tesla			
		CMAS (sd)	GMAS (sd)	SUp	CMAS (sd)	GMAS (sd)	SUp	CMAS (sd)	GMAS (sd)	SUp	
1E0N	27	63.80	10.73 (1.463)	5.9	59.92 (5.050)	152.3 (27.07)	0.3	87.92 (12.24)	23.98 (4.681)	3.6	
2HBB	51	476.3	64.42 (6.383)	7.3	329.9 (45.69)	873.6 (146.9)	0.3	523.2 (52.55)	163.0 (19.51)	3.2	
1JHG	100	1168	187.0 (33.59)	6.2	1175 (82.02)	1719 (414.7)	0.6	1414 (137.3)	270.2 (76.11)	5.2	Monte Carlo
1E0N	27	69.69	3.379 (0.025)	20.6	67.80 (0.037)	33.16 (0.458)	2.0	74.89 (0.147)	7.362 (0.073)	10.1	
2HBB	51	372.6	11.68 (0.116)	31.9	405.8 (1.574)	150.9 (2.008)	2.6	433.3 (0.800)	26.72 (0.246)	16.2	
1JHG	100	1462	32.63 (1.255)	44.8	1602 (11.15)	433.8 (12.02)	3.6	1674 (0.957)	71.96 (2.411)	23.2	Gibbs

Table 17: Comparison between different GPUs - Monte Carlo and Gibbs sampling.

8 Conclusion

In this paper we presented a novel perspective for addressing the Protein Structure Prediction Problem. We used a declarative approach for ab-initio simulation, implementing a multi agent system. Moreover, we used a GPU architecture to efficiently explore the search space and to propagate constraints. The results are remarkable. The use of GPU allows us to obtain speedups up to 75. The system can fold proteins of small-medium length with a low computational time (in the order of minutes) and quality of the results comparable with the state-of-the-art systems.

As future work, we plan to improve the search strategies of the agents. In particular, we shall try to make use of dynamic priorities between agents. These priorities are set by the supervisor agent, and dynamically modified on the base of the current partial results computed by the structure and the coordinator agents. Moreover, the prediction of the area where secondary structures might occur, currently delegated to the external tool JNet, will be incorporated in the system (as initial module of the supervisor agent).

An interesting direction is the study of an implementation of the multi-agent system on a multi-GPU environment. Different GPUs can be assigned to different structure and coordinator agents, which can exchange information during the whole folding process. The entire search phase is governed by the supervisor agent which assigns jobs to and retrieves results from processes running on different GPUs. We are also developing a visual interface that allows the use of the tool outside the community of computer scientists.

Acknowledgements

We would like to thank our colleagues and friends Alessandro Dal Palù, Ferdinando Fioretto, and Federico Fogolari for their advices and useful comments in various stages of this work.

This work was supported by the INdAM-GNCS under Grants 2010, 2011, and 2014; and by NSF under Grants CBET-1401639, HRD-1345232, CNS-1337884, and DGE-0947465.

References

[1] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell*. Garland Science, 2007.

[2] C. B. Anfinsen. Principles that Govern the Folding of Protein Chains. *Science*, 181:223–230, 1973.

[3] Rolf Backofen and Sebastian Will. A Constraint-Based Approach to Fast and Exact Structure Prediction in Three-Dimensional Protein Models. *Constraints*, 11(1):5–30, 2006.

[4] D. Baker and A. Sali. Protein Structure Prediction and Structual Genomics. *Science*, 294:93–96, 2001.

[5] P. Barahona and L. Krippahl. Constraint Programming in Structural Bioinformatics. *Constraints*, 13(1-2):3–20, 2008.

[6] Adam L. Beberg, Daniel L. Ensign, Guha Jayachandran, Siraj Khaliq, and Vijay S. Pande. Folding@Home: Lessons From Eight Years of Volunteer Distributed Computing. In *Parallel and Distributed Processing, IEEE International Symp.*, pages 1–8, 2009.

[7] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000.

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

[9] Christopher M. Bishop. *Neural networks for pattern recognition*. Clarendon Press, 1995.

[10] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., 2006.

[11] Luca Bortolussi, Agostino Dovier, and Federico Fogolari. Agent-based protein structure prediction. *Multiagent and Grid Systems*, 3(2):183–197, 2007.

[12] F. Campeotto, A. Dovier, and E. Pontelli. Protein Structure Prediction on GPU: a Declarative Approach in a Multi-agent Framework. In *Proc. of 2013 International Conference on Parallel Processing*, pages 474–479. IEEE Computer Society, 2013.

[13] Federico Campeotto, Alessandro Dal Palù, Agostino Dovier, Ferdinando Fioretto, and Enrico Pontelli. A constraint solver for flexible protein model. *J. Artif. Intell. Res. (JAIR)*, 48:953–1000, 2013.

[14] Federico Campeotto, Agostino Dovier, Ferdinando Fioretto, and Enrico Pontelli. A GPU implementation of large neighborhood search for solving constraint optimization problems. In *Proc. of ECAI 2014*, pages 189–194, 2014.

[15] Nicholas Carriero and David Gelernter. Linda in Context. *Commun. ACM*, 32(4):444–458, 1989.

[16] P. Clote and R. Backofen. *Computational Molecular Biology: An Introduction*. J. Wiley & Sons, 2001.

[17] Christian Cole, Jonathan D. Barber, and Geoffrey J. Barton. The jpred 3 secondary structure prediction server. *Nucleic Acids Research*, 36(Web-Server-Issue):197–201, 2008.

[18] Gabriela Czibula, Maria-Iuliana Bocicor, and Istvan-Gergely Czibula. Solving the Protein Folding Problem Using a Distributed Q-Learning Approach. *Journal of Computer Technology and Applications*, 2:404–413, 2011.

[19] A. Dal Palù, A. Dovier, F. Fogolari, and E. Pontelli. Protein Structure Analysis with Constraint Programming. In *Computational Approaches to Nuclear Receptors*, chapter 3, pages 40–59. The Royal Society of Chemistry, 2012.

[20] A. Dal Palù, A. Dovier, A. Formisano, and E. Pontelli. CUD@SAT: SAT Solving on GPUs. *Journal of Experimental & Theoretical Artificial Intelligence (JETAI)*, 2014 on-line.

[21] Alessandro Dal Palù, Agostino Dovier, and Federico Fogolari. Constraint Logic Programming Approach to Protein Structure Prediction. *BMC Bioinformatics*, 5(186), 2004.

[22] Alessandro Dal Palù, Agostino Dovier, Federico Fogolari, and Enrico Pontelli. CLP-based Protein Fragment Assembly. *Theory and Practice of Logic Programming*, 10(4-6):709–724, 2010.

[23] Alessandro Dal Palù, Agostino Dovier, Federico Fogolari, and Enrico Pontelli. Exploring Protein Fragment Assembly Using CLP. In Toby Walsh, editor, *Proc. of IJCAI*, pages 2590–2595. IJCAI/AAAI, 2011.

[24] Alessandro Dal Palù, Agostino Dovier, and Enrico Pontelli. Computing approximate solutions of the protein structure determination problem using global constraints on discrete crystal lattices. *IJDMB*, 4(1):1–20, 2010.

[25] Saravanan Dayalan, Nalaka Dilshan Gooneratne, Savitri Bevinakoppa, and Heiko Schroder. Dihedral angle and secondary structure database of short amino acid fragments. *Bioinformation*, 1:78–80, 2006.

[26] D. Fischer. 3D-SHOTGUN: a novel, cooperative, fold-recognition meta-predictor. *Proteins*, 51(3):434–441, 2003.

[27] Andras Fiser. Template-based protein structure modeling. In *Computational Biology*, pages 73–94. Springer, 2010.

[28] F. Fogolari, G. Esposito, P. Viglino, and S. Cattarinussi. Modeling of polypeptide chains as C-α chains, C-α chains with C-β, and C-α chains with ellipsoidal lateral chains. *Biophysical Journal*, 70:1183–1197, 1996.

[29] 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):1–17, 2007.

[30] Pedro Pablo González Pérez, Hiram I. Beltrán, Arturo Rojo-Domínguez, and Máximo Eduardo Sánchez Gutiérrez. Multi-Agent Systems Applied in the Modeling and Simulation of Biological Problems: A Case Study in Protein Folding. *World Academy of Science, Engineering and Technology*, 58:128, 2009.

[31] M.H. Hao and H.A. Scheraga. Designing potential energy functions for protein folding. *Curr. Opin. Struct. Biol.*, 9:184–188, 1999.

[32] R. Hussong, B. Gregorius, A. Tholey, and A. Hildebrandt. Highly accelerated feature detection in proteomics data sets using modern graphics processing units. *Bioinformatics*, 25(15):1937–1943, 2009.

[33] IBM Blue Gene Team. Blue Gene: A vision for protein science using a petaflop supercomputer. *IBM Systems Journal*, 40:310–327, 2001.

[34] Thé Van Luong, Nouredine Melab, and El-Ghazali Talbi. Large Neighborhood Local Search Optimization on Graphics Processing Units. In *Workshop on Large-Scale Parallel Processing (LSPP) in Conjunction with the International Parallel & Distributed Processing Symposium (IPDPS)*, Atlanta, États-Unis, 2010.

[35] A.V. Morozov and T. Kortemme. Potential functions for hydrogen bonds in protein structure prediction and design. *Advances in Protein Chemistry*, 4(72):1–38, 2005.

[36] Duc Thien Nguyen, William Yeoh, and Hoong Chuin Lau. Stochastic dominance in stochastic dcops for risk-sensitive applications. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 257–264, 2012.

[37] Luigi Palopoli and Giorgio Terracina. Coopps: a system for the cooperative prediction of protein structures. *J. Bioinformatics and Computational Biology*, 2(3):471–496, 2004.

[38] Levi C.T. Pierce, Romelia Salomon-Ferrer, Cesar Augusto F. de Oliveira, J. Andrew McCammon, and Ross C. Walker. Routine Access to Millisecond Time Scale Events with Accelerated Molecular Dynamics. *Journal of Chemical Theory and Computation*, 8:2997–3002, 2012.

[39] W. Pirovano and J. Heringa. Protein Secondary Structure Prediction. In *Data Mining Techniques for the Life Sciences*, Methods in Molecular Biology, pages 327–348. Springer, 2010.

[40] A Roy, A Kucukural, and Y Zhang. I-TASSER: a unified platform for automated protein structure and function prediction. *Nature Protocols*, 5:725–738, 2010.

[41] D. Rykunov and A. Fiser. New Statistical Potential for Quality Assessment of Protein Models and a Survey of Energy Functions. *BMC Bioinformatics*, 11:128, 2010.

[42] J. Sanders and E. Kandrot. *CUDA by Example. An Introduction to General-Purpose GPU Programming*. Addison Wesley, 2010.

[43] M.C. Schatz, C. Trapnell, A.L. Delcher, and A. Varshney. High-throughput sequence alignment using Graphics Processing Units. *BMC Bioinformatics*, 8:474, 2007.

[44] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming - CP98, 4th International Conference*, volume 1520 of *Lecture Notes in Computer Science*. Springer Verlag, 1998.

[45] M. Y. Shen and A. Sali. Statistical potentials for assessment and prediction of protein models and a survey of energy functions. *Protein Science*, 15:2507–2524, 2006.

[46] K.T. Simons, R. Bonneau, I. Ruczinski, and D. Baker. Ab initio protein structure prediction of CASP III targets using ROSETTA. *Proteins*, 3:171–176, 1999.

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

[48] B. Sukhwani and M.C. Herbordt. GPU acceleration of a production molecular docking code. In *GPGPU*, pages 19–27, 2009.

[49] M. Taiji, J. Makino, A. Shimizu, R. Takada, T. Ebisuzaki, and D. Sugimoto. MD-GRAPE: A Parallel Special-Purpose Computer System for Classical Molecular Dynamics Simulations. In *Proc. of the 6th Joint EPS-APS international conference on Physics Computing*, pages 200–203, 1994.

[50] El-Ghazali Talbi. *Metaheuristics - From Design to Implementation*. Wiley, 2009.

[51] P. Van Hentenryck and L. Michel. Constraint-Based Local Search. The MIT Press, Cambridge, Mass., 2005.

[52] M. Wooldridge. *An Introduction to Multi Agent Systems*. John Wiley and Sons, 2002.

[53] Sitao Wu, Jeffrey Skolnick, and Yang Zhang. Ab initio modeling of small proteins by iterative tasser simulations. *BMC Biology*, 5(1):17, 2007.

[54] Yang Zhang. I-TASSER server for protein 3D structure prediction. *BMC Bioinformatics*, 9(40), 2008.