Multi-Agent Simulation of Protein Folding

Luca Bortolussi[1], Agostino Dovier[1], and Federico Fogolari[2]

[1] Dep. of Maths and Computer Science, University of Udine, 33100 Udine, Italy.
bortolussi|dovier@dimi.uniud.it
[2] Dep. of Biomedical Science and Tech., University of Udine, 33100 Udine, Italy.
ffogolari@mail.dstb.uniud.it

Abstract. A protein is identified by a finite sequence of amino acids, each of them chosen from a set of 20 elements. The Protein Structure Prediction Problem, fundamental for biological and pharmaceutical research, is the problem of predicting the 3D native conformation of a protein, when its sequence of amino acids is known. All current mathematical models of the problem are affected by intrinsic computational limits, and by a disagreement on which is the most reliable energy function to be used.
In this paper we present an agent-based framework for ab-initio simulations, composed by different levels of agents. Each amino acid of an input protein is viewed as an independent agent that communicates with the others. These agents are coordinated by strategic and cooperative higher level agents. The framework allows a modular representation of the problem and it is easily extensible for further refinements and for different energy functions. Simulations at this level of abstraction allow fast calculation, distributed on each agent. We provide an implementation using the Linda package of SICStus Prolog, to show the feasibility and the power of the method.
Keywords: *Computational Biology, Agent-Based Technologies, Protein Folding Simulation.*

1 Introduction

The Protein Structure Prediction Problem (PSP) is the problem of predicting the 3D *native* conformation of a protein, when the sequence made of 20 kinds of amino acids (or *residues*) is known. The process for reaching this state is known as the protein *folding*. This problem is fundamental for biological and pharmaceutical research, and it has been tackled both computationally and experimentally. Currently, the native conformations of more than 29000 proteins are available in the Protein Data Bank (PDB) [3]. Unfortunately, identifying the structure of a protein in a laboratory can take more than one-man year. This shows the need of a computational tool to make reliable predictions.

The PSP problem can be modelled as an optimization problem involving energy functions to be minimized and constraints on the amino acids' positions. Even simple abstractions are NP-complete (see, e.g., [9]). Nevertheless, in the last thirty years, the global optimization for the PSP problem has been tackled with different classes of methods: simulated annealing, genetic algorithms, smoothing methods, branch and bound (cf. [12] for a review), and constraints [10]. These methods are all *ab-initio methods*, namely, approaches that are not based on similarities to already known proteins.

When some additional knowledge is available (e.g., a database of already annotated proteins), it is possible to employ a different class of methods (*homology modelling*): the protein is matched against very similar sequences and the conformation prediction exploits this valuable information.

In this work we concentrate on *ab-initio* modelling. These methods are based on the *Anfinsen thermodynamic hypothesis* [2]: the (*native*) conformation adopted by a protein is the one with minimum free energy, i.e. the most stable state. A fundamental role in the design of a predictive method is played by the spatial representation of the protein and the static energy function, which is to be at a minimum for native conformations. All-atom computer simulations are typically unpractical, because they are extremely expensive. In fact, each simulated nanosecond for a small protein requires many CPU hours on a PC. To overcome this limit, most of the approaches use *database fragment assembly* and/or *simplified models* to determine an approximate and faster solution. Simplified models of proteins are attractive in many respects, as they have smoother energy hyper-surfaces, and the dynamics is faster. Unfortunately, there is no general agreement on the potential that should be used with these models, and several different energy functions can be found in literature [26].

In this paper we present a new high-level framework for ab-initio simulation using Agent-based technologies, which extends the one presented in [5]. It is developed by following the architecture for agent-based optimization systems presented by Milano and Roli in [21]. This framework stratifies the agents in different levels, according to their knowledge and their power. Here we have three layers: one containing agents designed to explore the state space, one dealing with agents implementing global strategies and the last one containing cooperation agents.

Each amino acid in the protein is modelled as an independent agent, which has the task of exploring the configuration space. This is accomplished mainly by letting these agents interact and exchange information. These processes operate within a simulated annealing scheme, and their moves are guided by the knowledge of the position of surrounding objects. The communication network changes dynamically during the simulation, as agents interact more often with their spatial neighbours. The strategic agents govern the environmental properties and they also coordinate the basic agents activity in order to obtain a more effective exploration strategy of the state space. The cooperative agent, instead, exploits some external knowledge, related to local configurations attainable by a protein, to improve the folding process. The communication between agents is based on Linda tuple space (see, e.g., [8]), and the program is implemented in SICStus Prolog [13]. Moreover, we have also implemented an equivalent multithread version, which is much more faster (cf. Section 5).

The protein model adopted in this version is the one developed by Micheletti et al. in [11], which is still simple, although more accurate than the one used in [5]. Anyway, different potentials can be easily used, as the framework is modular and independent from the energy function. In addition, it is also intrinsically concurrent, and so it can be run in distributed systems.

The results are encouraging. The program produces quite stable outcomes, in the sense that the solutions found in different runs have similar energies. Unfortunately, the potential used is still too coarse to produce good predictions in terms of RMSD from

known native structures. Moreover, the introduction of the strategic level improves very much the quality of the solutions (in terms of energetic values), while the cooperative agent decreases the RMSD, but increases the energy. This shows the need of using a potential with an higher resolution.

The paper is organized as follows. In Section 2 we introduce the Protein Structure Prediction problem and discuss some related work. In Section 3 we present the Agent-based framework, while in Section 4 we briefly describe the energy model employed. In Section 5 we provide some details of the implementation and in Section 6 we show some results. Finally, in Section 7 we draw some conclusions.

2 Proteins and the PSP Problem

A protein is made by amino acids. In nature there are 20 types of amino acids, that can be identified by a letter in the set $\mathcal{A} = \{A, \ldots, Z\} \setminus \{B, J, O, U, X, Z\}$. The *primary structure* of a protein is a sequence $s_1 \cdots s_n$, where $s_i \in \mathcal{A}$. Each protein assumes a peculiar 3D conformation, called native conformation or *tertiary structure* The primary structure of a protein uniquely determines its tertiary structure. The *protein structure prediction (PSP) problem* is the problem of predicting the tertiary structure of a protein given its primary structure. The tertiary structure determines the function of a protein and it is the 3D conformation that minimizes the global energy of the protein [2]. Though, there is no common agreement on which energy function should model correctly the phenomenon.

The general structure of an amino acid is reported in Fig. 1. There is a part *common* to all amino acids, the N–C_α–C' backbone, and a *characteristic part* known as *side chain*, which consists of a number of atoms ranging from 1 to 18. Each amino acid is linked to the following with the incoming and outgoing edges represented by arrows. For proline, the side chain is bound to the nitrogen atom, replacing the hydrogen found in all other aminoacids. A well-defined energy function should consider all possible interactions between all atoms of every amino acid composing the protein. A review of the various forces and potentials at this abstraction level can be found in [23]. However, all-atoms molecular dynamics simulations (e.g. [6, 18]) are precluded by the intrinsic complexity of the needed operations.

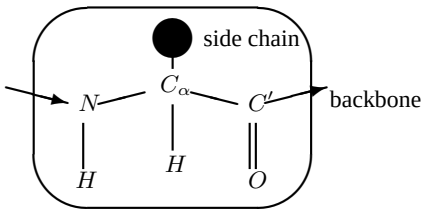

Fig. 1. Amino acids: Overall structure

A more abstract view of amino acids considers each of them as a single *sphere* centered in the C_α atom. It is reasonable to assume the distance between two consecutive C_α atoms to be 3.8 Å. Recent work has been done to model energy functions with this level of abstraction, and we refer to [26] for a detailed review. The most relevant features

of the folding process are local propensity to adopt well-defined secondary structures, as well as the polar and hydrophobic interactions. Local structures can be obtained either by rigid constraints (cf. [10]) or by statistical energy terms derived from databases. Interactions between aminoacids are modeled by considering chemical and physical properties or by statistical analysis. Some of the most used potentials are [27, 16, 4, 22]. Simplified *ab-initio* simulations actually are extremely difficult non-linear minimization problems, as the energy has an exponential number of local minima (cf. [23]), and they can try to be solved both on discrete lattices and off-lattice. In the former case, a constraint-based approach [10] was successful in solving proteins of length up to 50. There are also a lot of approaches for off-lattice minimization (cf. [23]), both sequential and parallel, but we are not aware of any agent-based one.

We adopt here an energy developed by Micheletti et al. [11], which will be explained in more detail in Section 4. The optimization scheme used is a parallel simulated annealing, and there are several versions of it (cf. [15] or [25]). However, our agent approach introduces new features, and allows a very clean separation between the different heuristics used.

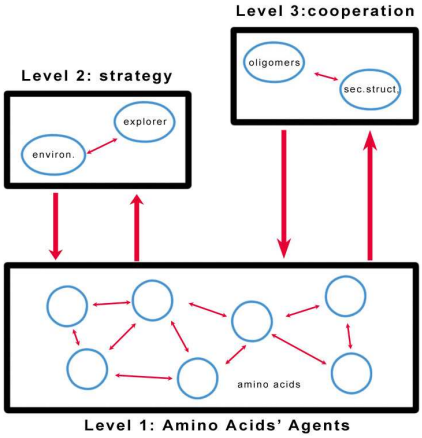

Fig. 2. Structure of the multi-agent simulation. Black boxes represent the levels, blue circles the agents and red arrows the communications.

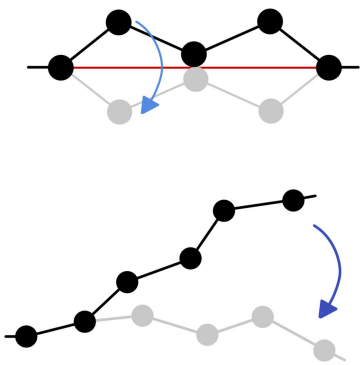

Fig. 3. Crankshaft move (above) and pivot move (below)

3 The Simulation Framework

In this section we describe the abstract framework of the simulation, which extends the one introduced in [5]. This scheme is independent both on the spatial model of the protein and on the energy model employed. Therefore, it can be instantiated using different representations. In fact, here we adopt a different and more effective energy model than in [5], though still coarse.

In the previous version, we associate to each aminoacid a single agent, which moves in the space and communicates with others in order to minimize the energy function.

Here we maintain this class of agents, but we also introduce other agents at different hierarchical levels, which have the objective of coordinating and improving the overall performance of the system.

Milano and alt. [21] have devised a general scheme to encode agent-based minimization, and our framework can be seen as an instantiation of that model. In particular, they identify four levels of agents, which interact in order to perform the optimization task. Level 0 deals with generation of an initial solution, level 1 is focused on the stochastic search in the state space, level 3 performs global strategic tasks and level 4 is concerned with cooperation strategies. According to this scheme, in [5], we implemented just level 1 agents, while here we have also level 2 and 3 processes (level 0 agents are trivial for this application). We deal separately with them in next subsections; see Figure 2 for a schematic representation.

We use Linda [8] as concurrent paradigm: all the communications are performed through writing and reading logical atoms in the Linda tuple space.

3.1 Level 1 — searching agents

We basically associate to each aminoacid an agent (from now on, we call them amino agents), which has the capability to communicate its current position to other processes, and to move in the search space, guided by its knowledge of the position of other agents. The general behaviour of these agents can be easily described declaratively by the predicate `amino(i,S)`, defined in Figure 4.

The first instruction is a blocking `read`, which tests if the term `authorized(i)` is present in the tuple space. This term is used to coordinate the activity with higher level agents: these processes can remove it and thus blocking the activity of the amino agents. Also the term `resting(i)` is used in this coordination task, and it allows an amino agent to tell other processes if he is moving or not. In fact, it is removed from the tuple space at the beginning of a computation, and it is added again at the end.

The second instruction is a blocking `in`, which removes the switch `trigger(i)` from the tuple space. This is a mechanism used to guarantee (a week form of) fairness to the system: each agent must wait for the movement of another process before performing its own move. In this way we avoid that a single agent takes the system resources all for itself. Clearly, at the beginning of the simulation the switches for all the amino processes are turned on, in order to let them move.

When the guards are satisfied, the process retrieves the most recent position of all other aminoacids (`get_pos`), which is stored in the tuple space in terms of the kind `pos(i,position)`. Successively, the current position of each agent is updated by `update_pos` through a mechanism described in section 3.1, and this new position is put in the tuple space by the following `out` instruction. Finally, the switches of all other processes are turned on by the $i-1$ instructions `trigger(j)`, with $j \neq i$, and then the process recursively calls itself. Actually, in the real implementation, triggers are added only if they are not already present.

The position of these agents is expressed in cartesian coordinates. This choice implies that the moves performed by these processes are local, i.e. they do not affect the position of other aminoacids. This is in syntony with the locality of the potential effects: the modification of the position of an aminoacid influence only the nearby ones.

The initial configuration of the chain can be chosen between four different possibilities: straight line, zig-zag, random, and, for known proteins, the deposited structure. The program terminates after a predefined number of executions of each agent, or when the temperature is low enough[3] and the sequences of computed positions are stored in an auxiliary file.

```
amino(i,S) :-
   read(authorized(i)),
   in(trigger(i)),
   in(resting(i)),
   get_pos([pos(1,Pos_1),...,
            pos(n,Pos_n)]),
   update_pos(i,S,
              [pos(1,Pos_1),...,
                  pos(n,Pos_n)],
                     Newpos),
   out(pos(i,Newpos)),
   out(trigger(1)),...,
   out(trigger(i-1)),
   out(trigger(i+1)),...,
   out(trigger(n)),
   out(resting(i)),!,
   amino(i,S).
```

Fig. 4. Basic Amino Agent

```
amino(i,S,Curr,Neigh_List) :-
    read(authorized(i)),
    in(trigger(i)),
    in(resting(i)),
    update_neigh(Curr,
          Neigh_List,New_NList),
    get_pos(New_NList,Pos_list),
    update_pos(i,S,New_NList,
            Pos_List,Newpos),
    out(pos(i,Newpos)),
    signal_move(New_NList),
    C1 is Curr + 1,
    out(resting(i)),!,
    amino(i,S,C,New_NList).
```

Fig. 5. Improved communication strategy

Moving Strategy The aminoacids move according to a simulated annealing scheme. This algorithm, which is inspired by analogy to the physical process of slowly cooling a melted metal to crystallize it (cf. [1]), uses a Monte Carlo-like criterion to explore the space. Each time the procedure `update_pos` is invoked, the amino acid a_i computes a new position p' in a suitable neighborhood (see below), and then compares its *current* potential P_c with the *new* potential P_n, corresponding to p'. If $P_n < P_c$, the amino acid updates its position to p', otherwise it accepts the move with probability $e^{-\frac{P_n - P_c}{\mathsf{Temp}}}$. This hill-climbing strategy is performed to escape from a local minimum. Temp is a parameter simulating the temperature effects. Technically, it controls the acceptance ratio of moves that increase the energy. In simulated annealing algorithms, it is initially high, and then it is slowly cooled to 0 (note that if $Temp$ is very low, the probability of accepting moves which increase the energy is practically 0). It can be shown that simulated annealing converges to the global optimum of the energy, if the temperature is lowered sufficiently slowly, i.e. logarithmically (cf. [1]). We discuss the cooling schedule in Section 3.2.

To guarantee ergodicity (cf. [1]), we let agents choose their next position in a very simple neighbourhood. In fact, it is a cube centered at the current position of the aminoacid, whose side has length set experimentally to 1 Å.

Communication Scheme The energy relative to a single aminoacid depends only from the adjacent aminoacids in the polymer chain, and from other aminoacids that are close enough to trigger the contact interactions. Far away agents won't bring any sensible contribution, at least as long as they remain distant.

This observation suggests a strategy to reduce considerably the communication overhead. Agent i first identifies its neighbours $\mathcal{N}$, i.e. all the aminoacids in the chain

[3] For simplicity, we omitted these details from the code presented.

that are at distance less than a certain threshold, fixed here to 14 Å. Then, for an user-defined number of moves M, it communicates just with the agents in $\mathcal{N}$, ignoring all the others. When the specified number M of interactions is reached, it will retrieve the position of all the aminoacids and then refresh its neighbour's list $\mathcal{N}$. In our LINDA framework, this means that the process i will turn on the switches `trigger(j)` and read the current position just for the agents $j \in \mathcal{N}$. The refresh frequency must not be too low, otherwise a far aminoacid could, in principle, come very close and even collide with the current one, without any awareness of what is happening. We experimentally found that $M = 100$ avoids these problems.

In Figure 5, we present the code for the modified agent behaviour.It is quite similar to the one presented before, with few obvious modifications in `get_pos` and `update_pos` clauses, and with the presence of a new function for updating the neighbour's list, i.e. `update_neigh`. This predicate performs the update only if $\mathrm{Curr} \equiv M \bmod 0$, otherwise copies `Neigh_List` into `New_NList`. A comparison of the performance of the two communication strategies can be found in Section 6.

3.2 Level 2 — strategy

The first layer of agents is designed to explore the state space, using a simulated annealing strategy. The way the solution space is searched influences very much the performance at finite of stochastic optimization algorithms. The amino agent activity results in an algorithm where subset of variables of the system are updated independently by different processes. However, the neighborhood of each agent is quite simple, in order to also avoid problems arising from delayed communication. In addition, most of its points have high distance penalties. This implies that the program is unwilling to produce good solutions in acceptable time periods. Hence we need a more coordinated and efficient search of the solution space, which can be achieved by a global coordination of the agents. This task can be accomplished by a higher level agent, which has a total knowledge of the current configuration, and it is able to control the activity of the single agents. Details are provided in next subsection.

At the same time, the simulated annealing scheme is based on the gradual lowering of the temperature, which is not a property of the amino agents, but it is rather a feature of the environment where they are acting. This means that the cooling strategy for temperature must be governed by an higher level agent, which is presented below.

Enhanced exploration of state space To improve the efficiency, we designed an higher level agent, called the "*orchestra director*", which essentially suspends the amino agents activity and moves them in the state space according to a different, global strategy.

It can move the chain using two different kind of moves: crankshaft and pivot (cf. Figure 3). Crankshaft moves essentially fix two points in the chain (usually at distance 3), and rotate the inner aminoacids along the axis identified by these two extremes by a randomly chosen angle. Pivot moves, instead, select a point in the chain (the pivot), and rotate a branch of the chain around this hub, again by a random angle. These global moves keep fixed the distance between two consecutive C_α carbon atoms, and are able to overcome the energy barriers introduced by the distance penalty term.

The agent is described by the code in Figure 6. In the first line, this agent removes from the tuple space the terms `authorized(i)`, thus suspending the execution of

all the amino agents. Actually, some of these agents may still be moving during the activation of the director, hence it waits for their suspension by a blocking read of the terms `resting(i)` (put in the tuple space by the amino agents while they are inactive). Once the director has got the position of the aminoacids (`get_position`), it performs some moves by calling the `move_chain` clause. This predicate calls itself recursively a predefined number `N` of times (in our experiments we set N to 100), and each time it selects what move to perform (i.e. crankshaft or pivot), the pivot points and an angle. Then it computes the energy associated to the old and the new configurations, and applies a Monte Carlo criterion to accept the move (cf. Section 3.1). Finally, it releases the amino agents by putting again in the tuple space the terms `authorized(i)`, and rests for a predefined amount of time (`sleep(WaitTime)`).

```
director(S,N,WaitTime) :-
    in(authorized(1)),...,
        in(authorized(n)),
    read(resting(1)),...,
         read(resting(n),
    get_position(Pos_List),
    move_chain(Pos_List,
         New_Pos_List,S,N),
    put_position(New_Pos_List),
    out(authorized(1)),...,
        out(authorized(n)),
    sleep(WaitTime),
    director(S,N,WaitTime).
```

Fig. 6. Strategic agent

```
cooperator(Sec_Str) :-
    get_position(Pos_List),
    identify_oligomers(Pos_list,
                       Oligo_List),
    communicate(Oligo_List),
    communicate_sec(Sec_Str),
    cooperator(Sec_Str).
```

Fig. 7. Cooperative agent

Environment The environmental variables of the simulation are managed by a dedicated agent. In this case, the environment simply controls the temperature of the simulated annealing algorithm. This parameter must not be conceived as a physical quantity, but rather as a control value which governs the acceptance ratio in the choice of moves increasing the energy.

From the theory of simulated annealing (cf. [1]), we know that the way the temperature is lowered is crucial for the performance of the algorithm. Convergence is guaranteed if the temperature is cooled logarithmically towards zero. Anyway, to reach good approximations of the global minimum, one has just to let the simulation perform a sufficient number of steps at each value of the control parameter, in order to stabilize the corresponding Markovian process.

We applied a simple and very used strategy: at each step the temperature is decreased according to $T_{k+1} = \alpha T_k$, where $\alpha = 0.98$. The starting temperature T_0 must allow an high acceptance ratio, usually around 60%, and we experimentally set T_0 as to attain this acceptance ratio at the beginning of the simulation. This value guarantees also that the system does not accept moves with too high energy penalties (cf. Section 6). Regarding the number of iterations at each value of T, we set it in such a way that the average number of moves per aminoacid is around 100, in order to change the value to all the variables a suitable number of times. The code for the environment agent is straightforward.

3.3 Level 3 — cooperation

In this section we present a dynamic cooperation strategy between agents, which is designed to improve the folding process and try to reach sooner the configuration of

minimum energy. The main idea behind is to combine concurrency and some external knowledge to force the agents to assume a particular configuration, which is supposed to be favorable. More specifically, this additional information can be extracted from a database, from statistical observations or from external tools, such as secondary structure predictors.

The cooperation is governed by an high level agent, which has access to the global spatial information and to some suitable external knowledge. What we have realized here is still a preliminary version, based on the identification of good local configurations. In particular, if aminoacids are, at some point of the simulation, close to a pattern "known" to be favorable, we force them to adopt it. These patterns are small local "building blocks" that compose the protein's tertiary structure, i.e. small oligomers (cf. [20] and next subsection). Being based on local information, this simple cooperation is not able to drive the folding process, but it can act as a stabilizing force. In addition, we exploit also some information about the secondary structure, favoring from the beginning the formation of local patterns supposed to appear in the protein (see below).

To coordinate the action of single agents and let a particular configuration emerge from their interaction, we adopt a strategy which is very similar in spirit to the "computational fields" technique introduced by Mamei et al. in [19]. The idea is to create a virtual force field that can drive the movement of the single agents towards the desired configuration. In our setting we deal with energy, not with forces, so we find more convenient to introduce a biasing term modifying the potential energy calculated by a single agent. In this way, we can impose a particular configuration by giving an energy penalty to distant ones (in terms of RMSD).

Cooperation via Oligomers Micheletti et al. ([20]) studied in detail the recurrent oligomeric structures of proteins. They looked for repeated patterns of 5 consecutive aminoacids in known tertiary structures stored in PDB, and they found that just 40 different protein pieces are needed to reproduce the whole ensemble of structures, with small errors. In fact, the tertiary structure of a protein can be reconstructed using this small set of oligomers with a precision of 1 Å or less.

Our idea is to use this knowledge for a local improving of the folding. If 5 consecutive aminoacids are in a configuration which is close to a particular block, we introduce a biasing potential term that forces them to adopt this structure and we see if this choice is effective or not. "Close" here means that the RMSD between the current aminoacids' configuration and the oligomer is below a certain user-defined threshold. However, the introduction of a local biasing can have some negative drawbacks. In particular, if this heuristic is activated too early, it may convey rigidity to the simulation, making inaccessible favorable areas of the state space. This means that the chain could be frozen in unnatural configuration, just because it was passing close to it by chance. To avoid this collateral effects, we activate the cooperative agent late in the simulation phase, when the temperature is low and the structure is already quite rigid. Then the modifications of the potential will result in blocking the chain in a configuration which is locally resembles a real protein. Moreover, if the energy function is not too approximate, and the optimization has brought the chain quite close to global minimum, the cooperation energy may let the chain jump quickly to this minimum, with a further stabilizing effect.

While trying to cover the chain with oligomers, we must ensure that overlapping blocks (for instance, the block from aminoacid 1 to 5, and block from 3 to 7) coincide in the overlapped region. Of course, this restricts the choice of consecutive pieces to a subset of the 40 oligomers. Moreover, we also assign to each block a number depending on its frequency of appearance relative to the type of the aminoacids it corresponds to. In this way, we can avoid to force the chain in non-physical configurations.

The global coordination agent identifies the best set of oligomers to cover the actual conformation of the chain, i.e. the set that minimizes the sum of the RMSD between the structure and each oligomer, multiplied by a factor depending on the frequency of appearance. To tackle combinatorial explosion (the number of possible coverings is exponential in the length of the chain), we use a build-up approach that identifies the best covering sets for subchains and then glues them together. Once the selection has been performed, the cooperator activates the biasing potential terms in each amino agent. This is achieved by putting in the tuple space the list of the selected oligomers. The energy term computed by each aminoacid is proportional to the RMSD between the actual configuration and the oligomers corresponding to pieces of the chain containing it. The declarative code of the cooperator agent is shown in Figure 7. Practically, its behavior is governed by the `identify_oligomers` clause, which executes the selection of the cover set, and by the `communicate` predicate, which posts the information to the aminoacids. The `communicate_sec` clause broadcasts knowledge about the secondary structure, as explained below.

Cooperation via Secondary Structure In the literature it is recognized that the formation of local patterns, like α-helices and β-sheets, is one of the most important aspects of the folding process (cf. [23]). Actually, there are a lot of programs capable of predicting with good accuracy the location of these local structures, e.g. [17]. We plan to use the information extracted from them to enhance the simulation. For the moment, however, we introduced a preliminary version of cooperation via secondary structure, which identifies the location of secondary structure directly from pdb files (we are not cheating too much, because secondary structure predictors are very effective, especially with short proteins).

Once the cooperation agent possess this knowledge, it activates another computational field that forces aminoacids to adopt the corresponding local structure, via the `communicate_sec` predicate. The mathematical form of this new potential is similar to the one used with oligomers, i.e. it penalizes all configurations having an high RMSD from a "typical" helix or sheet.[4] Of course, this energy regards only the aminoacids supposed to form a secondary structure, and it is activated from the beginning of the simulation, as it should be able to drive the folding process, at least locally.

4 Energy function

In this section we briefly describe the energy model we use in our simulation. Before entering into details, we stress once more that the choice of the energy function is orthogonal to the development of the concurrent framework, which can be easily adapted to encapsulate different models. However, the results in terms of structure prediction

[4] There are different kinds of α-helices and β-sheets, but we omit here further details.

are strictly dependent from the adopted model: the more detailed the energy function is, the better the outcome will be. Unfortunately, computational complexity grows together with accuracy, and often one has to choose a compromise between these two features.

In the literature there is a plenty of different potentials. Here we have adopted the energy developed by Micheletti et al. in [11], mainly due to its simplicity. Each aminoacid is represented by a single center of interaction, identified with the C_α carbon atom. The original energy function comprises three terms, devoted to interaction, cooperation and chirality. In addition, it contains some hard constraints that forbid non physical configurations. However, the asynchronous communications between agents make the usage of hard constraints a too strict policy for the movement. In fact some good configurations may become unreachable due to the outdated information used (cf Section 6 for further comments). A possible way out of this problem is to convert the hard constraints into energy barriers that penalize the non-physical configurations.

In the following we describe the energy terms involved in a concise way, omitting their mathematical expressions; further details can be found in [11]. If we indicate with $\mathbf{x}$ the spatial disposition of the aminoacid's chain and with $\mathbf{t}$ their type, the energy can be expressed as

$$\begin{aligned} E(\mathbf{x},\mathbf{t}) = E_{cooperative}(\mathbf{x},\mathbf{t}) + E_{pairwise}(\mathbf{x},\mathbf{t}) + E_{chiral}(\mathbf{x},\mathbf{t}) \\ + E_{steric}(\mathbf{x}) + E_{chircst}(\mathbf{x}) + E_{dist}(\mathbf{x}). \end{aligned} \tag{1}$$

The pairwise term captures the interactions that occur between two aminoacids that are close enough. The contact between two aminoacids is modeled with a sigmoid function, and is weighted using an empirical contact energy table (cf. Section 2). The cooperative term involves four different aminoacids, and it tries to improve the packing of secondary motifs. It advantages the situations where the aminoacid i is bonded to j, and one aminoacid close to i is bonded to one close to j. The chiral term, instead, is used to favor the formation of helices for some putative segments.

In addition to these three energy terms, we have several constraints implemented via energy barriers. E_{steric} imposes three steric constraints to the position of C_α and C_β atoms. The position of C_β is calculated from the chain of alpha carbon atoms using the Park and Levitt rule (cf. [24]). Two of such atoms must be at a distance r greater than 3. This is achieved by means of a potential barrier of the form $\left(\left(\frac{3}{r}\right)^6 - 1\right)$ if $r < 3$ and 0 otherwise. The $E_{chircst}$ is similar in spirit to the last term, a part from forbidding local configurations which have unnatural torsional angles. Finally, the E_{dist} term tries to keep fixed the distance between two consecutive C_α atoms, around the value of 3.8 Å. This is achieved through a parabolic potential of the form $(r - 3.8)^2$. These constraint terms are experimentally weighted in order to homogenize their values with the ones assumed by original terms of the function.

5 Implementation

In this section we describe some details of the implementation of the simulation technique in the language SICStus Prolog [13]. We have interfaced SICStus with C++ where energy functions are computed. In particular, the whole mechanism for updating the positions (i.e. the `update_pos` predicate — Sec. 3) is implemented in C++ and dynamically linked into Prolog code. This guarantees a more efficient handling of the considerable amount of operations needed to calculate the potentials. The Prolog code

is not very different from its abstract version presented in section 3.1, and its length is less than 150 lines. We have also written a C++ manager which launches SICStus Linda processes, visualizes the protein during the folding, and in general interacts with the Operative System.

Actually, LINDA communication is not very efficient: each communicative act takes about 100 milliseconds, thus leading to a very slow simulation. Therefore, we have also written a multithreading version in pure C++, which can run both under Windows and Linux. This version reproduces the communication mechanisms of LINDA using the shared memory, so it is equivalent to the program presented in the paper, though much more efficient. All the codes can be found in `http://www.dimi.uniud.it/dovier/PF`.

6 Experimental results

In this section we present the results of some tests of our program. We are mainly interested in two different aspects: seeing if and how the novel features introduced here (fast communication, strategy, cooperation) improve the simulation and checking how good are the predicted structures with respect to the resolution of the energy function used. Note, however, that this potential is structurally very simple, so we are not expecting outstanding results out of it. Actually, it was used in [11] to produce coarse structures that were then improved by all-atom molecular dynamics simulations.

We ran the simulation on different proteins of quite small size, taken from PDB [3], because longer chains are out of the resolution of the potential. We also had to tune a lot of parameters of the program, especially the cooling schedule for the temperature, the weights of the penalty terms and the scheduling of the strategic and the cooperative agents.

The quantities used to estimate the goodness of the results are the value of the energy and the root mean square deviation (RMSD) from the known native structure. Note that the relation between these two quantities is connected with the energy function used, not with the strategy for searching the space. All the tests were performed on a single processor machine (a notebook with a 2.66 GHz Pentium 4), and therefore the execution time is biased by the parallelism inherent in the simulation, which loads the single processor with a considerable computational overhead.

In Table 1, we show the best results obtained in terms of RMSD and energy without cooperation, while in Table 2 cooperation was active. The energy considered here is the one presented in Section 4; the contributions of cooperative computational fields are not considered. The numbers shown are an average of 5 runs; standard deviation is shown in brackets.

Protein	RMSD	energy
1LE0	4.21 (0.72)	-10.253 (1.19)
1KVG	3.89 (0.77)	-19.41 (0.56)
1PG1	7.25 (0.59)	-54.56 (2.75)
1VII	7.22 (0.88)	-51.06 (2.81)
1E0M	8.65 (0.91)	-76.47 (3.78)

Table 1. Results without cooperation

Protein	RMSD	energy
1LE0	2.99 (0.53)	-5.75 (1.44)
1KVG	1.29 (0.12)	-9.27 (1.68)
1PG1	2.73 (0.46)	-23.04 (1.86)
1VII	6.81 (0.38)	-32.13 (6.81)
1E0M	5.96 (0.38)	-22.18 (1.29)

Table 2. Results with cooperation

From Table 1, we can see that, without cooperation, the simulation is quite stable: most of the runs produce solutions with energy varying in a very small range of values. On the contrary, the RMSD is quite high. This depends mostly on the low resolution of the potential. In fact, this energy function has terms which compact the chain, but no term imposing a good local structure. Therefore, the simulation maximizes the number of contacts between aminoacids, giving rise to a heavily non-physical shape. The situation, however, changes for 1LE0, 1PG1, 1KVG and 1E0M with the introduction of the cooperative effects. In this case, the maximization of contacts goes together with potential terms imposing good local shapes (and some global contact, in case of β-sheets secondary structures), creating better structures but worsening the energy (less contacts are formed). In particular, we can see a remarkable improvement of RMSD for proteins 1KVG, 1E0M and 1PG1; a visual comparison of the outcome for this last protein is shown in Figure 8. However, the situation is different with 1VII. In this case, in fact, the introduction of the cooperative terms worsens the energy, but does not improve sensibly the RMSD. This can be explained observing the shape of 1VII and the behaviour of the potential for its native configuration. In particular, this protein has three small helices, and the interaction terms of the energy, evaluated for the native configuration, have a value of 20, while the values found with our simulation are around -20 with cooperation and -40 without it. This simply means that this energy function penalizes a configuration which has helices and is compact (i.e. the native structure of 1VII). Therefore, while imposing the formation of helices with cooperation, the simulation tries to minimize as much as possible the interaction terms, leaving the structure more opened, and not improving the RMSD (although it is locally more physical). With the other proteins, the situation is better, as they all have also β-sheets, which are a non-local secondary structure: they impose some contacts between distant pieces of the chain. This non-local information seems to interact better with the compressive action of the energy function. Therefore, these tests show that, in order to produce better predictions, we need a more accurate potential.

To test the different terms of the cooperative field, we compare separately runs with and without cooperation via oligomers, and via secondary structure. It comes out that the quality of the solutions in terms of energy are always worse with the cooperative agent active (cf. above). In addition, using just the oligomers' information, the RMSD is not improved considerably, but this depends on the low resolution of the energy function used. In fact, the structures identified by the energy minimization are coarse, and do not have good local features. Therefore, when trying to force the chain locally into a protein-like shape, we create a conflict with the potential. In addition, this cooperative term is not able to modify the global structure of the chain, and so it is not able to improve sensibly by itself the results in terms of RMSD from the native configuration. Anyway, when we introduce also the secondary structure information, the situation changes, and the RMSD is improved in most of the cases (cf. Table 1 and 2). This is quite remarkable, as the information relative to secondary structure is still local. We argue that a better form of cooperation, with some capability of driving globally the folding process, will enhance the results even more (cf. Section 7).

To evaluate the enhancements introduced by the strategic agent, we deactivate it and compared the results. As expected, the aminoacid agents alone are not able to explore

well the state space, and the minimum values found in this case are very poor (for the 1VII, much greater than zero). This depends essentially by the fact that most of their moves violate the distance constraint, especially at high temperatures. At low temperatures, instead, the system seems driven more by the task of minimizing this penalty term, than by the optimization of the "real" components of energy. Therefore, the simulation gets stuck very easily in bad local minima, and reaches good solutions just by chance. The results shown in Table 1, instead, are obtained activating the "orchestra director" agent just a couple of times for each value of the temperature. This suffices to obtain much better energy minima. Note that combining the aminoacid agents and the strategic one corresponds to having a mixed strategy for exploring the state space, where two different neighborhoods are used: the first, local and compact, is searched by the aminoacid agents, while the second one, which links configurations quite far away, is searched by the strategic agent.

Regarding the optimized communication, we saw that it reduces the number of communications quite a lot, and it fastens considerably the SICStus version of our simulator, which suffers from the low speed of access to the tuple space. The multithread version, instead, does not show a great reduction in time, but this depends form the fact that communications there are performed via shared memory (which is very fast).

Finally, it is well known that asynchronous parallel forms of simulated annealing can suffer from a deterioration of results with respect to sequential versions (cf. [15]), due to the use of outdated information in the calculation of the potential. So we also run some sequential simulations, using essentially the moves performed by the orchestra director. The quality of the results obtained (in terms of energy values) is of the same level than the multi-agent optimization. Moreover, the sequential simulation employed more or less the same time of the multi-agent one to find them. This is quite encouraging, because the speed of the latter one can be increased by running it on a parallel machine.

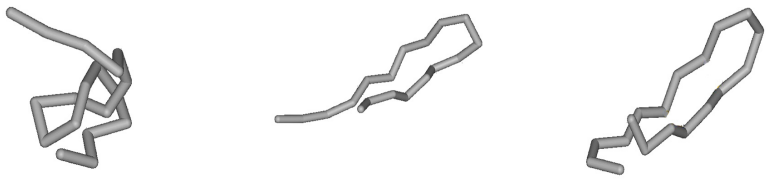

Fig. 8. Protein 1PG1. From left to right: a solution without cooperation, a solution with cooperation and the native state.

7 Future work and Conclusions

In this paper we present a multi-agent based framework to simulate a protein folding process, designed according to the MAGMA scheme [21]. This approach is independent from the energy model used, and can be easily adapted to more complex spatial representations and potential functions. We basically identify every biological entity (aminoacid) with a concurrent agent. We have designed also other agents, aimed at coordinating the activity of the basic processes and inducing some basic form of cooperation.

Though our goal is to provide a powerful tool for folding proteins, this is still a preliminary version, devoted to analyze the improvements that can arise from the introduction of a multi-layer architecture. In fact, the energy function used here is too coarse to provide good biological models, as confirmed also from out tests. In the future, we plan to use more reliable energies, maybe encapsulated in an iterative process using more and more detailed —and computationally expensive— potentials and refining the previous solutions. Moreover, we want to run the system in a parallel machine, to take full advantage from the intrinsic concurrency of its design.

Regarding the cooperation level, the current version shows interesting potentials. In fact, the introduction of a local cooperation is able to improve sensibly the resolution of the energy function (in terms of predicted structure). However, its local action is still too weak to drive globally the folding process, hence we want to extend it using more external knowledge. In particular, we want to introduce information about the real physical dynamic of the folding. For example, we could add a computational field that mimics the hydrophobic force, therefore compactifying the structure at the beginning of the simulation.

Or results showed that one of the main problems the system is facing is the introduction of the penalty terms in the potential. In particular, the term that should keep fixed the distance between two consecutive aminoacids is troublesome. So many points in an aminoacid's neighbour violate it that the simulation, especially at high temperatures, seems driven more by the tentative to bound this effect than by the minimization task itself. To circumvent this problem, we plan to change the coordinates expressing the configuration of the system. In particular, we can shift from the cartesian coordinates to angular ones. This has the advantage of reducing the degrees of freedom of the system, but at the price of both making non-local the moves of a single aminoacid and rising the computational cost of function evaluations. Therefore, much more attention must be put in the design of the communication and coordination mechanisms between agents.

We want also to improve the strategic level, perfecting the strategic coordination between agents for the exploration of the state space. In addition, we want to introduce in the program some concepts borrowed from the evolutionary algorithms [7], thus letting different populations of agents coexist, search the space and interact. Furthermore, to tackle the slowness of the simulated annealing scheme, we wish to improve it in the direction of the parallel tempering [14].

References

1. E. Aarts and J. Korst. *Simulated Annealing and Boltzmann machines*. John Wiley and sons, 1989.
2. C.B. Anfinsen. Principles that govern the folding of protein chain. *Science*, 181:223–230, 1973.
3. H. M. Berman et al. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000. `http://www.rcsb.org/pdb/`.
4. M. Berrera, H. Molinari, and F. Fogolari. Amino acid empirical contact energy definitions for fold recognition. *BMC Bioinformatics*, 4(8), 2003.
5. L. Bortolussi, A. Dal Palù, A. Dovier, and F. Fogolari. Protein folding simulation in CCP. In *Proceedings of BioConcur2004*, 2004.
6. B. R. Brooks et al. Charmm: A program for macromolecular energy minimization and dynamics. *J. Comput. Chem.*, 4:187–217, 1983.

7. P. Calegari, G. Corai, A. Hertz, D. Kobler, and P. Kuonen. A taxonomy of evolutionary algorithms in combinatorial optimization. *Journal of Heuristics*, 5:145–158, 1999.
8. N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, 1989.
9. P. Crescenzi, D. Goldman, C. Papadimitrou, A. Piccolboni, and M. Yannakakis. On the complexity of protein folding. In *Proc. of STOC*, pages 597–603, 1998.
10. A. Dal Palù, A. Dovier, and F. Fogolari. Constraint logic programming approach to protein structure prediction. *BMC Bioinformatics*, 5(186), 2004.
11. G. M. S. De Mori, C. Micheletti, and G. Colombo. All-atom folding simulations of the villin headpiece from stochastically selected coarse-grained structures. *Journal Of Physical Chemistry B*, 108(33):12267–12270, 2004.
12. C. Floudas, J. L. Klepeis, and P. M. Pardalos. Global optimization approaches in protein folding and peptide docking. In F. Roberts, editor, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. 1999.
13. Swedish Institute for Computer Science. Sicstus prolog home page. `http://www.sics.se/sicstus/`.
14. C. J. Geyer. Markov chain monte carlo maximum likelihood. *Computing Science and Statistics*, pages 156–163, 1991.
15. D. R. Greening. Parallel simulated annealing techniques. *Physica D*, 42:293–306, 1990.
16. A. Liwo, J. Lee, D.R. Ripoll, J. Pillardy, and H. A. Scheraga. Protein structure prediction by global optimization of a potential energy function. *Proceedings of the National Academy of Science (USA)*, 96:5482–5485, 1999.
17. Pôle Biolnformatique Lyonnais. Gor iv secondary structure prediction method. `http://npsa-pbil.ibcp.fr`.
18. A. D. Jr. MacKerell et al. All-atom empirical potential for molecular modeling and dynamics studies of proteins. *J. Phys. Chem. B*, 102:3586–3616, 1998.
19. M. Mamei, F. Zambonelli, and L. Leonardi. A physically grounded approach to coordinate movements in a team. In *Proceedings of* ICDCS, 2002.
20. C. Micheletti, F. Seno, and A. Maritan. Recurrent oligomers in proteins - an optimal scheme reconciling accurate and concise backbone representations in automated folding and design studies. *Proteins: Structure, Function and*, 40:662–674, 2000.
21. M. Milano and A. Roli. Magma: A multiagent architecture for metaheuristics. *IEEE Trans. on Systems, Man and Cybernetics - Part B*, 34(2), 2004.
22. S. Miyazawa and R. L. Jernigan. Residue-residue potentials with a favorable contact pair term and an unfavorable high packing density term, for simulation and threading. *Journal of Molecular Biology*, 256(3):623–644, 1996.
23. A. Neumaier. Molecular modeling of proteins and mathematical prediction of protein structure. *SIAM Review*, 39:407–460, 1997.
24. B. Park and M. Levitt. Energy functions that discriminate x-ray and near-native folds from well-constructed decoy. *Proteins: Structure Function and Genetics*, 258:367–392, 1996.
25. M. Resende, P. Pardalos, and S. Duni Ekşioğlu. Parallel metaheuristics for combinatorial optimization. In R. Correa et al., editors, *Models for Parallel and Distributed Computation - Theory, Algorithmic Techniques and Applications*, pages 179–206. Kluwer Academic, 2002.
26. J. Skolnick and A. Kolinski. Reduced models of proteins and their applications. *Polymer*, 45:511–524, 2004.
27. T. Veitshans, D. Klimov, and D. Thirumalai. Protein folding kinetics: timescales, pathways and energy landscapes in terms of sequence-dependent properties. *Folding & Design*, 2:1–22, 1996.