Computing Approximate Solutions of the Protein Structure Determination Problem using Global Constraints on Discrete Crystal Lattices

Alessandro Dal Palù[1], Agostino Dovier[2], Enrico Pontelli[3]

[1]Dip. di Matematica, Università di Parma
alessandro.dalpalu@unipr.it
[2]Dip. di Informatica, Università di Udine
dovier@dimi.uniud.it
[3]Dept. of Computer Science, New Mexico State University
epontell@cs.nmsu.edu

Abstract. Crystal lattices are discrete models of the three-dimensional space that have been effectively employed to facilitate the task of determining proteins' natural conformation. This paper investigates alternative *global constraints* that can be introduced in a constraint solver over discrete crystal lattices. The objective is to enhance the efficiency of lattice solvers in dealing with the construction of approximate solutions of the protein structure determination problem. Some of them (e.g., *self-avoiding-walk*) have been explicitly or implicitly already used in previous approaches, while others (e.g., the *density* constraint) are new. The intrinsic complexities of all of them are studied and preliminary experimental results are discussed.

1 Introduction

Discrete crystal lattices are regular discretizations of the three dimensional space. They have been originally introduced to provide formal descriptions of crystalline solids, where entities are composed of orderly arrangements of molecules, atoms, or ions, with definite and rigid shapes and with regularly defined faces. Typical discrete crystal lattices are composed of simple unit cells stacked together in 3-dimensions, repeated according to a predetermined pattern.

In more recent years, discrete crystal lattices have been found very useful to investigate the 3D conformation of protein structures, in particular in the context of energy landscape studies [25, 16, 2, 22, 1]. Commonly, Monte Carlo simulations, based on discrete spatial models, are run to design and test models of interactions between protein components [25]. In general, a simulation seeks a minimal entropy conformation, according to the energetic model in use.

Constraint solving is a programming paradigm that can be effectively employed to solve energy minimization problems, and thus, can be used to compute putative stable conformations. Examples of efficient, exhaustive and optimal search can be found in [2, 4]. Depending on the complexity of the model

in use, the search space and the solution time can grow dramatically. In these cases, constraint programming can be exploited to generate suboptimal candidates, making use of heuristics to guide the exploration of the search space [9]. Another remarkable feature of constraint programming is the capability to include additional experimental information as it becomes available. For example, Nuclear Magnetic Resonance data is successfully handled in [17, 18], rigid body information in [19], and secondary structure prediction in [9]. Moreover, statistical information retrieved by mining the protein data bank, e.g., a collection of rotamers, can be introduced to provide additional constraints in the search of conformations. The information can guide the search, discard infeasible conformations and improve the quality of the results.

Even though the resulting solutions are only approximations of the optimum, the time required for their generation is typically very low, compared to an abinitio molecular dynamics simulation. A viable framework is, thus, to integrate a fast screening using constraint programming and subsequently to refine the pool of candidate solutions using molecular dynamics simulations [9].

In these investigations, polymers are laid out in particular subsets of $\mathbb{N}^3$. These subsets are described by the vectors that specify the set of neighbors of each point. Lattice models, such as the Face-Centered Cube (`FCC`) and the chess knight are among those used in some of the existing studies. The protein folding problem in the context of discrete lattice structures has been studied as a *constraint optimization problem* in the `FCC` lattice, using a simplified energy model in [3] and with a more precise energy model in [9]. In these constraint-based approaches, each point P of the lattice is represented by a triplet of *finite domain variables* (P_x, P_y, P_z), where each variable describes a separate coordinate of the point. In [10], we addressed this problem and proposed a novel constraint solver, called *COLA (COnstraint solver on LAttices)*, whose primitive domain allows the representation of lattice points as atomic values, and this increases the 'propagation' capability of the solver. We demonstrated that COLA can produce more precise approximations of the 3D conformation of proteins in a significantly shorter amount of time. The constraints used in COLA are only *binary constraints*, i.e., constraints that express relations only between pairs of variables at a time.

Global constraints are proven constructs that facilitate the declarative encoding of problems. They allow the programmer to express knowledge about global relationships existing between several variables at the same time, that can be effectively employed by the search algorithm to prune infeasible parts of the solution search space, enabling significant improvements in performance. In this paper, we propose a study targeting the problem of dealing with global constraints in the general context of constraint solvers on lattice domain—and specifically in COLA.

We introduce different global constraints, all motivated by the representation of the protein structure determination problem as a constraint problem on discrete crystal lattices. We investigate the computational properties of these different global constraints, focusing in particular on the complexity of deciding

satisfiability and of the associated propagation process. Some of the global constraints we analyze have been implicitly or explicitly already used in encoding biological problems (e.g., *self-avoiding-walk* for the protein folding in [3] and [9]) with *ad-hoc* solutions for their propagation. However, their intrinsic complexities have never been precisely analyzed and formally compared. The implementation of several of these constraints have been discussed in the existing literature, and the investigation presented in this paper provides a formal justification of the perceived differences in expressive power and performance. Particular emphasis will be placed on the investigation of the *density* constraint, a novel global constraint that makes use of information produced from electron cryomicroscopy technology. For this constraint, we present preliminary experimental results, demonstrating encouraging performance.

The ultimate objective of this manuscript is to provide a uniform reference and formal investigation of global constraints in modeling and solving the protein structure determination problem. We hope that this paper will inspire further interest in this problem and promote discussion about suitable global constraints for discrete lattice structures and efficient implementation techniques.

All proofs are reported in the Appendix.

2 Lattices and COLA

A *discrete lattice* (or, simply, a *lattice*) is a graph (P, E), where P is a set of triples $(x, y, z) \in \mathbb{N}^3$, connected by undirected edges (E). Given $A = (x, y, z) \in P$, we will denote x, y, z with A_x, A_y, A_z respectively.

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 to be δ-*connected*. Three popular examples of lattices are described next, and illustrated in Figure 1.

A *cubic lattice* (P, E) is defined by the following properties:

- $P = \{(x, y, z) \mid x, y, z \in \mathbb{N}\}$;
- $E = \{(A, B) \mid A, B \in P,\ sqeucl(A, B) = 1\}$.

where $sqeucl(A, B) = (B_x - A_x)^2 + (B_y - A_y)^2 + (B_z - A_z)^2$.

The cubic lattice is 6-connected—see Figure 1(a). Cubic lattices are encountered, for example, in modeling various chemical compounds (e.g., cesium chloride).

A *face-centered cube (*`FCC`*)* lattice (P, E) is defined by the sets:

- $P = \{(x, y, z) \mid x, y, z \in \mathbb{N} \wedge x + y + z \text{ is even}\}$;
- $E = \{(A, B) \mid A, B \in P,\ sqeucl(A, B) = 2\}$.

In a `FCC` lattice we consider the 3D space organized in cubes, each side having length 2, and where the center point of each face is also admitted. The practical rule to compute the points belonging to the lattice is to check whether the sum of the point's coordinates (x, y, z) is even. Pairs of points at Euclidean distance $\sqrt{2}$ are linked and form the edges of the lattice; their distance is called *lattice unit*.

Observe that, for lattice units, it also holds that $|x_i - x_j| + |y_i - y_j| + |z_i - z_j| = 2$. The FCC lattice is 12-connected—see Figure 1(b).

A *chess knight* lattice is defined as follows:

- $P = \{(x, y, z) \mid x, y, z \in \mathbb{N}\}$;
- $E = \{(A, B) \mid A, B \in P,\ sqeucl(A, B) = 5\}$.

Each edge allows a move like a knight on a chessboard, i.e., 2 units in one direction, 1 in another direction, 0 in the third direction. The chess knight lattice is 24-connected—see Figure 1(c).

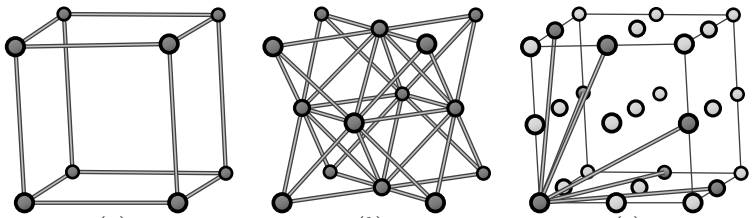


Fig. 1. Basic Component of a Cubic, FCC, and Chess Knight Lattices

In general, we model problems on discrete lattices as a collection of *unknowns* (i.e., *variables*) related by a collection of *constraints*—i.e., relations between such variables. The variables represent points that have to be placed in the lattice space, and the constraints commonly represent spatial constraints on the mutual positions of such points. A variable V is associated to a *domain* D^V, that represents the lattice points that can be legally used as placements for such variable.

In the case of COLA, a *domain* D is described by a pair of lattice points $\langle low(D), up(D)\rangle$. The domain D defines a set of lattice points in the 3D *box* identified by the two opposite vertices $low(D)$ and $up(D)$. COLA handles the domain operations of *intersection*, *union*, and *dilation*, described in [10]. In modeling a constraint satisfaction problem, each variable represents an entity to be placed in a point in the lattice space. Distance binary constraints based either on Euclidean distance *sqeucl* or on $norm_\infty(A, B) = \max\{|B_x - A_x|, |B_y - A_y|, |B_z - A_z|\}$ are admitted in COLA.

The work presented in [10] describes the implementation of these concepts in a concrete constraint solving system, capable of performing *bounds consistency* on the previously described constraints. The COLA solver has been applied to the problem of solving the protein folding problem in the FCC lattice, producing interesting results for proteins of length up to 100 amino acids.

3 Global constraints

The main contribution of this paper is the identification of which *global constraints* should be introduced in COLA to enhance its declarative nature and facilitate the efficient resolution of complex problems, in the domain of protein structure determination. In particular, we expect our design to be general and applicable to other constraint solvers on lattice domains. In order to be able to perform forms of consistency which are more accurate than bounds consistency (explored in our previous work [10]), we assume that the finite domain associated to each variable is a finite set of lattice points, instead of a simple box, as in COLA.

Intuitively, a global constraint is a non-binary constraint. More formally, given n variables $X_1, \ldots, X_n$, respectively having domains $D^{X_1}, \ldots, D^{X_n}$, a *global constraint* C on the variables $X_1, \ldots, X_n$ can be defined as a subset $C \subseteq D^{X_1} \times \cdots \times D^{X_n}$. For each global constraint C, we are interested in verifying two properties [6]:

- *(CON) Consistency:* $C \neq \emptyset$
- *(GAC) Generalized Arc Consistency:* $\forall i \in \{1, \ldots, n\}\, \forall a_i \in D^{X_i}$

$$\exists a_1 \in D^{X_1} \cdots \exists a_{i-1} \in D^{X_{i-1}} \exists a_{i+1} \in D^{X_{i+1}} \cdots \exists a_n \in D^{X_n}\ (a_1, \ldots, a_n) \in C$$

In the specific case where the constraint C is binary, i.e., it involves only two variables X_1, X_2, the GAC notion is known as *arc consistency (AC)*: C is arc-consistent iff

$$\forall a_1 \in D^{X_1}\ \exists a_2 \in D^{X_2}.\ (a_1, a_2) \in C \quad \wedge \quad \forall a_2 \in D^{X_2}\ \exists a_1 \in D^{X_1}.\ (a_1, a_2) \in C$$

Related to the notion of *GAC* is the notion of *filtering*, i.e., the problem of removing values from the domains of variables in order to obtain an equivalent constraint which is *GAC*. If the filtering is computationally too expensive, one can run fast approximated algorithms, that eliminate some values in some domains obtaining an equivalent constraint C', which is not, however, guaranteed to be GAC.

Other properties are of interest, e.g., generalized bounds consistency and directional arc/bounds consistency [14]. Nevertheless, in this paper, we only deal with the two properties described above, as they are more fundamental to the initial design of a realistic implementation of such global constraints.

Observe that, under the assumption that the domains are not empty, the definition of *GAC* implies *CON*. Thus, if we prove that testing *GAC* can be accomplished in time polynomial (in the size of the problem), the same will hold for *CON*. If testing *CON* can be shown to be NP-complete, then NP-hardness of testing *GAC* will immediately follow. Additionally, if we assume an explicit representation of the domains, then the NP-completeness of *CON* will actually imply the NP-completeness of *GAC*.

In the rest of the paper, we discuss different types of global constraints. These constraints, along with their main complexity characterization (for performing

constraint	`alldifferent`	`contiguous`	`saw`	`alldistant`
GAC	$O(dn^{1.5})$	$O(nd^2)$	NP-hard	NP-hard
constraint	`chain`	`rigid block`	`density`	
GAC	NP-hard	$O(nd)$	NP-hard[1]	

Table 1. Computational complexity of testing CON for the global constraints analyzed

CON), are summarized in Table 1[1]. We start with simple and general global constraints (such as the `alldifferent` constraint—see Section 3.1), and move towards constraints more closely tied to the properties of discrete lattices. In particular, the typical problems encoded on discrete lattices deal with finding adequate placements of objects in the lattice space. Placements require the entity to occupy contiguous locations in the lattice (`contiguous` constraint, Section 3.2), two parts of the same entity cannot be in the same location (`saw` constraint, Section 3.3), and components of the entity must maintain some minimum distance to account for the size of the entity (`alldistant` constraint, Section 3.4). Combinations of these conditions lead to more specialized global constraints (`chain` constraint, Section 3.5, and `block` constraint, Section 3.6). Finally, we present the `density` constraint (Section 4). Throughout the discussion, we will use the protein folding problem as guide and motivation.

3.1 The `alldifferent` constraint

The `alldifferent` constraint [26] is probably the best-known global constraint used in constraint programming. It is used to assert that a collection of variables are assigned pairwise distinct values. Its semantics is as follows: if $X_1, \ldots, X_n$ are variables with domains $D^{X_1}, \ldots, D^{X_n}$, then

$$\texttt{alldifferent}(X_1, \ldots, X_n) = (D^{X_1} \times \cdots \times D^{X_n}) \setminus \left\{(a_1, \ldots, a_n) \in (D^{X_1} \times \cdots \times D^{X_n}) : \exists i, j.\ (1 \leq i < j \leq n \wedge a_i = a_j)\right\}$$

It is well-known that testing the CON and GAC properties, as well as performing GAC filtering for the `alldifferent` constraint, can be done in polynomial time. These problems can be solved, for example, by adapting algorithms for bipartite graph matching (the first contribution in this direction is [24]).

The `alldifferent` constraint has a significant role in the modeling of the protein folding problem on discrete lattices [10]—to express the fact that a point in the lattice cannot be used to accommodate two distinct amino acids.

3.2 The `contiguous` constraint

The `contiguous` global constraint is used to describe the fact that a list of variables represent lattice points that are adjacent (in terms of positions in the

[1] Under certain restrictions the testing can be performed in polynomial time. More details are provided in Sect. 4.

lattice graph). Let E be the set of edges in a lattice, and let $X_1, \ldots, X_n$ be a list of variables (respectively, with domains $D^{X_1}, \ldots, D^{X_n}$). The `contiguous` constraint can be defined as follows:

$$\texttt{contiguous}(X_1, \ldots, X_n) = (D^{X_1} \times \cdots \times D^{X_n}) \setminus \\ \left\{(a_1, \ldots, a_n) \in (D^{X_1} \times \cdots \times D^{X_n}) : \exists i.\ (1 \leq i < n \wedge (a_i, a_{i+1}) \notin E)\right\}$$

Testing the GAC of `contiguous` can be done in polynomial time. In fact, the `contiguous` constraint is equivalent to the conjunction of the $n-1$ binary constraints of the form $C_{i,i+1}$, with $i \in \{1, \ldots, n-1\}$, such that

$$C_{i,i+1} = (D^{X_i} \times D^{X_{i+1}}) \setminus \{(a_i, a_{i+1}) : a_i \in D^{X_i} \wedge a_{i+1} \in D^{X_{i+1}} \wedge (a_i, a_{i+1}) \notin E\}$$

The graph induced by these constraint is acyclic. Thus, under these conditions AC implies GAC [15] (see also the proof in the Appendix).

Since AC for binary constraints can be tested in polynomial time, the same computational complexity will be maintained for GAC. Polynomiality of CON also follows directly.

The `contiguous` constraint is particularly relevant when modeling protein folding problems, since it allows us to state that the sequence of amino acids composing the primary sequence of a protein should remain contiguous in the discrete lattice.

3.3 The `saw` constraint

The `saw` constraint is used to describe the fact that each assignment to the variables $X_1, \ldots, X_n$ represents a *self-avoiding walk (SAW)* in the lattice. Formally, the constraint can be defined as follows:

$$\texttt{saw}(X_1, \ldots, X_n) = \texttt{contiguous}(X_1, \ldots, X_n) \cap \texttt{alldifferent}(X_1, \ldots, X_n)$$

The `saw` constraint can be used, for example, to model the fact that the primary sequence of a protein can not create cycles when placed in the 3D space. `saw` is called *SAWalk* in [4], where authors state: "*we are not aware of any efficient arc consistency algorithm for this combined constraint in the literature*". We now prove that such an algorithm does not exist.

Testing the *CON* property for `saw` is clearly in NP. We have proved that it is NP-complete by reduction of the NP-complete Hamiltonian Cycle (HC) problem on a particular class of planar graphs, called *special planar graphs* in [8]. The complete proof is proposed in the Appendix.

A simple polynomial approximation of the `saw` constraint can be obtained by replacing the constraint with a number of binary constraints. The application of AC filtering on these binary constraints represents a polynomial approximation of GAC filtering for `saw`. Another simple polynomial filtering process can be obtained by iterating the `alldifferent` and `contiguous` GAC filtering.

Figure 2 compares the three types of filtering on a small example, in the case of a 2D version of the cubic lattice. The domains are shown on the left of the arrow (D_1 contains a single point, while $D_2, \ldots, D_{10}$ include 10 points). On the right, the figure shows the results of the different forms of filtering—where

1. all the circles (of any color) represent points left in the domain by AC filtering;
2. light grey (green) circles are points that are removed by the iterated GAC filtering of `alldifferent`+`contiguous`;
3. the white circles are the additional points removed by the GAC filtering of the `saw` constraint.

The total size of all the domains (initially equal to 91) is reduced to 38, 19, and 17 in the different approaches.

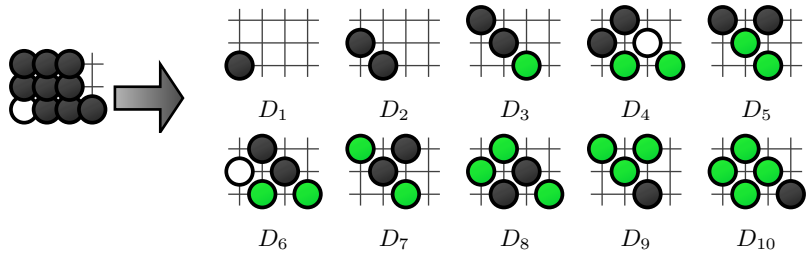


Fig. 2. Propagation based on AC (all points), iterated `alldifferent`+`continuous` GAC (black and white), and `saw` GAC (black).

3.4 The `alldistant` constraint

When we model biological problems on a discrete lattice, we often observe that the `alldifferent` global constraints is not sufficiently expressive. In particular, we often require that values assigned to a group of variables are sufficiently spread in the lattice, ensuring a minimal distance between each pair of points assigned to the variables. This is required, for example, to address the fact that different amino acids of a protein have different volume occupancy.

In the `alldistant` constraint, given n variables $X_1, \ldots, X_n$, with respective domains $D^{X_1}, \ldots, D^{X_n}$, and n numbers $c_1, \ldots, c_n$, we are looking for a solution $X_1 = p_1, \ldots, X_n = p_n$ such that, for each pair $1 \leq i, j \leq n$, we have that p_i and p_j are located at distance at least $c_i + c_j$. More formally:

$$\begin{aligned}\texttt{alldistant}(X_1, \ldots, X_n, c_1, \ldots, c_n) = (D^{X_1} \times \cdots \times D^{X_n}) \setminus \\ \{(a_1, \ldots, a_n) \in (D^{X_1} \times \cdots \times D^{X_n}) : \\ \exists i, j.\ 1 \leq i < j \leq n \wedge sqeucl(a_i, a_j) < (c_i + c_j)^2\}\end{aligned}$$

Note that if we consider the `alldistant` with $c_1 = \frac{1}{2}, \ldots, c_n = \frac{1}{2}$ then we achieve the same effect as `alldifferent`.

Figure 3 shows a simple example of application of GAC for `alldistant`. Let the domains of all the three variables be the set of grey points in the leftmost

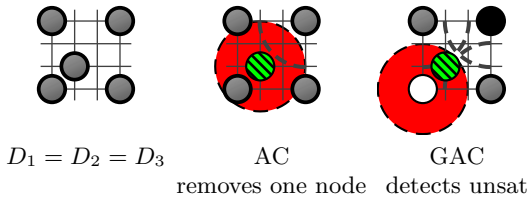


Fig. 3. Different effects of AC and GAC on $\texttt{alldistant}(X_1, X_2, X_3, 2, 2, 2)$

picture. In the center picture, the green (hatched) point is at distance less than $2+2$ from all other points. It is therefore removed by AC. For every other point, there is always a point at distance greater than 4 (the point at the opposite corner). Finally, consider the rightmost picture. If the white point is selected for D_1, only the black point is available in D_2. No points remain for D_3. Thus, the white point must be removed. The same will happen for the other points: GAC filtering detects unsatisfiability.

To prove the NP-completeness of the consistency problem, we developed a reduction of the BIN-Packing problem to the `alldistant` consistency problem (see the proof in the Appendix). NP completeness of GAC follows, as usual. The problem of filtering is open, and it could be investigated, e.g., through adaptation of the *sweep algorithms* used in [5]. This line of research is left for future work.

3.5 The `chain` constraint

The `chain` global constraint states that the n variables are a self-avoiding walk and, moreover, a certain distance between amino acids must be respected, *save for consecutive* variables that must be distant exactly one lattice point.

More formally, given n variables $X_1, \ldots, X_n$, with domains $D^{X_1}, \ldots, D^{X_n}$, and n numbers $c_1, \ldots, c_n$:

$$\begin{aligned}\texttt{chain}(X_1, \ldots, X_n, c_1, \ldots, c_n) = ((D^{X_1} \times \cdots \times D^{X_n}) \setminus \\ \{(a_1, \ldots, a_n) \in (D^{X_1} \times \cdots \times D^{X_n}) : \\ \exists i, j.\ 1 \leq i+1 < j \leq n \wedge sqeucl(a_i, a_j) < (c_i + c_j)^2\}) \setminus \\ \{(a_1, \ldots, a_n) \in (D^{X_1} \times \cdots \times D^{X_n}) : \\ \exists i\ 1 \leq i < n \wedge (a_i, a_{i+1}) \notin E\}\end{aligned}$$

Note that if we consider the `chain` with $c_1 = \frac{1}{2}, \ldots, c_n = \frac{1}{2}$ then we achieve the same effect as `saw`. Being therefore a generalization of `saw`, CON and GAC for `chain` are both NP-complete.

3.6 The `rigid block` constraint

It is a frequent situation, when dealing with protein structure determination, to have knowledge of local features of the structure, e.g., presence of secondary

structure components, such as α-helices and β-strands. Thus, we may wish to express the fact that a collection of points (e.g., amino acids) have to be located in the discrete lattice according to a predefined pattern (e.g., an helix).

This notion can be represented using another type of global constraint, called `rigid block` constraint. A rigid block defines a layout of points in the space that has to be respected by all admissible solutions. Let $X_1, \ldots, X_n$ be a list of variables (having, respectively, domains $D^{X_1}, \ldots, D^{X_n}$), and let $\boldsymbol{B} = B_1, \ldots, B_n$ be a list of lattice points—that, intuitively, describe the desired layout of the rigid block. $\mathsf{block}(X_1, \ldots, X_n, \boldsymbol{B})$ is a n-ary constraint, whose solutions are assignments of lattice points to the variables $X_1, \ldots, X_n$, that can be obtained from $\boldsymbol{B}$ modulo *translations* and *rotations*.

More precisely, we define a *rotation* of a lattice point $p = (p_x, p_y, p_z)$ as

$$rot(\phi, \theta, \psi)(p) = X \cdot Y \cdot Z \cdot p^T, \text{ where}$$

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \quad Y = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad Z = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Although the rotation angles ϕ, θ, ψ are real valued, only few combinations of them define automorphisms on the lattice in use. The total numbers of distinct automorphisms r depends on the lattice—e.g., in the cubic lattice, we have that $r = 16$, and in the `FCC` we have that $r = 24$.

We extend the definition of rotation to the case of lists of lattice points (denoted by $rot(\phi, \theta, \psi)(\boldsymbol{B})$), where $\boldsymbol{B}$ is a list of points and the result is a list in which every element of $\boldsymbol{B}$ is rotated according to the previous definition.

Given a list of points $\boldsymbol{B}$, we define the concept of *templates* as the set:

$$\mathsf{Templ}(\boldsymbol{B}) = \left\{ rot(\phi, \theta, \psi)(\boldsymbol{B}) \ : \ \begin{array}{c} \exists \phi, \theta, \psi.\ rot(\phi, \theta, \psi)(\boldsymbol{B}) \text{ is an} \\ \text{automorphism on the lattice} \end{array} \right\}$$

which contains the distinct 3-dimensional rotations of the points $\boldsymbol{B}$ in the lattice. Note that, for a given list of points ($\boldsymbol{B}$), the cardinality of $\mathsf{Templ}(\boldsymbol{B})$ is at most r. We say that $\boldsymbol{\ell} = (\ell_x, \ell_y, \ell_z)$ is a *lattice vector* if the translation by $\boldsymbol{\ell}$ of lattice points generates an automorphism on the lattice.

Let $\boldsymbol{\ell}$ be a lattice vector; with $\mathsf{Shift}[\boldsymbol{\ell}]$ we denote a mapping that translates a rigid block according to the vector ℓ. Formally, for each $i = 1, \ldots, k$, $\mathsf{Shift}[\boldsymbol{\ell}](\boldsymbol{B})[i] = B_i + \boldsymbol{\ell}$. Shifts are used to place a template into the lattice space, preserving the orientation and the distances between points.

A `rigid block` constraint $\mathsf{block}(X_1, \ldots, X_n, \boldsymbol{B})$ is then defined as the set:

$$\left\{ (a_1, \ldots, a_n) \in D_1 \times \ldots \times D_n \ : \ \exists \boldsymbol{\ell}\, \exists P. \begin{pmatrix} P \in \mathsf{Templ}(\boldsymbol{B}) & \wedge \\ \mathsf{Shift}[\boldsymbol{\ell}](P) = (a_1, \ldots, a_n) \end{pmatrix} \right\}$$

With a fixed rotation of the block, CON is linear in the size of the smallest variable domain (a simple intersection of possible translations for each domain has to be performed). GAC is polynomial as well, since it is sufficient to repeat the CON test for each domain.

Propagation of this kind of constraint is studied in a wider context in [19]. Moreover, the idea of considering rigid blocks to model substructures of proteins has also been introduced in [13].

4 Density Constraints

Electron cryomicroscopy is an experimental technique that allows structure determination for large and membrane proteins [27, 20, 7]. The basic idea is to retrieve several 2D scans of a high concentration protein solution, in which many copies of the same molecule are freely oriented in the fluid. The combination of these scans produces a *density map* from which it is possible to determine the 3D structure of large complexes (e.g., the Herpes virus or the identification of the secondary structure elements of proteins [27]). Although the resolution offered by this technique is not comparable to that offered by other techniques, it is considerably cheaper (w.r.t., e.g., NMR methods) and faster (w.r.t., e.g., crystallography techniques). Nonetheless, the information produced can be encoded in a global constraint that we present in this section.

4.1 Density maps

A density map is a data set obtained through electron cryomicroscopy analysis. It represents the electron density of molecules in a given portion of space and usually it has a resolution R ranging from 6Å to 12Å. The *resolution* R is the minimal distance that allows two distinct peaks of density to be distinguished. Common practice is to *sample* the data with a partitioning of the space into cubes with side of length S, such that $2S \leq R$. Even if $2S = R$ is sufficient to represent data with resolution R, it is useful to work with smaller S, typically $S = 1\mathring{A}$. From the information theory point of view this choice carries no extra information on the density, however it produces a finer discrete map that allows to place the corresponding amino acids with higher precision (placements every S in space on each coordinate instead of $R/2$).

The density map D can be seen as a function that associates a *density value* $D(x, y, z) \geq 0$ to each cube at position (x, y, z), which represents the sample measurement.

In Figure 4, on the right, we show a simple density map of a protein with 4 identical amino acids, arranged on the xy plane. We plot only the maximal z layer. The map is simulated with a resolution of 10Å and sampled at 1Å. The darker colors indicate higher densities. The white circles highlight the position of the center of the individual amino acids. In the same figure, on the left, we show the contribution of a single amino acid.

Each molecule component provides a typical contribution to the density map. This contribution can be approximated by a specific *contribution function* $\mathcal{F} : \mathbb{N}^3 \times \mathbb{N}^3 \rightarrow \mathbb{R}^+ \cup \{0\}$. For example, the simplest choice could be to use a *Gaussian* contribution:

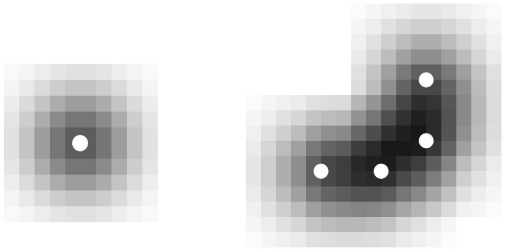


Fig. 4. Density map of an amino acid (left) and of a 4-amino acid protein (right)

$$\mathcal{F}(\boldsymbol{x}, \boldsymbol{p}) = \mathcal{G}_{a,\sigma}(\boldsymbol{x}, \boldsymbol{p}) = ae^{-\frac{|\boldsymbol{x}-\boldsymbol{p}|^2}{2\sigma^2}}$$

where $\boldsymbol{p} \in \mathbb{N}^3, a \in \mathbb{R}^+, \sigma \in \mathbb{R}^+$ are respectively the reference point for the center of the object, the intensity of the map, and the decay control parameter. The parameters a and σ can be estimated according to the type of the component, by first generating density maps for the single components and then performing a least square approximation. Let us observe that the Gaussian is only one of the possible forms for the contribution functions: our approach is parametric w.r.t. the function used.

The chemical description of a molecule—i.e., the set of components and their chemical bonds—allows us to decompose it into n components of interest. For example, a protein can be decomposed into the set of composing amino acids, or, with higher precision, into the set of atoms composing it. Each component $i \in \{1, \ldots, n\}$ can be placed in the space in the position $\boldsymbol{p}_i$, and it provides a specific contribution function $\mathcal{F}_i(\boldsymbol{x}, \boldsymbol{p}_i)$, which can be pre-computed.

Given a density map and a molecule, the ultimate goal is to find the possible placements $[\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n]$ of the components, so that their combination produces a density map close to D, namely for each $\boldsymbol{x} \in \mathbb{N}^3$

$$\sum_{i=1}^{n} \mathcal{F}_i(\boldsymbol{x}, \boldsymbol{p}_i) \approx D(\boldsymbol{x})$$

where $\approx$ means that we could introduce some tolerance in how closely we approximate D, due to the errors contained in the experimental data and introduced by approximations of $\mathcal{F}$ functions.

In Figure 4, for example, given the density map and the approximation of the single amino acid, we would retrieve the four placements (white circles) of the amino acids in the original protein.

In order to produce the contribution functions for a molecule, once its chemical formula is known, it is possible to derive exactly the electron charge of the molecule, and thus the expected amount of density in the map. This information provides the reference to the experimental data and allows us to relate correctly the single contributions to the density map.

4.2 The `density` constraint

We formalize the constraint induced by the knowledge of a density map associated to a molecule. This global constraint relies on the idea of discretization of the space into regions (e.g., cubes). These regions can be large enough to contain more than one chemical component.[2]

Assume the density map D is known. Let us consider the following items:

- k disjoint regions, numbered $i = 1, \ldots, k$, each described by:
  - the set of points $R_i \subseteq \mathbb{N}^3$ of the region. Let $R = (R_1, \ldots, R_k)$.
  - a (density) value $d_i = \sum_{(x,y,z)\in R_i} D(x, y, z) \in \mathbb{R}^+ \cup \{0\}$.
- n components, numbered $1, \ldots, n$, each of them characterized by the corresponding contribution function $\mathcal{F}_i$, where $\mathcal{F} = (\mathcal{F}_1, \ldots, \mathcal{F}_n)$,
- n variables $\boldsymbol{X} = (X_1, \ldots, X_n)$ with domains $O_1, \ldots, O_n$, where $O_i \subseteq R_1 \cup \cdots \cup R_k$.

The global constraint $\texttt{density}(\boldsymbol{X}, \mathcal{F}, R, D)$ (*density constraint*) is satisfied by all the n-tuples $\langle p_1, \ldots, p_n\rangle \in O_1 \times \cdots \times O_n$ such that for all $i = 1, \ldots, k$ it holds that:

$$\sum_{j=1}^{n} \sum_{\boldsymbol{x}\in R_i} \mathcal{F}_i(\boldsymbol{x}, p_j) \leq d_i + \varepsilon \tag{1}$$

with $\varepsilon \in \mathbb{R}^+ \cup \{0\}$. The constraint states that each region i provides an upper bound d_i to the sum of the density contributions provided by all the components in that area.

4.3 Complexity Results

We will study the complexity of the consistency problem for slightly simplified versions of the density constraints. For the sake of simplicity, our analysis is performed in the 2D space. We define the `Density 1` problem as follows:
Input:

1. k disjoint regions $1, \ldots, k$, each of them characterized by: the set of points R_i of the region i and a (density) value $d_i \in \mathbb{R}^+ \cup \{0\}$,
2. n components providing density contributions $a_1, \ldots, a_n \in \mathbb{N}$.

Question: Establish whether there is an assignment $\sigma : \{X_1, \ldots, X_n\} \to \mathbb{N}^2$ such that for all $i = 1, \ldots, n$:

$$\sigma(X_i) \in \bigcup_{j=1}^{k} R_j \qquad \sum_{j=1\ldots n, \sigma(X_j)\in R_i} a_j \leq d_i \tag{2}$$

[2] In the preliminary paper [11] we proposed two global constraints called *average density constraint* and *punctual density constraint*. The one considered here is more general of both of them.

Proposition 1. *The* `Density 1` *problem is NP complete.*

The proof (see the Appendix) is a reduction from bin packing. In this proof, we assume the possibility of arbitrary density contributions for the components. However, in a real problem at hand, density contributions come from a specific finite set of values, depending on the nature of the individual components. The following version of the density problem, in which the sets of component values are known in advance (let α be its cardinality), and a maximum number of components per region is fixed (bounded by a parameter β), admits a polynomial consistency check. Let $\alpha \in \mathbb{N}$ and $\beta \in \mathbb{N}$ be fixed. Let the `Density 2` problem be defined as `Density 1` with the further requirement:

3. The density contributions $a_1, \ldots, a_n$ are chosen from a set of α different elements and they are such that for all $i = 1, \ldots, n$ it holds that $a_i\beta \geq \max\{d_1, \ldots, d_k\}$

Then it holds that:

Proposition 2. *The* `Density 2` *problem can be solved in polynomial time.*

The proof is reported in the Appendix. However, this result is only of theoretical interest. The complexity is $O(n^q)$ but, in practice, the exponent q is rather large. For instance, assuming $\alpha = 20$ (one characteristic value for each amino acid) and $\beta = 5$ (at most 5 amino acids can be in a region), we obtain $q = 637,560,000$.

4.4 Preliminary experimental results

We can include density constraints in the context of a constraint solving framework used to determine the placement of amino acids in $\mathbb{N}^3$. Given a protein sequence $S \in \{1, \ldots, 20\}^n$, its density map D, the j-th amino acids density map parameters a_j and a Gaussian function $\mathcal{G}_j$, $j \in \{1, \ldots, 20\}$,[3] we define the variables X_i with domains $D_i \subseteq \mathbb{N}^3$, with $i \in \{1, \ldots, n\}$. The space is assumed to be regularly discretized, according to the structure of the cubic lattice, with R be the set of cubic regions considered. The constraints added to the problem are:

- `density`$(\boldsymbol{X}, \mathcal{G}, R, D)$
- `contiguous`$(\boldsymbol{X})$ which imposes that each pair of consecutive amino acids is placed at Euclidean distance equal to 3.8Å.

The CSP defined above in general does not admit solutions; this is due to two main reasons. The first one is that, given the discrete domains of amino acids positions, it is not possible to find a pair of points at exact distance equal to 3.8Å. The distance constraint `contiguous` has to be, in fact, relaxed, in order to allow admissible solutions. In the implementation, we used the range 3.3Å–5.2Å as acceptable distances.

[3] I.e., we have individual parameters a and σ for each one of the 20 distinct amino acids.

The second reason is more subtle: given the approximations made to model the contribution functions as Gaussian distributions and the discretized locations of contributions, it is impossible to recreate exactly the original density map D. This translates to the fact that the best solution, in which the components are mapped as close as possible to the original positions, provides fluctuations w.r.t. the original density map. The slight local excess of density would immediately falsify the `density` constraint. It is convenient, thus, to increase the original density map by adding a density threshold, obtained, e.g., as a fraction of a single amino acid maximal punctual contribution.

The combination of the two constraints allows a filtering that can be performed as follows. A domain point $\boldsymbol{p}$ for amino acid S_i has support only if the presence of S_i does not violate D. Moreover there must be a compatible assignment of S_{i-1} and S_{i+1} that respects both the `contiguous` and the `density` constraints.

It is possible to further enhance the filtering, considering that, for each supported assignment of S_i, if there is a point $\boldsymbol{t}$ such that $D(\boldsymbol{t})$ is greater than the sum of contributions of S_{i-1}, S_i, and S_{i+1} in $\boldsymbol{t}$, then it will be possible to add another generic amino acid that contributes to $\boldsymbol{t}$, in order to reduce the gap. This boils down to finding a support for this amino acid.

To show the impact of the density constraint, we focus on simple examples in the 2D plane and on a small peptide in the 3D space. The prototype is implemented on an AMD Opteron 2.2GHz Linux machine.

Examples in the 2D space

In the first example, we arrange a chain of 30 amino acids with a spiral shape, respecting the distance of 3.8Å between consecutive elements. This example shows that non-trivial arrangements can be reconstructed, despite the fact that the domain of admissible positions is discrete.

In Figure 5, we show the input density map (on the left) and the computed density map (in the center) generated by an admissible solution (on the right). The darker pixels represent the denser regions. The side of a pixel is 1Å and the simulated resolution is 10Å.

Fig. 5. Spiral with 30 amino acids: density (left), predicted density (center) and placement (right)

In the second example, we show how a uniform density map (obtained by a square tiling arrangement of 16 amino acids on the plane) can be reconstructed. The density maps and resulting configuration are shown in Figure 6 (we report

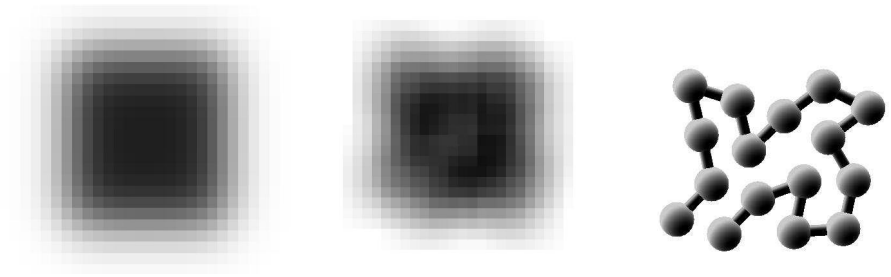


Fig. 6. Square with 16 amino acids: density (left), predicted density (center) and placement (right)

the first admissible arrangement found).

The results are summarized in Table 2. The Filtering column shows the time required for the initial pruning of domains before the search is started. In the spiral example, the filtering allows us to significantly reduce the size of initial domains for central variables, while for side amino acids the domain covers the whole spiral. The Domain size column reports the size of the domains of the variables involved. Observe that after the filtering only 16 points are allowed to the first and last variables of the spiral. Central variables, instead, have 387 points allowed. The presence of small domains allows us to exploit a first fail strategy that is able to find a solution after few backtracks.

	Amino acids	Filtering	Domain size	Search	Nodes
Spiral	30	119.1 s	16–387	0.13 s	103
Square	16	23.8 s	214	95.6 s	124,663

Table 2. Summary of the experimental results for examples 1 and 2

Observe that, in the square example, there are many equivalent solutions, roughly bounded by the number of possible self avoiding walks inside the square.

Examples in 3D space

Let us consider a more realistic example, using data from a real protein. We considered a small beta hairpin (PDB ID: 1LE0, model 1), made of 12 amino acids. We simulate its density map using a discretization of 1Å and a resolution of

8Å. The contiguous range is set to 3.2—5.0Å. Moreover, we consider a threshold of tolerance between simulation and reference, of 0.65 times the highest density value found in the density map of Glycine (the smallest amino acid). The search, launched without any particular heuristic and/or optimization, finds a solution in 15 minutes.

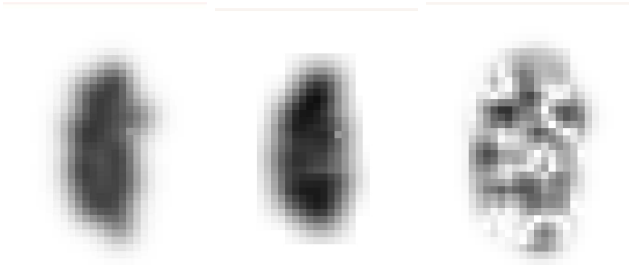


Fig. 7. Original density (left), predicted density (center) and 4x error (right)

In Figure 7, we show a central z layer of the density map. On the left, we show the simulated density map, while in the center we show the sum of the contributions of the amino acids in the solution. The darker pixel represents a denser region. The picture has been plotted using the same scale. It can be noticed than the computed placement tends to give a more compact result, since the side chains are approximated by a simple Gaussian sphere. On the right, we show the error between the two maps, enhanced by 4 times w.r.t. the other plots scale, to make it more visible.

In Figure 8, we show the 3D plot of the original backbone of 1LE0 (on the left) and the backbone detected in the solution (on the right). It is interesting to note that the beta shape is well preserved. However, there is a shift in the position of the first/last amino acids. Basically, the start of the protein is misplaced, due to the ambiguity in the density map and to the current complete flexibility of bend angles between amino acids.

Fig. 8. Original protein (left) and simulated protein(right)

We would like to stress that the space searched depends exponentially on the number of neighbors, once an amino acid is fixed. In these examples, there are 52 neighbors that respect the `contiguous` constraint (recall that we use a range of allowed distances). The presence of the `density` constraint allows us to deal with a 52^n search space working with 1Å accuracy. This is extremely desirable in order to obtain sound results. In systems like COLA [12], the accuracy was set at 3.8Å with only 12 neighbors per point (due to the Face Centered Cubic lattice used). We wish to find a good compromise between these two different discretizations.

These results, although preliminary, provide a good indication of the potential of the density constraints. For example, the knowledge of known rigid blocks (e.g., helices, beta strands) could sensibly constrain the search space allowing faster and more realistic predictions.

5 Conclusions and future work

In this paper, we presented an analysis of various global constraints designed to provide declarative encoding of problems on discrete crystal lattices. Discrete crystal lattices have been extensively used to provide discretizations of the 3D space that are amenable of efficient computation for modeling approximated solutions to the protein structure determination problem.

Although several researchers have identified constraint solving as an effective paradigm to produce approximated solutions of protein conformations in discrete crystal lattices, the majority of the literature has ignored the problem of investigating what kind of global constraints are relevant and how efficiently these can be computed. This paper offers the first formal study of global constraints in discrete lattices for protein structure determination.

The introduction of global constraints is motivated by the need of a specific formalization and efficient solving techniques of the protein structure determination problem. The different global constraints considered are motivated by different aspects of modeling a protein conformation. For each constraint, we discussed its computational complexity, providing novel complexity results, that are essential to understand the cost of the different constraints and to identify the critical computational aspects in handling these constraints in a concrete constraint framework.

The future work is focused on the development of effective implementations of these global constraints within the COLA framework. Some preliminary experiments have been conducted with encouraging results. The inherent complexity of the problem at hand, demonstrated by the high computational complexity of the global constraints, requires the introduction of approximation algorithms, especially in the computation of the propagation algorithms. Effective data structures to facilitate the task of filtering are also being explored. In particular, the `density` constraint could be implemented making use of *oct-tree* data structures to efficiently represent the density maps and to speed up constraint propagation.

Another important aspect of future work is the investigation of the interaction between different global constraints. We conjecture that selected sets of global constraints can effectively help each others in performing a more dramatic pruning of the search space. For example, the integration of density information and other global constraints (e.g., `rigid block`, `saw`) is expected to provide good results.

We also plan to define a space model that can accommodate the differences between the typical density discretization (cubic lattice with 1Å side) and the statistical free energy model, better encoded by an `FCC` lattice with 3.8Å lattice unit.

Acknowledgements The research has been partially supported by the FIRB Project RBNE03B8KK, by the PRIN Project 2005015491, and by NSF grants CNS-0220590 and HRD-0420407.

References

1. R. Agarwala, S. Batzoglou, V. Dancik, S. Decatur, S. Hannenhalli, M. Farach, S. Muthukrishnan, S. Skiena. (1997) 'Local rules for protein folding on a triangular lattice and generalized hydrophobicity in the HP model', *J. of Computational Biology*, 4:275–296.
2. R. Backofen. (2001) 'The protein structure prediction problem: A constraint optimization approach using a new lower bound', *Constraints*, 6:223–255.
3. R. Backofen and S. Will. (2003) 'A Constraint-Based Approach to Structure Prediction for Simplified Protein Models that Outperforms Other Existing Methods', *International Conference on Logic Programming*, Springer Verlag.
4. R. Backofen and S. Will. (2006) 'A Constraint-Based Approach to Fast and Exact Structure Prediction in Three-Dimensional Protein Models', *Constraints*, 11(1):5–30.
5. N. Beldiceanu and M. Carlsson. (2001) 'Sweep as a Generic Pruning Technique Applied to the Non-overlapping Rectangles Constraint', *Principles and Practice of Constraint Programming*, Springer Verlag, pp. 377–391.
6. C. Bessiere, E. Hebrard, B. Hnich, T. Walsh. (2004) 'The complexity of global constraints', *National Conference on Artificial Intelligence*, AAAI Press.
7. B. Böttcher, S.A. Wynne, and R.A. Crowther. (1997) 'Determination of the Fold of the Core Protein of Hepatitis B Virus by Electron Cryomicroscopy', *Nature*, 386:88–91.
8. P. Crescenzi, D. Goldman, C. Papadimitriou, A. Piccolboni, M. Yannakakis. (1998) 'On the Complexity of Protein Folding', *Journal of Computational Biology*, 5(3):423–466.
9. A. Dal Palù, A. Dovier, and F. Fogolari. (2004) 'Constraint logic programming approach to protein structure prediction', *BMC Bioinformatics*, 5(186).
10. A. Dal Palù, A. Dovier, and E. Pontelli. (2005) 'A Constraint Logic Programming Approach to 3D Structure Determination of Large Protein Complexes', *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, Springer Verlag, pp. 48–63.
11. A. Dal Palù, A. Dovier, E. Pontelli. (2007) 'The density constraint', *Workshop on Constraint-based Methods for Bioinformatics*, Porto, pp. 10–19.

12. A. Dal Palù, A. Dovier, and E. Pontelli. (2007) 'A constraint solver for discrete lattices, its parallelization, and application to protein structure prediction', *Software Practice and Experience*, DOI: 10.1002/spe.810.
13. A. Dal Palù, S. Will, R. Backofen, and A. Dovier. (2004) 'Constraint Based Protein Structure Prediction Exploiting Secondary Structure Information', *Proc. of Italian Conference on Computational Logic (CILC)*.
14. R. Dechter. (2003) *'Constraint Processing'*, Morgan Kaufmann.
15. E.C. Freuder. (1982) 'A sufficient condition for backtrack-bounded search', *Journal of the ACM*, 29(1)24–32.
16. W.E. Hart and A. Newman. (2003) 'The Computational Complexity of Protein Structure Prediction in Simple Lattice Models', *Handbook on Algorithms in Bioinformatics*, CRC Press.
17. L. Krippahl and P. Barahona. (1999) 'Applying Constraint Propagation to Protein Structure Determination', *Principles and Practice of Constraint Programming*, Springer Verlag, pp. 289–302.
18. L. Krippahl and P. Barahona. (2002) 'PSICO: Solving Protein Structures with Constraint Programming and Optimisation', *Constraints*, 7:317–331.
19. L. Krippahl and P. Barahona. (2003) 'Propagating N-Ary Rigid-Body Constraints', *Princ. & Practice Constraint Progr.*, Springer Verlag, pp. 452–465.
20. E.J. Mancini, M. Clarke, B.E. Gowen, T. Rutten, and S.D. Fuller. 'Cryo-electron Microscopy Reveals the Functional Organization of an Enveloped Virus.' *Mol. Cell*, 5:255266, 2000.
21. C.H. Papadimitriou. (1994) *'Computational Complexity'*, Addison-Wesley.
22. G. Raghunathan and R.L. Jernigan. (1997) 'Ideal architecture of residue packing and its observation in protein structures', *Protein Science*, 6:2072–2083.
23. S. Raman, B. Qian, D. Baker, R.C. Walker. (2008) 'Advances in Rosetta protein structure prediction on massively parallel systems', *IBM Journal of Research and Development*, 52(1–2).
24. J.-C. Regin. (1993) 'A filtering algorithm for constraints of difference in CSPs', *R.R. LIRMM*, 93–068.
25. J. Skolnick and A. Kolinski. (2004) 'Reduced models of proteins and their applications', *Polymer*, 45:511–524.
26. W.J. van Hoeve. (2001) 'The Alldifferent Constraint: a Survey', *Sixth ERCIM Workshop on Constraints*.
27. Z.H. Zhou, M. Dougherty, J. Jakana, J. He, F. Rixon, and W. Chiu. (2000) 'Seeing the Herpesvirus Capsit at 8.5Å', *Science*, 288:877–880.

APPENDIX: COMPLEXITY PROOFS

Polinomiality of `contiguous`

The `contiguous` constraint, defined as follows:

$$\texttt{contiguous}(X_1,\ldots,X_n) = (D^{X_1}\times\cdots\times D^{X_n}) \setminus \\ \big\{(a_1,\ldots,a_n)\in(D^{X_1}\times\cdots\times D^{X_n}) : \exists i.\ (1\leq i<n\ \wedge\ (a_i,a_{i+1})\notin E)\big\}$$

is equivalent to the conjunction of the $n-1$ binary constraints of the form $C_{i,i+1}$, with $i\in\{1,\ldots,n-1\}$, such that

$$C_{i,i+1} = (D^{X_i}\times D^{X_{i+1}})\setminus\{(a_i,a_{i+1}) : a_i\in D^{X_i}\wedge a_{i+1}\in D^{X_{i+1}}\wedge(a_i,a_{i+1})\notin E\}$$

The constraint propagation, as well as the CON tests, can be executed in polynomial time. This result can also be justified as follows. Let us assume that for every $i\in\{1,\ldots,n-1\}$, $C_{i,i+1}$ is arc consistent. Let us choose $i\in\{1,\ldots,n\}$ and $a_i\in D^{X_i}$.

- Since $C_{i-1,i}$ is AC, then there is $a_{i-1}\in D^{X_{i-1}}$ such that $(a_{i-1},a_i)\in C_{i-1,i}$. The process can be recursively repeated until $C_{1,2}$ is reached.
- Since $C_{i,i+1}$ is AC, then there exists $a_{i+1}\in D^{X_{i+1}}$ such that $(a_i,a_{i+1})\in C_{i,i+1}$. The same process can be recursively repeated until $C_{n-1,n}$ is reached.

Thanks to this backward and forward process, we can collect a set of elements such that $(a_1,\ldots,a_i,\ldots,a_n)\in C$, thus proving that $\texttt{contiguous}(X_1,\ldots,X_n)$ is GAC.

Observe also that, if there exists $C_{i,i+1}$ that is not AC, then C will not be GAC. In fact, that would imply that there is $a_i\in D^{X_i}$ s.t. $\forall b_{i+1}\in D^{X_{i+1}}$ we have that $(a_i,b_{i+1})\notin E$. This means, in particular, that for all $b_1\in D^{X_1}$, ..., $b_{i-1}\in D^{X_{i-1}}$, $b_{i+1}\in D^{X_{i+1}}$, ..., $b_n\in D^{X_n}$, we have that $(b_1,\ldots,b_{i-1},a_i,b_{i+1},\ldots,b_n)\notin C$.

NP-completeness of `saw`

We prove the NP-completeness of the consistency problem for `saw` constraints by reduction of the NP-complete Hamiltonian Cycle (HC) problem on a particular class of planar graphs, called *special planar graphs* in [8]. The proof consists of two steps. First we show how to embed a special planar graph G in a graph G' whose nodes and edges are in a cubic lattice. G' is then "enlarged" (replacing each edge with two connected edges with a new intermediate node) obtaining a new graph G'', that we use to define variables+domains for an equivalent `saw` problem.

A special planar graph $G=(N,E)$ [8] is composed of a number of *loops* (e.g., the four loops 1, 2, 3, 4 in Figure 9), each containing only nodes with degree 3 (e.g., the various nodes A,B,C,D in each loop), along with *paths* of length 2 connecting nodes that belong to distinct loops. Let $n=|N|$. Observe that loops have to contain at least three nodes (otherwise you would have two edges

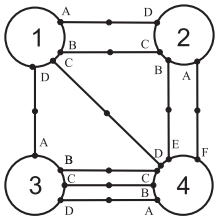


Fig. 9. An example of special planar graph G

connecting two nodes). Moreover, it is easy to see that if there is a loop in the graph containing an odd number of nodes, then no HC exists for G. For this reason, we concentrate on graphs containing only loops with even (with at least 4) numbers of nodes. We assume, moreover, that G contains at least 2 loops (otherwise the HC problem is trivially true).

Let us define how to obtain G' from G. For each loop i, of size $2n_i$, and consisting of the nodes $p_1, \ldots, p_{2n_i}$, we generate a subgraph of G', called *gadget*, which contains $2n_i \times (n_i+2)$ nodes arranged as follows. These nodes are obtained using a clockwise enumeration of the loops—the starting point is irrelevant. There is a core of the gadget made of a loop of $2n_i$ nodes, arranged as a $2 \times n_i$ rectangle. ¿From each of the core nodes there is a path leading to the nodes $p_1, \ldots, p_{2n_i}$. Those nodes are called the *output* nodes of the gadget. In Figure 10 we report a gadget for a 8-nodes loop.

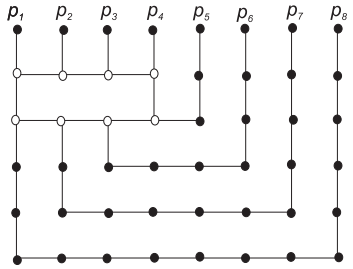


Fig. 10. Example of gadget for a loop of size 8

Let us fix one of the dimensions of the cube (w.l.o.g., $z = 0$) and let us work on the resulting 2-dimensional plane. Let us consider an arbitrary enumeration of the loops $\ell_1, \ldots, \ell_k$ of G and let us align the gadgets on the plane according to such ordering. In particular, all the output nodes of the gadget have $y = 0$,

and they are adjacent in the x dimension (see Figure 11). All other nodes of the gadget have $y > 0$.

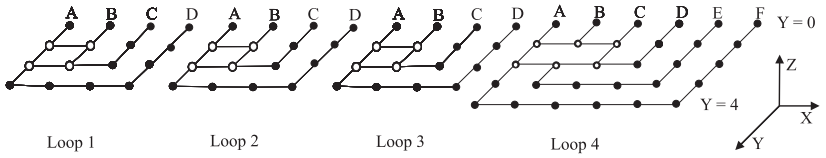


Fig. 11. The aligned gadgets for graph G in Figure 9

Consider, in lexicographical ordering, the loop pairs $\langle \ell_a, \ell_b \rangle$, $a < b$, that are connected by edges in E. We wish to create copies of the output nodes of ℓ_a and ℓ_b in a separate plane (to avoid intersection of edges) and recreate on this plane the connection structure of G.

Let us illustrate the process for all the pairs $\langle \ell_1, \ell_{b_1} \rangle$, $\ldots$,$\langle \ell_1, \ell_{b_h} \rangle$ such that there are edges between loop 1 and loop b_i in G.

In the plane i, we add copies of the "relevant" output nodes for loop 1 and b_i. E.g., if an output node of loop 1 or loop b_i is at coordinates $(x, y, 0)$, and such node is part of an edge connecting these two loops, then a copy of such node will be created at coordinates (x, y, i). Furthermore, copies of such nodes are also placed in all the intermediate planes (planes $0 \leq z \leq i - 1$), and edges connecting these copies are created—i.e., edges of the type $((x, y, z), (x, y, z+1))$. The edges between the nodes of distinct loops of G are simulated by paths in the plane i. The output nodes of the gadgets 1 and b_i respect a clockwise traversal of the loops 1 and b_i. Since G is planar, we can connect using non-intersecting paths in plane i (see Figure 12). As a strategy, start with the rightmost output node of loop 1.

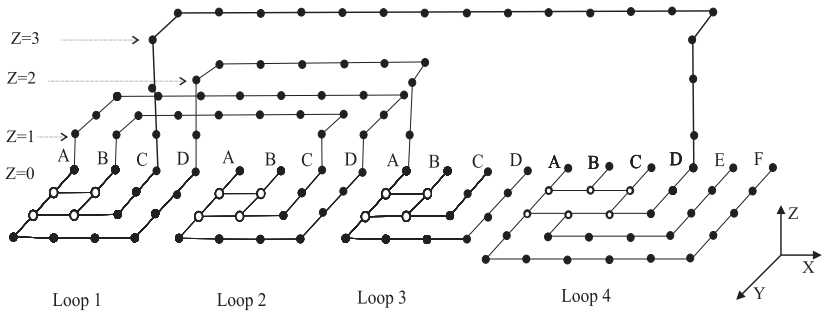


Fig. 12. The encoding of the edges outgoing from loop 1 in Figure 9

The process is repeated for all the other pairs of connected loops (incrementing the plane levels). In Figure 13 we show the graph G' corresponding to the graph G of Figure 9.

A rough estimate of the size of the resulting graph is the following. The number of x indices used is $O(n)$ (gadgets outputs are the same as loop lengths). The number of values of y used is $O(n)$ for the gadgets plus $O(|E|) = O(n^2)$. The number of values of z used is again $O(|E|) = O(n^2)$. Thus the global "box" containing G' contains $O(n^5)$ points.

Since the graph G' maintains the topology (and at most introduces new nodes with degree 2 on paths), it holds that the Hamiltonian Cycle problem on G' has a solution iff G has. Basically, G' is a copy of G where the edges linking distinct loops are stretched (by adding new nodes of degree 2). We will refer these sequences of edges as *loop2loop*. loop2loops have always length at most 4 (out of the gadgets) plus 2 (within the gadgets). We will use the same terminology in G''.

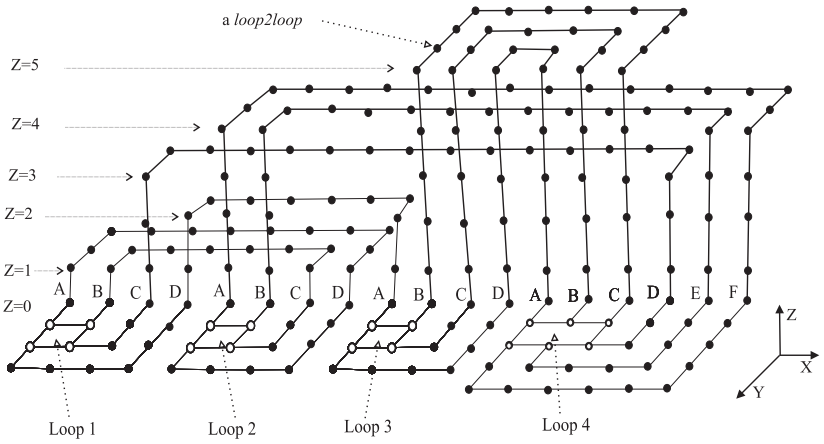


Fig. 13. The graph G' obtained from G in Figure 9

We need an additional step to encode the HC problem using the `saw` constraint. The basic idea is that a self-avoiding walk is an Hamiltonian Path. The problem is that in G' representatives of elements of N may lie at distance 1 in spite of them not being connected by an edge in E. If we introduce variables and assign to their domains the nodes of N', self-avoiding walks on the nodes of N' can have trajectories that do not exist in G.
We define the graph $G'' = (N'', E'')$ as follows:

- for each edge $((x, y, z), (x+1, y, z)) \in N'$, N'' contains the nodes $a = (2x, 2y, 2z)$, $b = (2x + 1, 2y, 2z), c = (2x + 2, 2y, 2z)$, and the edges (a, b) and (b, c).

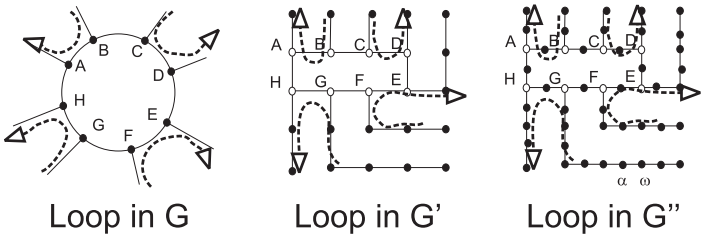


Fig. 14. Loops in G, G', and G''. Observe that the Hamiltonian paths in G and G' cannot touch half of the extra nodes added in loops in G''

- for each edge $((x, y, z), (x, y + 1, z))$, N'' contains the nodes $a = (2x, 2y, 2z)$, $b = (2x, 2y + 1, 2z), c = (2x, 2y + 2, 2z)$, and the edges (a, b) and (b, c).
- for each edge $((x, y, z), (x, y, z + 1))$, N'' contains the nodes $a = (2x, 2y, 2z)$, $b = (2x, 2y, 2z + 1), c = (2x, 2y + 2, 2z + 2)$, and the edges $(a, b), (b, c)$.

The graph G'' is a copy of G', in which each edge is substituted by a subgraph of the form (edge, new node, edge). Let $m = |N''|$. In Figure 14, on the right it is depicted a fragment of G'' obtained from the fragment of G' in the center of the figure. Observe that edges in E'' connect nodes at Euclidean distance 1. This property can be exploited by `saw` to simulate the graph connectivity.

Consider again Figure 14 from left to right. Let us assume that G admits an Hamiltonian Cycle. Any Hamiltonian Cycle traverses the loop in a way similar to the one depicted. There is a corresponding Hamiltonian Cycle traversing the loop in G'. A corresponding path exists in G''; however it is not Hamiltonian since *half of the nodes of the loop* cannot be traversed by that path. We must take care of that designing our encoding.

Let L be the global number of nodes in loops of G divided by 2. Define the variables $X_1 \dots X_{m-L}$, and for $1 < i < 2m$ let $D^{X_i} = N''$. For the variables X_1 and X_{m-L} we specify a singleton domain as follows. Identify in G'' two consecutive nodes of degree 2 in a loop2loop. Let us call them α and ζ (see also Figure 14—right). Then $D^{X_1} = \{\alpha\}, D^{X_{m-L}} = \{\zeta\}$. The definition of the CSP is completed by the constraint $\texttt{saw}(X_1, \dots, X_{m-L})$.

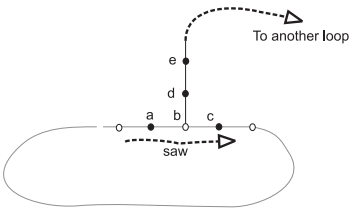
Theorem 1. *G' admits an Hamiltonian Cycle iff G'' admits a self avoiding walk with $m - L$ nodes starting from α and ending in ζ.*

Proof. ($\rightarrow$) Let us assume that G' has an HC. The same cycle can be mimicked on the extended graph G''. All nodes in loop2loops are traversed by this path. Instead, for each loop, a number of points which is half of the number of points of the original loop in G is not traversed by the path. Then, the cycle has length $m - L$.

Since α and ζ have degree 2 and are in a loop2loop, the cycle must contain the edge (α, ζ). Removing such edge from the path we obtain a SAW of length $m - L$ starting in α and ending in ζ.

($\leftarrow$) Let $\alpha, p_2, \ldots, p_{L-m-1}, \zeta$ be a SAW consisting of $m-L$ nodes of G'' starting from α and ending in ζ. Since $|N''| = m$, exactly L nodes of N'' are left out by this SAW.

1. If the SAW enters and exists all the loops as in Figure 14, then it will leave out L nodes and it corresponds to an Hamiltonian Path in G' starting in α and ending in ζ. Since α and ζ are consecutive nodes of degree 2, it is sufficient to add the edge (ζ, α) to find the cycle which corresponds to an Hamiltonian Cycle in G.
2. Since α and ζ are in the same loop2loop it is impossible that a SAW starting in α and ending in ζ does not enter any loops, unless $m - L = 1$, which cannot be true by construction of G''.
3. It remains to analyze the case in which the SAW traverses a loop in a way different from that of point 1. Assuming that it exists, we will find a contradiction with its length $m-L$. For these SAWs, there is at least one loop2loop left out from a path traversing one loop, as in the following figure:



We have already seen that loops2loops are of length ≥ 6 in G' (13 in G''). Let e and d be the first two nodes of the loop2loop left out. These two nodes cannot be crossed by the SAW. As a matter of fact, if d or e are reached by a SAW, there is no way to come back to ζ without repeatedly visiting the same nodes.
On the other hand, the SAW inside the loop visits both the points a and c adjacent to the entering point b of the analyzed loop2loop. With respect to a SAW of the form dealt with in point 1, the SAW visits one additional point in the loop but looses two points outside the loop. This happens for every loop and for every loop2loop excluded by the SAW. Thus, more than L points of G'' are left out. This is a contradiction. □

This reduction is polynomial, thus the CON of `saw` global constraint is NP-complete, and, consequently, GAC is NP-hard. Observe that the proof has been carried out using the cubic lattice. It is easy to modify the mapping for other 3D lattices.

NP-completeness of `alldistant`

We show how to reduce the BIN-Packing problem to the consistency problem for the `alldistant` constraint. Let us consider n items of size $c_1, \ldots, c_n$ and k bins (bin $0, \ldots,$ bin $k-1$) of capacity B. W.l.o.g., let us assume that, for all $i \in \{1, \ldots, n\}$, it holds that $c_i \leq B$ (otherwise the problem is trivially unsatisfiable).

We reduce the problem using only one dimension of the lattice (assume, e.g., that all y and z coordinates are fixed to 0). We consider consecutive lattice collinear points. For the sake of simplicity, we consider lattice points $(0,0,0)$, $(1,0,0)$,$(2,0,0)$,... and we refer to them simply as $0, 1, 2, \ldots$[4]

The reduction is defined as follows. Let us introduce n lattice variables $X_1, \ldots, X_n$. For $i \in \{1, \ldots, n\}$ the domain D^{X_i} is defined as

$$D^{X_i} = \bigcup_{j=0}^{k-1} [4jB + c_i \,..\, 4jB + 2B - c_i]$$

For example, consider the instance: $c_1 = 4, c_2 = 3, c_3 = 5, c_4 = 1, B = 7, k = 2$. Then $D_1 = [4\,..\,10] \cup [32\,..\,38]$, $D_2 = [3\,..\,11] \cup [31\,..\,39]$, $D_3 = [5\,..\,9] \cup [33\,..\,37]$, $D_4 = [1\,..\,13] \cup [29\,..\,41]$.

Intuitively, each interval $[0\,..\,2B]$, $[4B\,..\,6B]$, $[8B\,..\,10B]$, ... corresponds to a bin. Each assignment of the variable X_i in D_i is such that all values $[X_i - c_i, X_i + c_i]$ are included in exactly one of the above intervals. The item i is assigned to the bin corresponding to such interval. If the values of X_i and X_j are in two different intervals, then $|X_i - X_j| > 2B \geq c_i + c_j$.

We show that there is a solution for the instance of the BIN-packing problem if and only if there is a solution for the CSP $\texttt{alldistant}(X_1, \ldots, X_n, c_1, \ldots, c_n)$. In the above example, a solution of the CSP is $X_1 = 4$, $X_2 = 11$, $X_3 = 33$, $X_4 = 40$, from which one can conclude that we should place items 1 and 2 in bin 0 and items 3 and 4 in bin 1.

For one direction, assume that the CSP admits a solution σ. Consider all the variables taking values in $\sigma(X_i) \in [4Bj\,..\,4Bj+2B]$. Assume that those variables are $X^j_1, \ldots, X^j_{m_j}$, and assume that $\sigma(X^j_1) < \cdots < \sigma(X^j_{m_j})$. This means that

- $\sigma(X^j_1) \geq 4Bj + c^j_1$ (constraint on the domain),
- $\sigma(X^j_2) \geq 4Bj + c^j_1 + (c^j_1 + c^j_2) = 4Bj + 2c^j_1 + c^j_2$ (`alldistant` constraint),
- $\sigma(X^j_3) \geq 4Bj + c^j_1 + (c^j_1 + c^j_2) + (c^j_2 + c^j_3) = 4Bj + 2c^j_1 + 2c^j_2 + c^j_3$ (`alldistant` constraint),
- and so on, until $\sigma(X^j_{m_j}) \geq 4Bj + 2(c^j_1 + c^j_2 + c^j_{m_j-1}) + c^j_{m_j}$

Moreover, for the constraint on the domain, it holds that $\sigma(X^j_{m_j}) \leq 4Bj + 2B - c^j_{m_j}$. This means that $2(c^j_1 + c^j_2 + \cdots + c^j_{m_j}) \leq 2B$. Thus, put all items associated to the considered variables to bin j to obtain a solution of the bin packing.

The vice versa is similar. Given a solution of the bin packing, for each bin j, consider the items $\texttt{item}^j_1, \texttt{item}^j_2, \ldots, \texttt{item}^j_{m_j}$ assigned in to the bin j. Then set $\sigma(X^j_1) = 4Bj + c^j_1, \sigma(X^k_2) = 4Bj + 2c^j_1 + c^j_2, \ldots, \sigma(X^j_{m_j}) = 4Bj + 2(c^j_1 + c^j_2 + c^j_{m_j-1}) + c^j_{m_j}$.

[4] For some lattice structures, it may be necessary to choose a different subset of points. The proof can be adapted by choosing, e.g., a collinear set of lattice points (some scaling of coefficients may be needed).

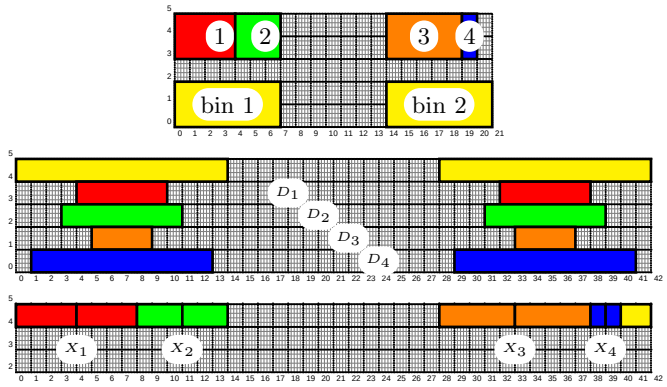


Fig. 15. An example of reduction from bin packing

The `density` constraint

The `Density1` problem is clearly in NP. To prove its NP completeness, let us reduce the bin packing problem to the `Density 1` problem.[5]

Let us define $\mathsf{box}[(x_1, y_1), (x_2, y_2)] = \{(x, y) \in \mathbb{N}^2 : x_1 \leq x \leq x_2, y_1 \leq y \leq y_2\}$. Consider an instance $a_1, \ldots, a_n, C, B$ of bin packing, where the a_i's are the tokens, C is the bin capacity, and B the number of bins. Let $M \geq 1$ be an arbitrary integer number. We can construct an instance of `Density 1` as follows:

- $k = B$ and $d_1 = \cdots = d_k = C$
- $R_1 = \mathsf{box}[(0, 0), (M-1, M-1)], R_2 = \mathsf{box}[(d, 0), (2M-1, M-1)], \ldots, R_k = \mathsf{box}[((k-1)M, 0), (kM-1, M-1)]$
- The n components provide density contributions $a_1, \ldots, a_n$
- The contribution functions are as follows:

$$\mathcal{F}_i(\boldsymbol{x}, \boldsymbol{p}) = \begin{cases} a_i, \text{if } \boldsymbol{x} = \boldsymbol{p} \\ 0 \text{ otherwise} \end{cases}$$

It is easy to see that this instance of bin packing has a solution iff the corresponding instance of `Density 1` has. □

Now, let $\alpha \in \mathbb{N}$ and $\beta \in \mathbb{N}$ be fixed. Then the `Density 2` problem is defined as follows.
Input:

- k disjoint regions $1, \ldots, k$, each described by:
  - R_i be the set of points of the region i
  - a (density) value $s_i \in \mathbb{R}^+ \cup \{0\}$
- n components with density contributions $a_1, \ldots, a_n$ chosen from a set of α different elements and such that $a_i\beta \geq \max\{d_1, \ldots, d_k\}$

[5] Observe that the bin packing problem is (strongly) NP-complete (see, e.g., [21, pp. 203–205]).

– n variables $X_1, \ldots, X_n$

Question: Establish whether there is a one-to-one assignment $\sigma : \{X_1, \ldots, X_n\} \longrightarrow \mathbb{N}^2$ such that for all $i = 1, \ldots, n$ formulae (2) hold.

We show that the `Density 2` problem can be solved in polynomial time. By hypothesis, each a_i is chosen in a set of α elements, say $\{m_1, \ldots, m_\alpha\}$. The set of components in a region R can therefore be identified as a tuple $\langle t_1, \ldots, t_\alpha \rangle$, where t_i represents the number of occurrences of components of value m_i in that region.

Let $S = \max\{d_1, \ldots, d_k\}$. By hypothesis, for all $j = 1, \ldots, n$, we have that $a_j\beta \geq S$. Thus, for all $i = 1, \ldots, \alpha$, we have that $m_i\beta \geq S$. We know that for every region R and every solution σ it holds that

$$\sum_{i=1}^{\alpha} t_i m_i = \sum_{j=1\ldots n, \sigma(X_j) \in R} a_j \leq S$$

Let $m = \min\{m_1, \ldots, m_\alpha\}$. Assume, by contradiction, that $\sum_{i=1}^{\alpha} t_i > \beta$. Then,

$$\sum_{i=1}^{\alpha} t_i m_i \geq \sum_{i=1}^{\alpha} t_i m > \beta m \geq S$$

which is an absurdum. Therefore, $\sum_{i=1}^{\alpha} t_i \leq \beta$ (and, in particular, $t_i \leq \beta$).

Now, let us find an upper bound for the number of these tuples. Let us distribute in a line α white balls (from left to right, one for each element m_i) and β black balls. The number of black balls immediately on the right to the i-th white ball denotes the number of occurrences of elements m_i in the region. For instance, if $\alpha = 4$ and $\beta = 3$:

- ● ● ● ○ ○ ○ ○ stands for $\langle 0, 0, 0, 0 \rangle$
- ○ ● ● ○ ● ○ ○ stands for $\langle 2, 1, 0, 0 \rangle$
- ● ○ ● ○ ● ○ ○ stands for $\langle 1, 1, 0, 0 \rangle$
- ● ● ○ ○ ○ ○ ● stands for $\langle 0, 0, 0, 1 \rangle$

It is easy to see that there is a disposition for each possible tuple. Thus, the number of possible tuples is $q = \frac{(\alpha+\beta)!}{\alpha!\beta!}$ and any possible solution can be given (and tested) by enumerating, for each value of q, how many regions are in that way. An upper bound is therefore $k^q = O(n^q)$. □