Protein Folding Simulation in CCP

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

[1] Dip. di Matematica e Informatica, Univ. di Udine
Via delle Scienze 206, 33100 Udine (Italy).
(bortolus|dalpalu|dovier)@dimi.uniud.it
[2] Dip. Scientifico-Tecnologico, Univ. di Verona
Strada Le Grazie 15, 37134 Verona (Italy). fogolari@sci.univr.it

Abstract A protein is identified by a finite sequence of aminoacids, each of them chosen from a set of 20 elements. The Protein Structure Prediction Problem is the problem of predicting the 3D native conformation of a protein, when its sequence of aminoacids is known. This problem is fundamental for biological and pharmaceutical research. Due to intrinsic computational limits, no general solution is available. In particular, simulation-based techniques that handle every chemical interaction between all atoms in the aminoacids (and the solvent) involve a huge number of computations. As a consequence, simulation programs of this family run extremely slow. Moreover, these programs are typically written in imperative languages and hard to be parallelized. In this paper we present a novel high-level simulation approach to the problem using concurrent constraint programming. Each aminoacid in the input sequence is viewed as a process that communicates with the others. The framework allows a modular representation of the problem and it is easily extensible for further refinements. The implementation in Mozart is rather natural; the code is intrinsically concurrent and thus easy to be parallelized.

Keywords: Computational Biology, Concurrent Constraint Programming.

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 aminoacids (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. Currently, the native conformations of more than 24000 proteins are available in the Protein Data Bank (PDB) [3], which typical length is less than 500 aminoacids.

Unfortunately, the decision versions of the mathematical abstractions of this problem are shown to be NP-complete (see, e.g., [8]). Nevertheless, in the last thirty years the PSP problem has been tackled with different classes of methods: *homology modelling*, when structures of very similar sequences are compared; *fold recognition*, when residues or sequence properties are compared; and *ab-initio* prediction, when no similarity is found with available structures. In the last case, an all-atom computer simulation is typically unpractical and most of the approaches use database fragment assembly and/or simplified models.

Ab-initio methods are based on the *Anfinsen thermodynamic hypothesis* [1]: the (*native*) conformation adopted by a protein is the most stable one, i.e. the one with minimum free energy. A fundamental role in the design of a predictive method is played by the representation of the protein and the energy function, which is to be at a minimum for native conformations. Simplified models of proteins are attractive as they allow clear derivation of kinetic and thermodynamic properties, they lead to much faster computations and they generate also smoother energy hyper-surfaces which implies faster dynamics.

In this paper we present a new approach for a high-level simulation method in Concurrent Constraint Programming [18]. Each aminoacid in the protein is modelled as an independent and parallel process. A process reacts to modifications of spatial positions of other aminoacids; each process evolves in a Montecarlo simulation framework, exploiting the most recent information available about the surrounding objects. Every time a process updates its position, it also tells the changes to the others with a communication based on the instantiation of logical variables.

We have implemented the above idea in Mozart [21], adopting a simple model, where each aminoacid is represented by an off-lattice, single center of interaction. The energy function is composed by the empirical contact term developed in [4]. Moreover, to model properly an off-lattice energy field, we also include a bond length term, a bend angle term and a torsion angle term, according to [22]. To tune properly the various parameters, we define an automatic procedure that tests the energy model on a representative selection of proteins from the PDB, by means of a simulated annealing process. We run a poly-alanine test sequence, which has an high tendency to fold into an α-helix. Even with a so coarse model (in terms of aminoacid representation and energy function), we obtain a proper helical structure.

2 Related Work

We refer to [20] for a detailed review. We wish to focus here on the *ab-initio* approaches, i.e. not based on similarities to already known proteins. All-atoms ab-initio simulations by means of molecular dynamics (e.g. [17, 5, 14]), are precluded by the intrinsic complexity of the needed operations. More efficient methods are offered by simplified models. It is accepted that important features of protein sequences are the local propensity to adopt well-defined secondary structures and the polar and hydrophobic interactions. The secondary structure propensity may be included in simulation through either rigid constraints (as done in [10, 9]) or energy terms which depend on the type of aminoacids involved. Interactions between aminoacids may be treated either considering their chemical and physical properties or using a statistical approach. As far as empirical contact energies are concerned, in [4] it is compiled a table which has been proven to be rather accurate, when tested on several decoys' sets. Similar tables have been provided based on different criteria by other authors [15]. Thirumalai et al. [22] have designed a forcefield suited for representing a protein through its C_α-chain,

which includes bonds, bend and torsion angle energy terms. Scheraga et al. [13] have used a similar model including side-chain centroids.

The problem can also be formulated as a non-linear minimization problem, where the spatial domain for the aminoacids is a discrete lattice. A constraint-based approach to this problem on the so-called *Face Centered Cubic lattice*, with a further abstraction on aminoacids (they are split into two families H and P), is successfully solved in [2] for proteins of length up to 160. A constraint-based solution to the general problem (with the 20 aminoacids) is proposed instead in [10, 9], where proteins of length up to 50 are solved.

We are not aware of any other approach modelling aminoacids as concurrent processes.

3 Proteins and the PSP Problem

A protein is a sequence of 20 kinds of linked units, called aminoacids. This sequence is called the *primary structure* of a protein.

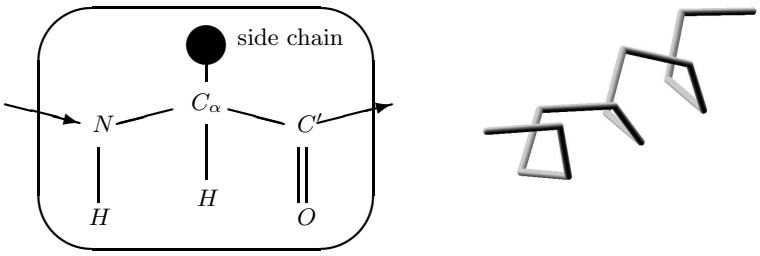

Fig. 1. Aminoacids' structure and the simulated sequence of alanines

Each protein always reaches a peculiar 3D conformation, called native conformation or *tertiary structure*, which determines the function of the protein. The *protein structure prediction problem* is the problem of predicting the tertiary structure of a protein given its primary structure. It is accepted that the primary structure uniquely determines the tertiary structure. Due to entropic considerations [1], it is also accepted that the tertiary structure is the one that minimizes the global energy of the protein. Though, the energy function involved in this phenomenon is not yet uniquely accepted.

Each aminoacid is made by several atoms (cf. Fig. 1); there is a part *common* to all aminoacids, 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 aminoacid is linked to the following with the incoming and outgoing edges represented by arrows in Fig. 1. A well-defined energy function should consider all possible interactions between all atoms of every aminoacid composing the protein. A

review of the various forces and potentials at this abstraction level can be found in [16].

A more abstract view of aminoacids considers each of them as a single *sphere* centered in the C_α atom. The distance between two consecutive C_α atoms is assumed to be 3.8 Å, measure chosen as unitary. Recent work has been done to model energy functions with this level of abstraction. A pair of non consecutive aminoacids contributes to the energy when the two aminoacids are in *contact*, namely under a given distance that can be approximated by 2 units. A table that points out the energy associated to pairs of aminoacids in contact has been developed [15, 4]. Let us denote by $\mathsf{Pot}(x, y)$ the energy value associated to a contact between aminoacids x and y; this value can either be positive or negative, according to the pair x, y.

4 Energy function for folding simulation

In this section we describe four energy terms that we use to simulate the dynamics of the folding process. The four energy contributions are related to *bond distance* (E_b), *bend angle* (E_a), *torsion angle* (E_t), and *contact interaction* (E_c). For the sake of simplicity, assume that $\boldsymbol{s} = s_1, s_2, \ldots, s_n$ contain the names of the n aminoacids as well as their positions. The total energy E depends on $\boldsymbol{s}$ as follows:

$$E(\boldsymbol{s}) = \eta_b\, E_b(\boldsymbol{s}) + \eta_a\, E_a(\boldsymbol{s}) + \eta_t\, E_t(\boldsymbol{s}) + \eta_c\, E_c(\boldsymbol{s}) \tag{1}$$

where $\eta_b, \eta_a, \eta_t, \eta_c$ are set for blending the energy contributions.

For each pair of *consecutive* aminoacids s_i, s_{i+1}, we have a quadratic term

$$E_b(\boldsymbol{s}) = \sum_{1 \le i \le n-1} \left(r(s_i, s_{i+1}) - r_0\right)^2 \tag{2}$$

where $r(s_i, s_{i+1})$ is the distance between the C_α of the two aminoacids and r_0 is the typical aminoacid distance of 3.8 Å.

The bend energy is associated to the bend angle of a triplet of consecutive C_αs, which is simply the angle formed by the two bonds linking the three carbons. The distribution is quite constant for every protein in the PDB and independent from the types of aminoacids involved. The profile (see [11]) can be approximated by a combination of two Gaussian distributions , one around 120 degrees and the other, sharper, around 90 degrees. The energy is obtained applying the opposite logarithm to the distribution function:

$$E_a(\boldsymbol{s}) = \sum_{i=1}^{n-2} -\log\left(a_1\, e^{-\left(\frac{\beta_i - \beta_1}{\sigma_1}\right)^2} + a_2\, e^{-\left(\frac{\beta_i - \beta_2}{\sigma_2}\right)^2}\right). \tag{3}$$

The torsion angle energy function is modelled using statistical information of torsional behavior extracted from the PDB. In detail, four C_α atoms ($C_{\alpha i}$, $C_{\alpha i+1}$, $C_{\alpha i+2}$ and $C_{\alpha i+3}$) form a specific torsion angle: it is the angle between

the normal to the plane spanned by $C_{\alpha i}, C_{\alpha i+1}$ and $C_{\alpha i+2}$ and the normal to the plane spanned by $C_{\alpha i+1}, C_{\alpha i+2}$ and $C_{\alpha i+3}$. The angle is positive when, looking along $\boldsymbol{r}_{23} = C_{\alpha i+2} - C_{\alpha i+1}$, the atom $C_{\alpha i+3}$ rotates clockwise. This angle is influenced both by the type of the aminoacids involved and by their position in the protein. The information available in the PDB does not allow one to reconstruct a sharp distribution profile of each combination of aminoacids (there are only 2.000 proteins with less than 25% homology, i.e. that carry non redundant information). Consequently, we first identify four classes of aminoacids which share the same torsional behavior and we calculate the distribution profile for every sequence of 4 consecutive classes. This profile is approximated by the sum of two Gaussians. The function has the form:

$$E_t(\boldsymbol{s}) = \sum_{i=1}^{n-3} -\log\left(a_1\, e^{\frac{(\Phi_i-\phi_1)^2}{(\sigma_1+\sigma_0)^2}} + a_2\, e^{\frac{(\Phi_i-\phi_2)^2}{(\sigma_2+\sigma_0)^2}}\right) \tag{4}$$

where the parameters a_1, a_2, σ_1, σ_2, ϕ_1, ϕ_2 depend on the classes of the four residues, while σ_0 is used to adapt the distribution variance to an effective energy function. For each pair of aminoacids s_i and s_j, such that $|i-j| \geq 3$ we consider a term of the form

$$E_c(\boldsymbol{s}) = \sum_{i=1}^{n-3}\sum_{j=i+3}^{n}\left[|\mathsf{Pot}(s_i,s_j)|\left(\frac{r_0(s_i,s_j)}{r(s_i,s_j)}\right)^{12} + \mathsf{Pot}(s_i,s_j)\left(\frac{r_0(s_i,s_j)}{r(s_i,s_j)}\right)^{6}\right] \tag{5}$$

where $r(s_i, s_j)$ is the distance between the C_α of s_i and s_j and $r_0(s_i, s_j)$ is a parameter describing the steric hindrance between a pair of non consecutive aminoacids s_i and s_j. In our model, $r_0(s_i, s_j)$ is the sum of the radii of the two spheres that represent the two specific aminoacids. An approximation of them is derived in [11].

A crucial problem while dealing with energy functions is to set correctly the parameters involved, as their value changes dramatically the energy landscape and influences the behavior of the simulation. We performed an optimization using a sampling of 700 proteins from PDB database with less than 25% homology. We calculated the sum of the squares of the difference between the energy computed for their native conformation and the folding energy (approximated as proportional to the number of aminoacids). Then we minimized this function using a simulated annealing method, identifying an optimal value for the scaling parameters.

5 The CCP Simulation program

In this section we describe the novel abstract framework of simulation. Note that the approach is independent on the energy model employed. In our preliminary version w.l.o.g. we used the functions described in the previous section. Using a concurrent constraint logic programming language we can encode the problem associating an independent process to each aminoacid involved. Processes react to changes of position of other processes.

The main program. Let us assume that the input is the list $S = [s_1, \ldots, s_n]$ of aminoacids. We report below the abstract CCP program.

```
1 simulation(S):-
2    Init=[[I1|_],
           [I2|_],
           ...,
           [In|_]],
3    run(1,S,Init) ||
4    run(2,S,Init) ||
5      ... ||
6    run(n,S,Init).
```

```
7 run(ID, S, [P1, P2, ..., Pn]):-
8   getTails([P1, ..., Pn],[T1, ..., Tn]),
9   ask(T1=[_|_]) -> skip +
10  ask(T2=[_|_]) -> skip +
11  ... +
12  ask(Tn=[_|_]) -> skip,
13  getLast([P1, ..., Pn],[L1, ..., Ln]),
14  updatePosition(ID,S,[L1,..,Ln],NP),
15  tell(TID=[NP|_]),
16  run(ID,S,[P1, ..., Pn]).
```

The main procedure/clause is described by lines 1–6. The variable `Init` contains n lists, each of them associated to the respective aminoacid. `I1 ... In` are the initial *positions* of the aminoacids $s_1, \ldots, s_n$, respectively. Each list will contain the history of moves done by the corresponding aminoacid. This procedure concurrently calls `n` executions of the procedure `run` each of them instantiated by the index `i` in the input sequence passed as parameter.

The process related to the aminoacid `ID` is activated when a new position is computed by some other aminoacid (lines 9–12). This control is made by checking that unbounded variables become a list.

New positions are appended to the tail, ensuring that the last element is always a new unbounded variable. For example, if the element `P4` has to be added to the list [`P1`, `P2`, `P3`|`Tail`], the information is updated posting the constraint `Tail` = [`P4`|`NewVar`], where `Tail` and `NewVar` are variables (line 15), obtaining the new list [`P1`, `P2`, `P3`, `P4`|`NewVar`].

Note that many concurrent updates can occur while a process is monitoring the changes. When at least one modification happened, the predicate `getLast` retrieves the latest information available and `updatePosition` calculates the new position as explained in Sect. 5. When the new position is available, it is added by means of `tell` to the list and the process is loop once more.

Simulating moves. The procedure `run` (lines 7–16) computes a new position for the corresponding aminoacid implementing a Montecarlo-like simulation (cf., e.g., [7, page 44]). Each aminoacid can move in the space guided by its evaluation of the energy function. The knowledge of other's positions and aminoacids' type is sufficient to completely evaluate the function. Each time the procedure `updatePosition` is invoked, the aminoacid a_i estimates its *current* potential P_c, according to its position p. Moreover, a new position p' is computed.

The position p' is close to p and reflects the capability of a_i to rotate around its neighbors while bonded in an almost rigid way. Technically p' is randomly selected in the space using the following probability distribution. Consider the set of points $\mathbb{S}$ which contains the points in which a_i can place, while maintaining the same distances from the adjacent neighbors (i.e. $\mathbb{S}$ describes a circle). According to a Gaussian distribution modelled by the distance of points in $\mathbb{S}$ and p, a point in $\mathbb{S}$ is randomly selected. We also implement the possibility to move out of the

theoretical allowed positions, and thus the point can be randomly shifted in the neighborhood according to a Gaussian distribution.

The aminoacid evaluates then the *new* potential P_n associated to p' (P_c and P_n are computed using the formula (1)). If $P_n < P_c$, the aminoacid updates its position to p'. If $P_n \geq P_c$, although the position p' seems not suitable to improve the local potential, it is possible that moving the aminoacid allows to exit from a local minimum. In this case the Montecarlo technique accepts the position p' with probability $e^{-\frac{P_n - P_c}{\mathsf{Temp}}}$. Temp is a parameter simulating the temperature effects, which remains constant in Montecarlo algorithms; the value of Temp controls the acceptance ratio of moves against the potential field and depends on the the values of the coefficients in the energy function. In Montecarlo simulations, it can be proven that allowing sufficient time the native conformation depending on the energy functions is reached.

6 Mozart implementation

In this section we describe some details of the implementation of the simulation technique in the language Mozart [21]. In Mozart it is natural to model concurrent programs in a simple notation mixing classes, constraints, and logic variables. Code's length is less than 1000 lines; the code can be found in `http://www.dimi.uniud.it/dovier/PF`.

The core scheme described in Sect. 5 is expanded with the help of two classes: `Protein` and `Amino`. The first class implements the `simulation` predicate and basically coordinates each process that is associated to an aminoacid, i.e. aminoacid creation, launch, statistics and termination. The `Amino` class describes each single aminoacid. `Protein` contains and runs a number of `Amino` classes equal to the number of aminoacids in the simulated protein.

As said before, lists are used to store positions of the various aminoacids during computation. The data structure is described by the class `PosList` that allows us to access the tail and last elements of the list in constant time, making use of Mozart class attributes (non logical variables).

The method `add` behaves as the `tell` operator in CCP. Each call `add(El _)` expands the list by one element and leaves the internal variable `tail` not instantiated, as described in Sect. 5.

The `Amino` class provides some methods for the process maintenance (e.g. `live`, `act`, `die`, `debug`) and others for the loop `act`, described in Sect. 5. The concurrent `ask`s (lines 9–12) check if at least one tail gets instantiated (i.e. a new position is communicated by another aminoacid). This test is accomplished by the method `Wait4News` (reported below). To implement this check it is necessary to generate n threads, one for each tail to be checked. Our solution is to nest then n threads that launch a `cond` blocking test for each `ask` (cf. TCCP translation of `ask` in [12]). In fact, since n is unbounded, it is not possible to statically write an explicit piece of code to handle the test. Each `cond` branch is associated to a distinct process that terminates either when its guard associated is realized or whenever another process terminates, by means of the additional shared variable

`Done`. On line 21, the notation `_|_` denotes the structure of a list, with general unassigned variables (i.e. `_`). When a tail is instantiated, this variable is unified with a term of the kind `El | _` and thus the `cond` test is verified.

```
17 meth Wait4News(N Done)
18         if (N\=@ID) then
19          thread
20             cond
21                {{Get @LocalAAList N} getTail($)} = _|_
22             then Done=unit
23             [] Done=unit then skip
24             end
25          end
26         end
27         if (N>1) then {self Wait4News(N-1 Done)} end
28 end
```

We give some details about the implementation of `updatePosition` and the subsequent `tell` (lines 14–15). The `updatePosition` method in Mozart reproduces what we described in Sect. 5. The computation of the energy is accomplished calling the method `energy(X Y Z E)`, where `X`,`Y` and `Z` are the coordinates of the aminoacid and `E` is the energy associated as described in Sect. 4. Note that we invoke twice the function: the first time with the actual coordinates and the second with the new position generated by a random shift. Once the two energies are compared, the next position is set and communicated via `add([X Y Z] _ )` to the appropriate `PosList`.

7 Experimental results

We tested our system either on known proteins or on artificial sequences of aminoacids. We picked some proteins with known native conformation and run a simulation for some time. We noticed that the native state of these proteins tends to move into another conformation, which has better energy according to our model. The main problem here is that our energy function is not able to properly capture the complex interaction which govern the global folding of the protein, being mostly calibrated on local interactions. We have still to work on energies and parameters. However, the code works properly on sequences of Alanines, that are known to have a high tendency to form a single helix. As initial state of the protein we set each aminoacid along a line with a step of the bond distance (3.8 Å). We run the simulation for 60 seconds on a PC, 1GHz, 256MB. In Figure 1 we show the resulting proper helix for a list of 14 Alanines.

We would like to comment this result obtained and compare it to a sequential simulation in **C** based on the same energy functions, where concurrency is simulated by a simple sequential loop on the aminoacids. We noticed that the behavior of the folding pathway is different. In Mozart, some aminoacids compute the energy with a partial updated information about the 3D conformation, due to concurrent communication and synchronism. In practice the aminoacids consider the latest information available about the others and this fact influence

the shape of the energy function. This shows that the concurrent approach is feasible to be employed in a folding process and it offers a new direction to be investigated, namely the parallel interaction between objects during the fold.

8 Future work and Conclusions

In this paper we present a novel concurrent constraint programming framework to simulate a protein folding process. The objective is to obtain the native conformation of a given protein; moreover, as a side effect, we compute all the history of the folding process. In our model, we identify each basic entity with an aminoacid and we define the relative energy function. This scheme is general and it allows fast prototyping. We implemented a preliminary version that is able to fold properly a helix, using a simplified energy field. In the future we plan to improve the level of approximation, i.e. to describe each residue by means of two elements: the C_α atom and the side-chain (cf. Fig. 1). This is a step towards all-atoms concurrent simulations. In particular, we want to model the side-chains as ellipsoids and we plan to use the UNRES force field [13].

It seems promising to design an optimized communication framework which adapts dynamically according to the 3D folding. For example, it could be possible to reduce the communications between non-influent pairs of entities (e.g. two distant aminoacids provide a poor energy contribution, thus a lazy position update is feasible) and conversely to concentrate the communication between interacting pairs. Moreover, we want to formalize a novel concept of cooperative approach, in which a cooperation strategy between processes is induced dynamically by the current configuration. We want to investigate the possibility offered by our concurrent framework to represent the dynamical evolution of the system and to manage the propensity to form local regular sub-conformations, to achieve a speed up in the folding pathway.

Our concurrent constraints simulation could be combined with other approaches. Since the computational costs also depend on the complexity of the energy function, it seems reasonable to proceed by levels: a first phase (e.g. the one described in [9]) could be used to quickly generate a preliminary and coarse solution that can be used as input of our concurrent approach. The result of our simulation can be then refined again by an all-atom simulation. The output of each phase is a folded protein with an optimal folding for the specific energy model. Refining the model and starting from a folding closer to the native state, makes the next phase more effective, since the conformational space to be searched is smaller.

Moreover, we plan to include our simulation approach in the SICStus Prolog code developed in [9], which performs a minimization of a simplified contact interaction energy, in order to take advantage of the simulation tool for implementing a non blind traversing of the search space. Other concurrent implementations (e.g., in JCC [19] and in traditional concurrent Prolog languages, such as CIAO Prolog, CS-Prolog II, or Arity/Prolog32) will be carried on after the inclusion of UNRES energy function.

References

1. C. B. Anfinsen. Principles that govern the folding of protein chains. *Science*, 181:223–230, 1973.
2. R. Backofen. The protein structure prediction problem: A constraint optimization approach using a new lower bound. *Constraints*, 6(2–3):223–255, 2001.
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 in the space of contact maps. *BMC Bioinformatics*, 4(8), 2003.
5. B. R. Brooks et al. Charmm: A program for macromolecular energy minimization and dynamics calculations. *J. Comput. Chem.*, 4:187–217, 1983.
6. M. Carlsson, G. Ottosson, and B. Carlson. An open-ended finite domain constraint solver. In Proc. of *PLILP'97*, vol. 1292 of *Lecture Notes in Computer Science*, pp. 191–206. Springer-Verlag, Berlin, 1997.
7. P. Clote and R. Backofen. *Computational Molecular Biology: An Introduction*. John Wiley & Sons, 2001.
8. P. Crescenzi, D. Goldman, C. Papadimitrou, A. Piccolboni, and M. Yannakakis. On the complexity of protein folding. In *Proc. of STOC*, pp. 597–603, 1998.
9. A. Dal Palù, A. Dovier, and F. Fogolari. Protein folding in $CLP(\mathcal{FD})$ with empirical contact energies. In *Recent Advances in Constraints*, vol. 3010 of *Lecture Notes in Artificial Intelligence*, pp. 250–265, 2004.
10. A. Dovier, M. Burato, and F. Fogolari. Using secondary structure information for protein folding in $CLP(\mathcal{FD})$. In *Sel. papers from 11th WFLP*, vol. 76 of *Electronic Notes in Theoretical Computer Science*, 2002.
11. 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.
12. S. Haridi, P. Brand, E. Klintskog, and T. Sjöland. Using Mozart for Modelling and Simulation of TCCP. In Proc. of *4th Int'l Workshop on Constraint programming for time critical applications—ESPRIT working group COTIC*, 2000.
13. A. Liwo, J. Lee, D. R. Ripoll, J. Pillardy, and H. A. Scheraga. Protein structure prediction by global optimization of a potential energy function. In *Proceedings of the National Academy of Science (USA)*, vol. 96, pages 5482–5485, 1999.
14. 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.
15. 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.
16. A. Neumaier. Molecular modeling of proteins and mathematical prediction of protein structure. *SIAM Review*, 39:407–460, 1997.
17. D. Qiu, P. Shenkin, F. Hollinger, and W. Still. The gb/sa continuum model for solvation. A fast analytical method for the calculation of approximate born radii. *J. Phys. Chem.*, 101:3005–3014, 1997.
18. V. A. Saraswat, M. C. Rinard, and P. Panangaden. Semantic Foundations of Concurrent Constraint Programming. Proceedings of *18th ACM POPL*, 1991.
19. V. A. Saraswat, R. Jagadeesan, and V. Gupta. JCC: Integrating timed default concurrent constraint programming into Java. In *Proceedings of EPIA 2003*, vol. 2902 of *Lecture Notes in Computer Science*, pages 156–170, 2003.

20. J. Skolnick and A. Kolinski. Reduced models of proteins and their applications. *Polymer*, 45:511–524, 2004.
21. Univ. des Saarlandes, Sweedish Inst. of Computer Science, and Univ. Catholique de Louvain. The Mozart Programming System. `www.mozart-oz.org`.
22. 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.