A New Constraint Solver for 3D Lattices and its Application to the Protein Folding Problem

Alessandro Dal Palù[1], Agostino Dovier[1], and Enrico Pontelli[2]

[1]Dipartimento di Matematica e Informatica
Università di Udine
(dalpalu|dovier)@dimi.uniud.it

[2]Department of Computer Science
New Mexico State University
epontell@cs.nmsu.edu

Abstract. The paper describes the formalization and implementation of an efficient constraint programming framework operating on 3D crystal lattices. The framework is motivated and applied to address the problem of solving the ab-initio *protein structure prediction* problem—i.e., predicting the 3D structure of a protein from its amino acid sequence. Experimental results demonstrate that our novel approach offers up to a 3 orders of magnitude of speedup compared to other constraint-based solutions proposed for the problem at hand.

1 Introduction

In this paper we investigate the development of a generic *constraint framework* for discrete three dimensional (3D) crystal lattices. These lattice structures have been adopted in different fields of scientific computing [7, 15], to provide a manageable discretization of the 3D space and facilitate the investigation of physical and chemical organization of molecular, chemical, and crystal structures. In recent years, lattice structures have become of great interest for the study of the problem of computing approximations of the folding of protein structures in 3D space [20, 3, 11, 12, 15]. The basic values, in the constraint domain we propose, represent individual lattice points, and primitive constraints are introduced to capture basic spatial relationships within the lattice structure (e.g., relative positions, Euclidean and lattice distances). Variables representing those points can assume values on a *finite* portion of the lattice. We investigate constraint solving techniques in this framework, with a focus on propagation and search strategies.
The main motivation behind this line of research derives from the desire of more scalable and efficient solutions to the challenging problem of determining the 3D structure of globular proteins. The *protein structure prediction* (or *protein folding*) problem can be defined as the problem of determining, given the molecular composition of a protein (i.e., a list of amino acids, known as the *primary structure*), the three dimensional (3D) shape (*tertiary structure*) that the protein assumes in normal conditions in biological environments. Knowledge of the 3D protein structure is vital in many biomedical applications, e.g., for perfect drugs design and for pathogen detection. We allow as input some *secondary structure* knowledge (i.e., local 3D rigid conformations) that can be obtained directly from the primary sequence using predictors [19]. We can classify our problem as *ab-initio*, since there is no other input information.

In recent decades, most scientists have agreed that the answer to the folding problem lies in the concept of the *energy state* of a protein. The predominant strategy in solving

the protein folding problem has been to determine a state of the amino acid sequence in the 3D space with minimum energy state. According to this theory, the 3D conformation that yields the lowest energy state represents the protein's natural shape (a.k.a. the *native conformation*). The energy of a conformation can be modeled using *energy functions*, that determine the energy level based on the interactions between any pairs of amino acids [6]. Thus, we can reduce the protein folding problem to an optimization problem, where the energy function has to be minimized under a collection of constraints (e.g., derived from known chemical and physical properties) [9].

The problem is extremely complex and it can be reasonably simplified in several aspects, in order to reduce the overall complexity, without compromising the biological relevance of the solutions. A common simplification relies on the use of *lattice space models* to restrict the admissible positions of the amino acids in the space [1, 21, 20]. In this discrete space framework, the use of constraint solving techniques can lead to very effective solutions [11, 3].[1] Previous work conducted in this area relied on mapping the problem to traditional *Constraint Logic Programming over finite domains (CLP(FD))* (or making use of integer programming solutions [15]). In [11, 12], we showed that highly optimized constraints and propagators implemented in CLP allow us to achieve satisfactory performances on small/medium size instances, improving precision over previous models [3]. Unfortunately, the CLP(FD) libraries we explored (SICStus Prolog and ECLiPSe) proved ineffective in scaling to larger instances of the problem [12]. Furthermore, these libraries provided insufficient flexibility in implementing search strategies and heuristics that properly match the structure of our problem.

In this paper, we overcome the limitations of CLP(FD) encodings by implementing the protein folding problem in our novel lattice constraint programming framework. The novel solver is an optimized C program, that implements techniques for constraint handling and solution search, dealing directly with lattice elements—i.e., our native FD variables represent 3D lattice points (*lattice variables*). We include an efficient built-in labeling strategy for lattice variables and new search techniques for specific rigid objects (predicted secondary structure elements). The experimental results obtained show a dramatic improvement in performance (10^2–10^3 speedups w.r.t. SICStus 3.12.0 and ECLiPSe 5.8). We also implemented ideas and heuristics discussed through the paper and show our solver is robust enough to tackle proteins up to 100 amino acids and to produce acceptable quality solutions, given the model in use. We show that the encoding of the protein folding problem on *Face-Centered Cubic (*fcc*) lattices*, using our native lattice constraint framework, allows us to process significantly larger proteins than those handled in [11, 12], directly or by viewing them as clusters composed of known parts. The code discussed in the paper can be found at `www.dimi.uniud.it/dovier/PF`.

2 A new Constraint Solver on 3D Lattices

We describe a framework developed to solve *Constraint Satisfaction Problems* (CSPs) modeled on 3D lattices. The solver allows us to define lattice variables with associated domains, constraints over them, and to search the space of admissible solutions.

[1] Even with simple lattice models, the problem is NP-complete [10].

2.1 Variables and Domains

Crystal Lattices. A *crystal lattice* (or, simply, a lattice) is a graph (N, E), where N is a set of 3D points $(P_x, P_y, P_z) \in \mathbb{Z}^3$, connected by undirected edges (E). Lattices contain strong symmetries and present regular patterns repeated in the space. If all nodes have the same degree δ, then the lattice is said δ-*connected*. Given $A, B \in N$, we define:
• the *squared Euclidean distance* as: $eucl(A, B) = (B_x{-}A_x)^2{+}(B_y{-}A_y)^2{+}(B_z{-}A_z)^2$
• the *norm infinity* as: $norm_\infty(A, B) = \max\{|B_x - A_x|, |B_y - A_y|, |B_z - A_z|\}$
In this work, we focus on fcc lattices, where:

$N = \{(x, y, z) \mid x, y, z \in \mathbb{Z} \text{ and } x + y + z \text{ is even}\}$ and
$E = \{(P, Q) \mid P, Q \in N, eucl(P, Q) = 2\}$.

Lattice points lie on the vertices and on the center point of each face of cubes of size 2 (Fig. 1). Points at Euclidean distance $\sqrt{2}$ are connected and their distance is called *lattice unit*. Two points are in *contact* iff their Euclidean distance is 2. This lattice is 12-connected. In [17] it is shown that the fcc model is a well-suited, realistic model for 3D conformations of proteins.

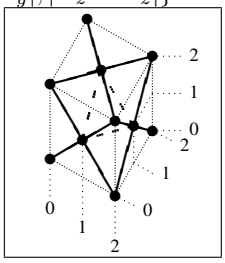

Fig. 1: An fcc-cube.

Domains. A *domain* D is described by a pair of lattice points $\langle \underline{D}, \overline{D} \rangle$, where $\underline{D} = (\underline{D}_x, \underline{D}_y, \underline{D}_z)$ and $\overline{D} = (\overline{D}_x, \overline{D}_y, \overline{D}_z)$. D defines a *box*:

$$Box(D) = \left\{(x, y, z) \in \mathbb{Z}^3 \;:\; \underline{D}_x \le x \le \overline{D}_x \;\wedge\; \underline{D}_y \le y \le \overline{D}_y \wedge\; \underline{D}_z \le z \le \overline{D}_z \right\}$$

We only handle the bounds of the effective domain, since a detailed representation of all the individual points in a volume of interest would be infeasible (due to the sheer number of points involved). The approach follows the same spirit as the manipulation of finite domains using bounds consistency [2]. The choice of creating a single variable representing a three dimensional point is driven by the fact that consistency is less effective when independently dealing with individual coordinates [16]. We say that D is *admissible* if $Box(D)$ contains at least one lattice point; D is *ground* if it is admissible and $\underline{D} = \overline{D}$; D is *empty* (*failed*) if D is not admissible. We introduce two operations:

- *Domain intersection*: Given two domains D and E, their intersection is defined as follows: $D \cap E = \langle \uparrow (D, E), \downarrow (D, E) \rangle$ where:
$$\uparrow (D, E) = (\ \max\{\underline{D}_x, \underline{E}_x\}, \max\{\underline{D}_y, \underline{E}_y\}, \max\{\underline{D}_z, \underline{E}_z\}\ )$$
$$\downarrow (D, E) = (\ \min\{\overline{D}_x, \overline{E}_x\}, \min\{\overline{D}_y, \overline{E}_y\}, \min\{\overline{D}_z, \overline{E}_z\}\ )$$
- *Domain dilation*: Given a domain D and a positive integer d, we define the domain dilation operation (that enlarges $Box(D)$ by $2d$ units) $D + d$ as:
$$D + d = \langle (\underline{D}_x - d, \underline{D}_y - d, \underline{D}_z - d), (\overline{D}_x + d, \overline{D}_y + d, \overline{D}_z + d) \rangle$$

Each variable V, that represent lattice points, is associated to a *domain* $D^V = \langle \underline{D}^V, \overline{D^V} \rangle$.

2.2 Constraints

We define the following binary constraints on variables, based on spatial distances. Given two lattice variables V_1, V_2 and $d \in \mathbb{N}$, we define the constraints:

$$\texttt{CONSTR\_DIST\_LEQ}(V_1, V_2, d) \Leftrightarrow \exists P_1 \in B_1, \exists P_2 \in B_2 \textit{ s.t. } norm_\infty(P_1, P_2) \le d$$
$$\texttt{CONSTR\_EUCL}(V_1, V_2, d) \Leftrightarrow \exists P_1 \in B_1, \exists P_2 \in B_2 \textit{ s.t. } eucl(P_1, P_2) = d$$
$$\texttt{CONSTR\_EUCL\_LEQ}(V_1, V_2, d) \Leftrightarrow \exists P_1 \in B_1, \exists P_2 \in B_2 \textit{ s.t. } eucl(P_1, P_2) \le d$$
$$\texttt{CONSTR\_EUCL\_G}(V_1, V_2, d) \Leftrightarrow \exists P_1 \in B_1, \exists P_2 \in B_2 \textit{ s.t. } eucl(P_1, P_2) > d$$

where $B_1 = Box(D^{V_1})$, $B_2 = Box(D^{V_2})$, and P_1, P_2 are lattice points.

All the constraints introduced are bi-directional (i.e., symmetric). Nevertheless, for practical reasons, we treat them as directional constraints, using the information of the first (leftmost) domain to test and/or modify the second domain. Consequently, every time a constraint C over two variables has to be expressed, we will add in the constraint store both constraints $C(V_1, V_2, d)$ and $C(V_2, V_1, d)$. A *Constraint Satisfaction Problem (CSP)* on the variables $V_1, \ldots, V_n$ with domains $D^{V_1}, \ldots, D^{V_n}$ is a set of binary constraints of the form above. A solution of the CSP is an assignment of lattice points to the variables $V_1, \ldots, V_n$, such that the lattice points belong to the corresponding variable domains and they satisfy all the binary constraints.

Proposition: *The general problem of deciding whether a CSP in the lattice framework admits solutions is NP-complete.*

Proof [Sketch]: The problem is clearly in NP. To show the NP-hardness, we reduce the *Graph 3-Colorability Problem* of an undirected graph $G(V, E)$ in our CSP (we refer to cubic lattices. For other lattices, additional `CONSTR_EUCL_G` constraints might be required to identify 3 points in the box). For each node $n_i \in V$, we introduce a variable V_i with domain $D^{V_i} = \langle (0,0,0), (0,0,2) \rangle$. $Box(D^{V_i})$ contains three lattice points $(0, 0, j)$, corresponding to the color j. For every edge $e = (n_i, n_j)$, we add the constraint `CONSTR_EUCL_G`$(V_i, V_j, 0)$, that constrains the points represented by the variables to be at a distance greater than 0 (i.e., have a different color). □

The *constraint store* is a data structure used to implement a CSP, representing constraints, variables, and their domains. In our implementation, it is realized as a dynamic array. For efficiency, we also maintain, for each variable V_i, the adjacency list containing links to all the constraints $C(V_i, V_j)$—those that have to be considered after a modification of the domain of D^{V_i}.

2.3 Constraint Solving

We modeled the solver considering the constrain phase separated from the search phase. Thus, neither new variables nor constraints can be added during the search.

Propagation and Consistency. The constraint processing phase is based on propagating the constraints on the bounds of the domains in the 3 dimensions at the same time, i.e., modifying the boxes of the domains.

The constraint `CONSTR_DIST_LEQ`(A, B, d) states that the variables A and B are distant no more than d in $norm_\infty$. It can be employed to simplify domains through bounds consistency. The formal rule is: $D^B = (D^A + d) \cap D^B$.

The constraint `CONSTR_EUCL_LEQ`(A, B, d) states that A and B are at squared euclidean distance less than or equal to d. The sphere of radius $\sqrt{d}$, that contains the admissible values defined by the constraint, can be approximated by the minimal surrounding box that enclose it (a cube with side $2\lfloor\sqrt{d}\rfloor$). The formal propagation rule is: $D^B = (D^A + \lfloor\sqrt{d}\rfloor) \cap D^B$. This rule can also be applied in the case of the `CONSTR_EUCL` constraint (this constraint implies `CONSTR_EUCL_LEQ`). The constraint `CONSTR_EUCL_G` does perform any propagation. We also assume that an eventual cost function (to be optimized during the search for solutions) does not propagate any information to the domains and thus it is handled as simple evaluation function.

Propagation is activated whenever the domain of a variable is modified. Let us consider a situation where the variables $G = \{V_1, \ldots, V_{k-1}\}$ have been bound to specific

values, V_k is the variable to be assigned next, and let $NG = \{V_{k+1}, \ldots, V_n\}$ be all the remaining variables. The first step, after the labeling of V_k, is to check for consistency the constraints of the form $C(V_k, V_i)$, where $V_i \in G$ (this is the *node consistency* check). The successive *propagation* phase is divided in two steps. First, all the constraints of the form $C(V_k, V_j)$ are processed, where $V_j \in NG$. This step propagates the new bounds of V_k to the variables not yet labeled. Thereafter, bounds consistency, using the same outline of AC-3 [2], is applied to the constraints of the form $C(V_i, V_j)$, where $V_i, V_j \in NG$. We carefully implemented a constant-time insertion for handling the set of constraints to be revisited, using a combination of an array to store the constraints and an array of flags for each constraint. This leads to the following result:

Proposition: *Each propagation phase has a worst-case time complexity of* $O(n + ed^3)$*, where* n *is the number of variables involved,* e *is the number of constraints in the constraint store, and* d *the maximum domain size.*

Proof [Sketch]: Let us assume that the variable V_i is labeled. Each propagation for a constraint costs $O(1)$, since only arithmetic operations are performed on the domain of the second variable. Let us assume that for each pair of variables and type of constraint, at most one constraint is deposited in the constraint store (it can be guaranteed with an initial simplification). In the worst case, there are $O(n)$ constraints of the form $C(V_i, V_j, d)$, where V_j is not ground. Thus, the algorithm propagates the new information in time $O(n)$, since each constraint costs constant time. The worst-case time complexity of AC-3 procedure is $O(ed^3)$, where e is the number of constraints in the constraint store and d the maximum domain size. □

Handling the Search Tree. The search procedures are implementations of a standard *backtracking+propagation* search procedure [2]. The evolution of the computation can be depicted as the construction of a search tree, where the internal nodes correspond to guessing the value of a variable (*labeling*) while the edges correspond to propagating the effect of the labeling to other variables (through the constraints). We implement two *variable selection* strategies: a *leftmost* strategy—it selects the leftmost uninstantiated variable for the next labeling step—and a *first-fail* strategy—it selects the variable with the smallest domain size, i.e., the box with the smallest number of lattice points. The process of selecting the value for a variable V relies on D^V, on the structure of the underlying lattice and on the constraints present. E.g., in a fcc lattice, if V is known to be only 1 lattice unit from a specific point in the lattice, then it has only 12 possible placements, that can be tested directly, instead of exploring the full content of $Box(D^V)$.

At the implementation level, the current branch of the search tree is stored into an array; each element of the array represents one level of the current branch. A value-trail stack is employed to keep track of variables modified during propagation, and used to undo modifications during backtracking. Moreover, we allow the possibility of collapsing levels of the search tree, by assigning a *set* of (related) variables in a single step. This operation is particularly useful when dealing with variables that represent points that are part of a secondary structure element.

2.4 Bounded Block Fails heuristic

We present a novel heuristic to guide the exploration of the search tree, called *Bounded Block Fails (BBF)*. This technique is *general* and can be applied to every type of search,

though it is particularly effective when applied to the protein folding problem [12]. The heuristic involves the concept of *block*. Let $\hat{V}$ be a list $[V_1, \ldots, V_n]$ of variables and constants (i.e., ground variables). The collection of variables in $\hat{V}$ is partitioned in *blocks* of fixed size k, such that the concatenation of all the blocks $B_1 B_2 \ldots B_\ell$ gives the ordered list of non ground variables in $\hat{V}$, where $\ell \leq \lceil \frac{n}{k} \rceil$. The blocks are dynamically selected, according to the variable selection strategy and the state of the search. Fig. 2 shows an example for a list of 9 variables and $k = 3$. Dark boxes represent ground variables.

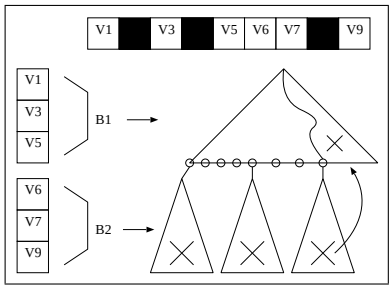

Fig. 2: The BBF heuristics

The heuristics consists of splitting the search among the ℓ blocks. Within each block B_i, the variables are individually labeled. When a branch in block B_i is completely labeled, the search moves to the successive block B_{i+1}, if any. If the labeling of the block B_{i+1} fails, the search backtracks to the block B_i. Here there are two possibilities: if the number of times that B_{i+1} completely failed is below a certain threshold t_i, then the process continues, by generating one more solution to B_i and re-entering B_{i+1}. Otherwise, if too many failures have occurred, then the BBF heuristic generates a failure for B_i as well and backtracks to a previous block. Observe that the count of the number of failures includes both the regular search failures as well as those caused by the BBF strategy. The list $t_1, \ldots, t_\ell$ of thresholds determines the behavior of the heuristic. In Fig. 2, $t_1 = 3$; note how, after the third failure of B_2, the search on B_1 fails as well.

BBF is an incomplete strategy, i.e., it can miss the optimum. However intuition and experimental results suggest that it is effective in finding suboptimal solutions whenever they are spread in the search tree. In these cases, we can afford to skip solutions when generating block failure, because others will be discovered following other choices in earlier blocks. In the context of searching for solutions in 3D lattices, a failure in the current branch means that the partial spatial structure constructed so far (by placing variables in the lattice) does not allow to proceed without violating some constraints. The BBF heuristic suggests to revise earlier choices (i.e., a "more drastic" revision of the structure built so far) instead of exploring the whole space of possibilities depending on the block that collects failures (i.e., a "more local" revision of the structure). The high density and the large number of admissible solutions typically available in the type of lattice problems we consider, permit to exclude some solutions, depending on the threshold values, and to still be able to find almost optimal solutions in shorter time.

3 An Application: The Protein Folding Problem on the fcc Lattice

A protein folds in the 3D space with a high degree of freedom and tends to reach the *Native* conformation (tertiary structure) with a minimal value of free energy. Native conformations are largely built from *secondary structure* elements (e.g., α-helices and β-sheets), often arranged in well-defined motifs.

In Fig. 3, α-helices (contiguous amino acids arranged in a regular right-handed helix) are in dark color and β-sheets (collections of extended strands, each made of contigu-

ous amino acids) in light color. Following similar proposals (e.g., [1, 3, 15]), we focus on fcc lattices. For details about the biological issu
lattice modeling see [11].
Let $\mathcal{A}$ be the set of amino acids ($|\mathcal{A}| = 20$). Given
mary) sequence $S = s_1 \cdots s_n$, with $s_i \in \mathcal{A}$, we re
with lattice variable V_i the *lattice position* of ami
s_i—i.e., the placement of the amino acid s_i in the
The modeling leads to the following constraints:

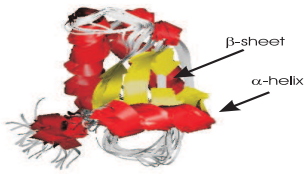

Fig. 3: Protein 1d6t native state

- for $i \in \{1, \ldots n-1\}$, $\texttt{CONSTR\_EUCL}(V_i, V_{i+1}, 2)$: adjacent amino acids in the primary sequence are mapped to lattice points connected by one lattice unit;
- for $i \in \{2, \ldots n-1\}$, $\texttt{CONSTR\_EUCL\_LEQ}(V_{i-1}, V_{i+1}, 7)$: three adjacent amino acids may not form an angle of $180°$ in the lattice;
- for $i, j \in \{1, \ldots n\}$, $|i-j| \geq 2$, $\texttt{CONSTR\_EUCL\_G}(V_i, V_j, 4)$: two non-consecutive amino acids must be separated by more than one lattice unit (no overlaps), and angles of $60°$ are disallowed for three consecutive amino acids.

In fcc, the angle between three consecutive amino acids can assume only values $60°$, $90°$, $120°$, and $180°$, but volumetric constraints make values $60°$ and $180°$ infeasible. The following additional constraints are also introduced [11]:

- $\texttt{CONSTR\_DIST\_LEQ}(V_i, V_j, 4)$ are added whenever the presence of a ssbond between the amino acids s_i and s_j is known; the ssbond (*disulfide bridge*) is a predictable limit on the distance in space between pairs of amino acids.
- $\texttt{CONSTR\_DIST\_LEQ}(V_i, V_j, \mathsf{cf} \cdot n)$ are added, where cf is the *compact factor*, expressed as a number between 0 and 1, and n is the protein length. The compact factor establishes an approximated maximal distance between amino acids.

A *folding* ω of $S = s_1 \cdots s_n$ is an assignment of lattice points to the variables $V_1, \ldots, V_n$ that is a solution of the CSP defined by the constraints above.

A simplified evaluation of the *energy* of a folding can be obtained by observing the *contacts* present in the folding. Each pair of non-consecutive amino acids s_i and s_j in contact (i.e., at Euclidean distance 2) provide an energy contribution, described by the commutative function $\mathsf{Pot}(s_i, s_j)$ [11]. These contributions can be obtained from tables developed using statistical methods applied to structures obtained from X-Rays and NMR experiments [6]. Finally, the *protein structure prediction problem* can be modeled as the problem of finding the folding ω of S such that the following energy cost function is minimized:

$$E(\omega, S) = \sum_{1 \leq i < n} \sum_{i+2 \leq j \leq n} \mathsf{contact}(\omega(V_i), \omega(V_j)) \cdot \mathsf{Pot}(s_i, s_j).$$

The function contact takes two lattice points and returns a value in $\{0, 1\}$:

$$\mathsf{contact}(A, B) = 1 \Leftrightarrow eucl(A, B) = 4.$$

Together with the primary sequence S, we allow input knowledge about presence of specific secondary structure elements (e.g., helices). These could be determined, for example, using standard secondary structure prediction systems (e.g., PHD or PSI-pred). This information can be used to impose several local constraints forcing a sequence of points to assume the a rigid spatial form. In this paper, we view each rigid object (an helix or a sheet) as a unique high-level disjunctive constraint, which is automatically activated when one point in the secondary structure is labeled.

3.1 Variable Instantiation in the fcc lattice

Once the constraints have been set up, the search phase is initiated. Different strategies are employed to prune the search space at this stage.

If the variable V_{i-1} (with first fail strategy also V_{i+1}) is ground and the variable V_i has to be instantiated, it turns out that there are only 12 possible assignments allowed by fcc—being the lattice 12-connected, and consecutive amino acids are connected by exactly one lattice unit. Thus, it is convenient to expand the search tree for only those 12 assignments that are compatible with the current domain of V_i. A more particular case, but very common, occurs when V_{i-1} and V_{i-2} (V_{i+1} and V_{i+2}) are both ground. In this case, the interaction of all constraints limits the values of V_i to at most 6 possible assignments—and only those that are also present in $Box(D^{V_i})$ are used to expand the search tree. This lattice dependent instantiation scheme allows us to directly assign feasible values, reducing the number of consistency checks. The use of this strategy leads to a speedup of 2-3 times w.r.t. a labeling that explores all points in $Box(D^{V_i})$.

Another labeling strategy relates to the handling of secondary structure elements. When the first variable belonging to a rigid object is labeled, all the other variables in it are assigned, according to precomputed patterns that describe every possible orientation of the secondary structure elements in the lattice. After a point in the secondary structure is labeled, there are only 24 possible assignments for the *whole* rigid object, due to the lattice constraints and symmetries. To save unnecessary work, moreover, after the consistency checks are performed, the bounds consistency procedure is run only once, after the propagation from the newly labeled variables has been completed.

3.2 Pruning Minimal Contacts Heuristic

In this section, we present a branch and bound (BB) strategy, adapted to the specific needs of the protein folding problem. In the case of the protein folding problem, a generic branch and bound scheme, based on the estimation of the energy of the conformation, proved to be rather ineffective with large input sizes. Our intuition is that the cost function can collect many contributions at the very end of a branch and drastically change its value. This behavior is particularly evident when processing large proteins. As a result, the prediction of the bounds for the energy function, computationally expensive, reveals to be potentially inaccurate.

We adopted a more coarse and constant time cost estimation. The strategy we propose implements branch and bound using the *number of contacts* generated by the given conformation as the information to perform pruning. In general, the global energy and the number of contacts are strongly related. Nevertheless, since the energy function is composed of weighted contributions of amino acids in contact, the two values may occasionally diverge.

The computation of estimates of the number of contacts is facilitated by the peculiar properties of the fcc lattice; e.g., each amino acid can form at most 3 contacts with other ones. When a new best conformation is found, we compute the number c of contacts realized. Assuming that, in the worst case, the last amino acids to be labeled generate 3 contacts each, at $c/3$ levels before the leaves, each subtree can be safely pruned whenever the number of contacts is less than c. This heuristic can be computed in

constant time since, given a partial assignment, an upper bound for possible contacts is immediately known. Since the energy is not precisely expressed by the number of contacts, we cannot guarantee the completeness of the heuristics. Nevertheless, empirical tests showed that this is not a significant problem; our experiments indicated also that the pruning of the last levels of the tree provides significant speedup during search.

In Table 1, we show some experimental tests of enumeration of the complete search tree, with and without the pruning heuristic presented above (under Windows on an AMD Duron 1.0GHz). For each protein ID of length N, we run a complete enumeration and then perform the same search with the contacts heuristic activated. In all cases, the heuristic improves time and reduces the number of nodes explored, without significantly changing the optima discovered.

		Enumeration			BB Heuristic		
ID	N	Energy	Nodes	Time	Energy	Nodes	Time
1kvg	12	-6,881	318,690	0.851s	-6,881	124,722	0.540s
1le0	12	-4,351	1,541,107	4.015s	-4,351	487,105	1.842s
1le3	16	-5,299	1,544,830	5.938s	-5,299	439,969	2.513s
1edp	17	-12,279	20,491	0.140s	-12,279	8,726	0.120s
1pg1	18	-10,352	56,934	0.280s	-10,352	7,908	0.140s
1zdd	34	-12,315	268,061	5.037s	-12,097	68,428	3.805s

Table 1: Effectiveness of contact pruning heuristic

4 Results and comparisons

Efficiency Analysis. The first test we discuss is designed to benchmark the speed of our solver. Our goal is to compare the solution to the protein folding problem using our lattice solver with the solution obtained by mapping the problem to finite domain constraints—using SICStus 3.12.0 (`clpfd`) and ECLiPSe 5.8 (`ic`). We run complete enumerations of the search tree using the first-fail strategy. To perform a fair comparison, we did not make use of branch and bound strategies in any of the implementations. We implement the protein structure prediction problem in SICStus and ECLiPSe using the best formalization we developed in [12]. In our solver, we implement the equivalent sets of constraints, reported in Section 3, but expressed in terms of finite domains. In Table 2, we compare the running times required to explore the whole search space. In the first column, we report the protein selected, in the second the time (in seconds) required by the lattice solver to explore the search tree, while the last two columns report the corresponding running times using SICStus and ECLiPSe (in brackets the speedup w.r.t. the lattice solver). For these examples, we use proteins whose search tree can be exhaustively explored in a reasonable time. These tests are performed using Windows (Pentium P4, 2.4GHz, 256Mb RAM). Table 2 shows that the choices made in the design and implementation of the new solver allow us to gain speedups in the order of 10^2–10^3 times w.r.t. standard general-purpose FD constraint solvers. Moreover, our implementation is robust and scales to large search trees with a limited use of memory. These positive results have also an interesting side-effect: the solver allows us to quickly collect the entire pool of admissible conformations for small proteins.

ID	Our	SICStus		ECLiPSe	
1edp	0.063s	8.92s	(142x)	1m.5s	(1039x)
1pg1	0.156s	16.00s	(103x)	1m.50s	(704x)
1kvg	0.406s	40.81s	(101x)	4m.22s	(646x)
1le0	1.922s	6m.31s	(203x)	33m.13s	(1036x)
1le3	2.859s	9m.46s	(205x)	59m.37s	(1251x)
1zdd	2.437s	insuff. memory		$>$ 8h.	($>$10000x)

Table 2: Complete Search

Quality of the Results. We analyze the foldings produced by our solver for proteins for which the native conformation is known. In our case, we consider proteins with known conformation from the PDB database [5]. Different ingredients come into play: the use of a simplified spatial model (fcc in our case), the use of a simplified energy function, and the use of a simplified protein model. Clearly, we cannot compare directly our results to the ones deposited in the PDB. In [11], we showed how to enrich fcc predictions to a solution relaxed in the continuous space. Only after that step a direct comparison with the original protein in the PDB is meaningful. Since in these tests we do not apply any refinement to our fcc solutions, we introduce a new *quality measure*, in order to mask the errors induced by the use of the lattice. We analyze the foldings as follows. We map an original protein from the PDB onto the fcc lattice, using the usual constraints for an admissible conformation. Moreover, to reproduce the same shape on the lattice, we add a set of distance constraints for each pair of amino acids taken from the original protein. The distance constraints are relaxed to a range of possible distances allowed for each pair, in order to allow the protein to find a placement in the discretized space. This process produces a *set of admissible foldings* that are very close to the original protein. These PDB over fcc proteins are the best representatives on fcc of quasi-optimal foldings according to the native conformation. Since it is not possible to collect the complete set of solutions, due to time complexity, we select, as representatives of the complete set, the enumeration of the first $1,000$ conformations found. Out of this set we identify the best conformation evaluated according to the *comparison function* introduced below.

The function used to compare the quality of the foldings cannot be the energy function used in the minimization process, since it accounts only for local contacts. We also decided not to use a standard RMSD[2] measure of spatial positions. This measure, in fact, computes only the deviation of corresponding positions between two conformations, and does not take into account other properties of the amino acids being compared. In our specific case, we want to include also the specific energy contribution carried by every pair of amino acids. We developed a *comparison function* that includes all these properties; basically the function is a more refined extension to continuous values of the contact energy function. The comparison of two conformations is reduced to comparing the values returned by the comparison function applied to the two conformations. The comparison function is as follows:

$$compare(S, \omega) = 4 \cdot \sum_{i \neq j} contrib(i, j) \; / \; \sqrt{eucl(\; \omega(i), \omega(j) \;)}$$

where S is the sequence of amino acids and ω is the conformation. The function is normalized w.r.t. the distance of a contact (i.e., 4). The function is a continuous extension of our energy model, and it is tolerant to small changes in positions of amino acids, compensating for the differences of the spatial and energetic models.

In Table 3, we compare the evaluations with the *comparison function* for different proteins; the `Our` column reports the value of the comparison function applied to the best folding obtained from our solver, using a complete search; the `PDB` (1) column reports the value for the best mapping of the PDB protein on the fcc lattice. The `PDB` (2) column reports the value for the original protein as in the PDB. This is useful to compare how much the protein is deformed when placed on the lattice.

[2] *Root-Mean-Square Deviation*, a typical measure of structural diversity.

It is interesting to discuss these data, since our previous implementations [11, 12] could not terminate a complete enumeration in reasonable time. The results indicate that the values are indeed very close. It is important to remember that we are constrained to fold the proteins on the lattice structure, and thus the values are expected to be closer to (1) than (2). In general, (2) should be an upper bound for (1). Moreover, note that our best folding on lattice is often better than the corresponding mapping from PDB to fcc. This is due to the fact that the pool of conformations used in computing the PDB on fcc mapping is not complete, and the constraints used in the two approaches are different. Visually, the predicted conformations are very close to the corresponding original ones (e.g., Fig. 4).

ID	Our	PDB (1)	PDB (2)
1kvg	-19,598	-17,964	-28,593
1le0	-11,761	-12,024	-16,030
1le3	-20,192	-14,367	-21,913
1edp	-46,912	-38,889	-48,665
1pg1	-44,436	-39,906	-58,610
1zdd	-64,703	-63,551	-69,571
1e0n	-57,291	-54,161	-60,728

Table 3: `compare` applied to best, PDB on fcc and PDB folding

For medium and large size proteins, determining the optimal folding is computationally infeasible. When computing an *approximated* solution for the folding of a protein, it is also important to relate the result of the computation to the optimal solution, in order to evaluate the impact of the pruning strategies adopted. Once again we use the scheme presented above to estimate the quality of our solutions—by comparing how far our heuristic landed from the hypothetical optimal solution.

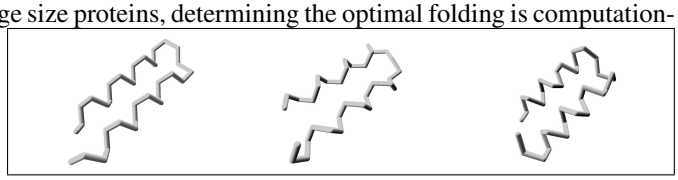

Fig. 4: Protein 1ZDD: our solution, fcc on PDB mapping and PDB

Heuristics tests. To show the power of our constraint solver in handling ad-hoc search heuristics, we test a set of selected proteins, with lengths ranging from 12 to 104. Table 4 reports the results of the executions; the Table indicates the PDB protein name, the protein length (n) in terms of amino acids, the BBF thresholds value assigned to $t_1 = \cdots = t_\ell$, the time to complete the search, the evaluation of the comparison function applied to the best solution, to the PDB on fcc, and to the original PDB. For BBF, we decided to define the block size equal to $\lfloor n/12 \rfloor + 1$ for $n \leq 48$ and equal to 5 for larger proteins. We empirically noticed that larger block sizes provide less accurate results, due to the higher pruning when failing on bigger blocks.

Proteins with more than 100 amino acids can be handled by our solver. This result is improved over the capabilities of the previous proposed frameworks (60 [11] and 80 [12] amino acids). This improvement is non-trivial, because of the NP-completeness of the problem at hand. The new heuristics provide more effective pruning of the search tree, and allow to collect better quality solutions. The tradeoff between quality and speed is controlled by the BBF threshold: higher values provide a more refined search and higher quality solutions. Moreover, the quality comparisons between our folding and the mapping of PDB on fcc and PDB itself, reveal that our solutions, even for larger proteins, are comparable to foldings of PDB on fcc. Note also that, for larger proteins, the size of pool of the selected solutions for PDB on fcc mappings, becomes insufficient, i.e., the difference of comparison function from the PDB value becomes

ID	n	CF	BBF	Time	Energy	PDB on fcc	PDB
1kvg	12	0.94	50	0.16s	-19,644	-17,964	-28,593
1edp	17	0.76	50	0.04s	-46,912	-38,889	-48,665
1e0n	27	0.56	50	1.76s	-52,558	-51,656	-60,728
1zdd	34	0.49	50	0.80s	-63,079	-62,955	-69,571
1vii	36	0.48	50	4.31s	-76,746	-71,037	-82,268
1e0m	37	0.47	30	19m57s	-72,434	-66,511	-81,810
2gp8	40	0.45	50	0.27s	-55,561	-55,941	-67,298
1ed0	46	0.41	50	8.36s	-124,740	-118,570	-157,616
1enh	54	0.37	50	45.3s	-122,879	-83,642	-140,126
2igd	60	0.35	20	2h42m	-167,126	-149,521	-201,159
1sn1	63	0.18	10	58m53s	-226,304	-242,589	-367,285
1ail	69	0.32	50	2m49s	-220,090	-143,798	-269,032
1l6t	78	0.30	50	1.19s	-360,351	-285,360	-446,647
1hs7	97	0.20	50	35m16s	-240,148	-246,275	-367,687
1tqg	104	0.15	20	10m35s	-462,918	-362,355	-1,242,015

Table 4. BBF experimental results (Linux, 2.8MHz, 512Mb RAM).

significant. For large proteins, it is an open problem in the literature how to precisely estimate the errors arising from discretizing the protein structure in a lattice space.

Scalability. A distinct advantage of our approach is its ability to readily use additional knowledge about known components of the protein in the resolution process, as long as they can be expressed as lattice constraints. In particular, some proteins, like hemoglobin, are constructed of a cluster of subunits, whose structure is known and already deposited in the PDB (or can be predicted). This approach follows the evolution of proteins, i.e., combination of already existing pieces into new bigger blocks. Often biologists explore unknown proteins by extracting the structure of sub-blocks by *homology* from the PDB. Our constraint-based approach can easily take advantage of the known conformations of the subsequences, treated as rigid spatial objects described by constraints, to determine the overall conformation of the protein. This ability is lacking in most other approaches to the problem; our previous finite domains encodings cannot handle proteins with more than 100 amino acids.

To study the scalability of our solver, we report some tests on artificial proteins having a structure of the type XYZ, i.e., composed of two known subsequences (X and Z), while Y is a short connecting sequence. We can show that our framework can easily handle proteins of size up to $1,000$ amino acids. We run some complete enumerations varying the length of Y and the proteins used as pattern for X and Z.

In our tests, we load the proteins X and Z as predicted in Table 4. We link them with a coil of amino acids with length $|Y|$ (leaving X and Z free of moving in the lattice as rigid objects). The search is a simple enumeration using Leftmost variable selection. Table 5 (a) shows that the computational times are extremely low, and dominated by the size of Y, instead of the size of XYZ. In the second part of the Table, we consider proteins constructed as follows: we start with X and Z equal to the 1E0N protein (whose folding can be optimally computed), and every successive test makes use of $X' = Z' = XYZ$—i.e., at each experiment we make use of the results from the previous experiment. This approach allowed us to push the search to sequences of size

X	Z	$\|X\|$	$\|Y\|$	$\|Z\|$	Time
1e0n	1e0n	27	5	27	11.3s
1e0n	1e0n	27	6	27	1m5s
1ail	1ail	69	5	69	1m25s
1ail	1ail	69	6	69	7m52s
1hs7	1hs7	97	5	97	3m7s
1hs7	1hs7	97	6	97	16m25s
1e0n	1e0n	27	3	27	0.40s
1e0n-2	1e0n-2	57	3	57	1.92s
1e0n-4	1e0n-4	117	3	117	9.26s
1e0n-8	1e0n-8	237	3	237	29.7s
1e0n-16	1e0n-16	477	3	477	1m48s

ID	Nodes	Time
1pg1	1.00	1.34
1kvg	1.95	2.39
1le0	1.00	1.06
1le3	1.02	1.16
1edp	2.96	2.00
1zdd	1.30	2.18

Table 5. From left to right, processing proteins XYZ (a), and ratios sphere/box approach (b)

up to $1,000$ amino acids. In these experiments, our concern is not only the execution time, but the ability of the solver to make use of known structures to prune the search tree.

Boxes vs Spheres. We tested a different formalization of the variables domains, where domains are represented as spheres instead of using Box. We reimplemented in our solver the domain description of a variable in terms of a center and a radius (with discrete coordinates) and the definition of an intersection of spheres as the smallest sphere that includes them. The idea is that a sphere should be more suitable to express the propagation of euclidean distance constraints. Unfortunately, results reported in Table 5 (b) show that this idea is not successful. The Table reports in the first column the test protein used, in the second the ratio of visited nodes in the search tree between sphere over box implementations. The last column provides the ratio of computation times between the two implementations. In particular, note that many more internal nodes are expanded in the sphere implementation. There are two reasons for this. First, computing spheres intersection is more expensive than intersecting boxes. Second, often two intersecting spheres are almost tangent. In this case the correct intersection is approximated by another sphere that includes a great amount of discarded volume.

5 Related Works

The problem of protein structure prediction is a fundamental challenge [20] in molecular biology. An abstraction of the problem, that has been investigated, is the ab-initio problem in the *HP* model, where amino acids are separated into two classes (H, *hydrophobic*, and P, *hydrophilic*). The goal is to search for a conformation produced by an *HP* sequence, where most HH pairs are neighboring in a predefined lattice. The problem has been studied on 2D square lattices [10, 15], 2D triangular lattices [1], 3D square models [15], and fcc lattices [17]. Backofen et al. have extensively studied this last problem [3, 4]. Integer programming approaches to this problem have also been considered [14]. The approach is suited for globular proteins, since the main force driving the folding process is the electrical potential generated by Hs and Ps, and the fcc lattices are effective approximations of the 3D space. Backofen's model has been extended in [11,

6], where the interactions between classes H and P are refined as interactions between every pair of amino acids, and modeling of secondary structures has been introduced.

The use of constraint technology in the context of the protein folding problem has been fairly limited. Backofen and Will used constraints over finite domains in the context of the *HP* problem [4]. Rodosek [18] proposed an hybrid algorithm which combines constraint solving and simulated annealing. Clark employed Prolog to implement heuristics in pruning a exhaustive search for predicting α-helix and β-sheet topology from secondary structure and topological folding rules [8]. Distributed search and continuous optimization have been used in ab-initio structure prediction, based on selection of discrete torsion angles for combinatorial search of the space of possible foldings [13]. Krippahl and Barahona [16] used a constraint-based approach to determine protein structures compatible with distance constraints obtained from NMR data.

In this work we adopted an approach different from the previous literature [3, 11, 12], where the modeling relied on traditional FD constraints. The description of a 3D lattice model using (single dimensional) FD-variables requires a complex interaction of constraints, in order to reproduce the natural correlation between the coordinates of the same lattice point. This leads to larger encodings with many constraints to be processed. Moreover, arc and bounds consistency reduce the domains one dimension at a time, and the system stores the explicit set of admissible (single-dimensional) points. Scalability is also hampered in this type of encodings. Our experience [11, 12] indicates that performance of these representations based on SICStus and ECLiPSe solvers is insufficient for larger instances of the problem.

The constraint model adopted in this paper is similar in spirit to the model used in [16]—as they also make use of variables representing 3D coordinates and box domains. The problem addressed in [16] is significantly different, as they make use of a continuous space model, they do not rely on a energy model, and they assume the availability of rich distance constraints obtained from NMR data, thus leading to a more constrained problem—while in our problem we are dealing with a search space of $O(6^n)$ conformations in the fcc lattice for proteins with n amino acids. Every modification of a variable domain, in our version of the problem, propagates only to a few other variables, and every attempt to propagate refined information (i.e., the good/no good sub-volumes of [16]) when exploring a branch in the search tree, is defeated by the frequent backtracking. Thus, in our approach we preferred a very efficient and coarse bounds consistency. The ideas of [16], i.e., restricting the space domains for rigid objects is simply too expensive in our framework (see [12]). We opted for a direct grounding of rigid objects, since in lattices there are few possible orientations. In our case, the position of objects can be basically anywhere, due to the lack of strong constraints. The techniques of [16] would be more costly and produce a poor propagation.

6 Conclusion and Future Work

We presented a formalization of a constraint programming framework on crystal lattice structures—a regular, discretized version of the 3D space. The framework has been realized into a concrete solver, with various search strategies and heuristics. The solver has been applied to the problem of computing the minimal energy folding of proteins

in the fcc lattice, providing high speedups and scalability w.r.t. previous solutions. The speedups derive from a more direct and compact representation of the lattice constraints, and the use of search strategies that better match the structure of the problem. We proposed general lattice (BBF) and problem-specific heuristics, showing how they can be integrated in our constraint framework to effectively prune the search space.

As future work, we plan to extend the investigation of search strategies and heuristics. We also propose to explore the use of parallelism to further improve scalability of the solution to larger instances of the problem.

Acknowledgments: This research has been supported by NSF grants 0220590, 0454066, and 0420407, by GNCS2005 project on constraints and their applications and by FIRB project RBNE03B8KK on protein folding.

References

1. R. Agarwala et al. Local rules for protein folding on a triangular lattice and generalized hydrophobicity in the HP model. *J. Computational Biology*, 275–296, 1997.
2. K. R. Apt. *Principles of constraint programming*. Cambridge University press, 2003.
3. R. Backofen. The protein structure prediction problem: A constraint optimization approach using a new lower bound. *Constraints*, 6(2–3):223–255, 2001.
4. R. Backofen and S. Will. A Constraint-Based Approach to Structure Prediction for Simplified Protein Models that Outperforms Other Existing Methods. *ICLP*, 2003, Springer Verlag.
5. H. M. Berman et al. The Protein Data Bank. *Nucleic Acids Research*, 28:235–242, 2000.
6. 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.
7. Center for Computational Materials Science, Naval Research Labs, *Crystal Lattice Structures*, `cst-www.nrl.navy.mil/lattice/`.
8. D. Clark et al. Protein topology prediction through constraint-based search and the evaluation of topological folding rules. *Protein Engineering*, 4:752–760, 1991.
9. P. Clote and R. Backofen. *Computational Molecular Biology*. John Wiley & Sons, 2001.
10. P. Crescenzi et al. On the complexity of protein folding. In *STOC*, pages 597–603, 1998.
11. A. Dal Palù, A. Dovier, and F. Fogolari. Constraint logic programming approach to protein structure prediction. *BMC Bioinformatics*, 5(186), 2004.
12. A. Dal Palù, A. Dovier, and E. Pontelli. Heuristics, Optimizations, and Parallelism for Protein Structure Prediction in CLP(FD). In Proc. of *PPDP'05*, 2005.
13. S. Forman. *Torsion Angle Selection and Emergent Non-local Secondary Structure in Protein Structure Prediction*. PhD thesis, U. of Iowa, 2001.
14. H. J. Greenberg et al. Opportunities for Combinatorial Optimization in Computational Biology. In *INFORMS Journal of Computing*, 2003.
15. W. Hart and A. Newman. The computational complexity of protein structure prediction in simple lattice models. CRC Press. 2003. (to appear).
16. L. Krippahl and P. Barahona. Applying Constraint Programming to Protein Structure Determination. In *CP'99*, Springer, 1999.
17. G. Raghunathan and R. L. Jernigan. Ideal architecture of residue packing and its observation in protein structures. *Protein Science*, 6:2072–2083, 1997.
18. R. Rodosek. A Constraint-based Approach for Deriving 3-D Structures of Cyclic Polypeptides. In *Constraints*, 6(2-3):257–270, 2001.
19. B. Rost. Protein Secondary Structure Prediction Continues to Rise. *J. Struct. Biol.* 134, 2001.
20. J. Skolnick et al. Reduced models of proteins and applications. *Polymer*, 45:511–524, 2004.
21. L. Toma and S. Toma. Folding simulation of protein models on the structure-based cubo-octahedral lattice with contact interactions algorithm. *Protein Science*, 8:196–202, 1999.