Computational Protein Design: trying an Answer Set Programming approach to solve the problem

Hugo Bazille, Jacques Nicolas

Computational Protein Design: trying an Answer Set Programming approach to solve the problem

Hugo Bazille, Jacques Nicolas

Inria centre de Rennes, Campus de Beaulieu, 35510 Rennes, France.

Abstract. Proteins are macromolecules made of a chain of amino-acids. The combinatorial nature of the space of possible protein conformations makes computer-aided protein study a major research field in bioinformatics. The problem of *computational protein design* aims at finding the best protein conformation to perform a given task. This problem can be reduced to an optimization problem, looking for the minimum of an energy function depending on the amino-acid interactions in the protein. We have designed a model based on Answer Set Programming. The CPD problem may be easily modeled as an ASP program but a practical implementation able to work on real-sized instances has never been published. We have raised the main source of difficulty for current ASP solvers and ran a series of benchmarks highlighting the importance of finding a good upper bound estimation of the target minimum energy to reduce the amount of combinatorial search. Our solution clearly outperforms a direct ASP implementation without this estimation and has comparable performances with respect to SAT-based approaches. It remains less efficient that the recent approach by cost function networks of D. Allouche & al., showing there exists still some place for improving the optimization component in ASP with more dynamical strategies.

1 Introduction

Proteins are essential compounds of living organisms, implied in almost all structural, catalytic, sensory, and regulatory functions. Proteins are amino-acid sequences with many different functions, mostly determined by their three dimensional structure. The study of these structures is thus an important field in biology with applications in various fields such as medicine, biotechnology, synthetic biology... [25]. Computer-assisted study of proteins offers opportunities to mimic the evolution and create new mutations or new structures in proteins [23].

The goal of *computational protein design (CPD)* is precisely to find among a collection of proteins those that most likely target a function. It is sometimes referred as the inverse folding problem: whereas protein folding is looking for the 3D structure of a given sequence, protein design searches the amino-acid sequences that would fold into a given 3D structure. As they are 20 possible amino-acids for each position in a protein sequence, each of them accepting several structural variants, the number of combinations to be tested is out of

range of any experimental approach, even for short sequences. Consequently, it became a strategic simulation challenge and numerous works have demonstrated the power of computer-aided protein design [1, 7, 11, 12, 16, 22]. Among the most striking results of this approach, one can mention the production of antimalarial drugs from the engineered bacteria *E. coli* [21] or the development of the most efficient computationally designed enzyme for the Kemp elimination to date [26].

In CPD, choosing the best amino-acid sequences to perform some function is formulated as an optimization problem. Even with the most recent advances in CPD, techniques still need to be improved: many approximations are made in order to make this problem solvable and more realistic models are needed. It is also necessary to generate a limited collection of solutions close to the optimum since, due to model approximations, the optimum of the combinatorial problem is not always the best protein in practice.

This paper presents a specific approach to solve the CPD problem, Answer Set Programming (ASP) [20]. This work is based on the previous results of [10], with a number of enrichments. In order to evaluate this work, two comparison studies were carried out: the first one quantifies the progress with respect to [10], and the second one is a more demanding task addressing the most advanced results in the domain [1, 2, 29]. The ASP model performances do not reach the level of the best current approaches and a first discussion on the advantages and drawbacks of the ASP approach for solving the CPD problem is provided.

We start by giving the biological background of this work and defining the CPD problem with a fixed rigid backbone and discrete sets of possible rotamers in proteins. Then, we give an overview of state of the art, with various techniques and paradigms. Next, we recall basic ASP notions and give our encodings of the CPD problem. Results are described in a last section, together with the analysis of the different encodings. We conclude by different perspectives that seem interesting to be studied in a future work.

2 The computational protein design problem (CPD)

The protein design problem is an optimization problem on the conformation (geometrical structure) of a protein whose structure is partially known.

2.1 Elements of protein structure

Proteins are biological macromolecules made of several *amino acids* linked together by *peptide bonds*. There are mainly 20 different amino acids with a common organization illustrated in figure 1. The 'R' represents the *side-chain* that makes the amino acid unique. An amino acid in a polypeptide is also called a residue. The sequence of carbon, oxygen and nitrogen atoms without the side chain defines a 3D structure called *backbone*. The geometry of the protein is relatively rigid and generally defined by three *dihedral angles* (φ, ψ and ω) for each amino acid along the backbone. A dihedral angle is the angle between two

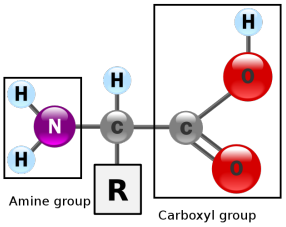


Fig. 1. Structure of an amino acid [1]

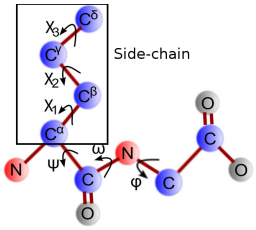


Fig. 2. Diihedral angles [2]

planes defined by a sequence of four atoms. Furthermore, each side-chain has up to four degrees of freedom: the dihedral angles χ (see an example figure 2).

The backbone structure is highly constrained (the 6 atoms implied in the peptide bond $C_\alpha C(O)NHC_\alpha$ form a plane). This paper assumes that its structure has no degree of flexibility. In contrast, side chains have different possible conformations called *rotamers*. Whenever there is a degree of freedom, there is an infinite number of possible rotamers, as the molecules can continuously rotate around the axis. However, it is often sufficient to consider a representative finite set of rotamers, which can be determined by a statistical analysis of the actual conformations. For each amino acid, there are from 1 to 30 of most common rotamers that are listed in public libraries. We have used the 2010 version [28] of the Dunbrack backbone-dependent rotamer library [14].

2.2 Protein design through energy minimization

The aim of CPD is to find a protein that will perform a desired function. This function depends on the backbone structure and on the rotamer configurations. For a given backbone structure, the protein tends to adopt a global stable configuration of minimal energy. Therefore, the practical goal of CPD is to find a sequence of residues and a conformation that folds into a defined backbone and minimizes the protein energy, which can be expressed as an optimization problem. The computation of realistic energy functions is a whole topic by itself and is not the subject of this paper. Energy values are part of the benchmark data sets we have used and they are considered here as input facts. The energy minimization depends only on the energy of interaction between rotamers or between the backbone and rotamers. Large scale protein design uses pairwise approximation: the energy function is supposed to be pairwise decomposable on residues. Moreover, some design positions are chosen to be mutated. With i_r the rotamer at position i, $E(i_r)$ the energy of interaction between rotamer i and the backbone, and $E(i_r, j_{r'})$ the energy of interaction between rotamers r at position i and r' at position j, the formula to minimize is:

$$E = \sum_i E(i_r) + \sum_i \sum_{j,j<i} E(i_r, j_{r'})$$

[1] From http://www.protocolsupplements.com/Sports-Performance-Supplements/wp-content/uploads/2009/06/amino-acid-mcat1.png

[2] From http://www.biomedcentral.com/content/figures/1471-2105-12-S14-S10-1-1.jpg

The energy functions $E(i_r)$ and $E(i_r, j_{r'})$ are based on empirical measures and depend on many parameters such as van der Waals potentials, electrostatics,...

Protein design is a NP-hard problem [24] associated to a huge hypothesis space. Indeed, a protein with m residues and a mean number of n rotamers per amino acid gives rise to $(20 \times n)^m$ possible conformations. Using reasonable parameters such as $m = 100$ and $n = 10$ leads to $\approx 10^{234}$ possible solutions. Variants of this problem are also studied, such as protein design with continuous rotamers or with backbone flexibility [11].

3 State of the art for CPD

In this section, we introduce a few state of the art algorithms to solve this problem. A first part introduces algorithms that are allowed to find approximations of the best solution and are supposed to scale to larger proteins. A second part introduces algorithms that are designed to find exactly the best solution.

3.1 CPD with approximations

Genetic algorithm based approaches have been tried on the CPD problem [27]. Each element of the population is a sequence of amino-acids with their rotamer conformations. A cross-over operator is defined by exchanging amino-acids at a position in two elements of the population. A mutation operator is defined by introducing a new amino-acid at some position. The algorithm discards the conformations of higher energy and a new population is generated by applying operators on the remaining rotamers. This knowledge-poor approach was applied early on CPD [13] but other stochastic methods are now preferred for this task.

Rosetta [19] is a large open source package that is the most representative of this tendency. It uses the Dunbrack backbone-dependant library to reduce the set of conformations and a Monte-Carlo based method to sample this space [12, 15]. It takes into account several factors to build a more precise energy function: van der Waals potentials, electrostatics... Even if is not used as a solver, Rosetta offers many functionalities and was useful in our first series of experiments to compute the needed interaction energy values.

SCWRL4 [16] is a program used to solve a restriction of the protein design problem: side-chain positioning. In this problem, amino acids are fixed and only the side-chain conformation is varying. This assumption greatly reduces the search space. An interaction graph that represents the side-chain placement problem is first created. The graph is decomposed into trees on which a branch-and-bound search can be performed. A rotamer collision detection step reduces the risk to produce physically impossible solutions. Most of the time, tree search algorithms are looking for exact solutions.

3.2 Exact search algorithms

As the search space of the CPD problem can be seen as a decision tree, some of the early methods to explore it have been based on graph traversal algo-

rithms such as A* [18]. The A* algorithm needs a heuristic function to perform the search. This function is calculated by adding in a first part the energy for all nodes already assigned and in a second part a suboptimal function over unassigned nodes. For each of node i_r, the suboptimal function is $E(i_r) + \sum_{j_s \ assigned} E(i_r, j_s) + \sum_{k \ unassigned} min_{r'} E(i_r, k_{r'})$. The interested reader may consult [11] to look at the current results achieved by such methods.

The problem of minimizing the total free energy of a protein conformation can also be represented as an Integer Linear Program (ILP). Indeed, binary variables $q_i(r_i)$ can represent the presence of a rotamer at place i. The presence of exactly one rotamer at each position is expressed by the formula $\sum_{r_i} q_i(r_i) = 1$. The energy minimization is then represented by a minimization term

$$min \sum_i \sum_{r_i} q_i(r_i) \ E(r_i) + \sum_{j \neq i} \sum_{r_j} E_{i,j}(r_i, r_j) \ q_i(r_i) \ q_j(r_j).$$

This approach is extended in [30] where only some variables have integer domains (Mixed ILP Programming).

CPD can also be formulated as a satisfiability problem with minimization of a formula. For each rotamer and each position, a boolean variable indicates if it is chosen or not. This approach enables to use SAT solvers, and effective encodings with improved branch and bounds algorithms are presented in [22]. Basically, given a position i and a set of possible rotamers $i_1 \dots i_r$ the problem of choosing one rotamer can be encoded by formula $\Phi_i = (i_1 \vee \dots \vee i_r) \wedge \bigwedge_{s \neq s'} \neg(i_s \wedge i'_s)$. Then, a global formula can be established: $\Phi = \bigwedge_i \Phi_i$. Clauses of the form $\neg(i_s \wedge j_{s'})$ derive from the computation of incompatible pairs of rotamers. In practice, hierarchical decisions are made: first the amino acid is assigned, then the dihedral angles (side chain). These programs are quite efficient, provided they take into account only three possibilities by dihedral angle.

The most recent advance in solving the CPD problem proposes to model it as a weighted constraint satisfaction problem [1, 2, 29], using cost function networks. A Cost Function Network (CFN) is a pair (X, W) where X is a set of n variables and W is a set of cost functions. Each variable $i \in X$ has a finite domain D_i of values. A value $r \in D_i$ is denoted i_r. For a set of variables $S \subseteq X$, D_S denotes the cartesian product of the domains of the variables in S. A cost function $w_S \in W$, with scope $S \subseteq X$, is a function $w_S : D_S \rightarrow [0, k]$ where k is a maximum integer cost used for forbidden assignments. There is a natural correspondence between the formalism of CFN and the elements of CPD: each variable i corresponds to a position to be redesigned and each value r in domain D_i corresponds to a possible rotamer r at position i. Furthermore, each unary cost function $w(i_r)$ represents the energy of interaction rotamer-backbone, and each binary cost function $w(i_r, j_{r'})$ represents the energy of interaction between two rotamers. This approach has led to major improvements and it dramatically outperforms previous works for exact results. It allows to solve big instances with tens of positions and thousands of rotamers.

We have studied yet another framework, Answer Set Programming (ASP), which has been tried only once to our knowledge, in a Master thesis [10], where

a direct translation of the problem specification has been tested, together with several variants such a as a separated treatment of positive and negative energy score values. The ASP solvers appeared to be able to solve only a very low number of residues. Since ASP has exhibited interesting performances in a number of combinatorial optimization problems, we have tried to understand the reasons of this relative failure. An earlier paper points to the difficulty of solving the related problem of protein folding on a lattice [5] but the ASP solvers have made progress since then and the interest of CPD is to offer a combinatorial problem where optimization plays a major role. We have emphasized the specific features of the problem that make it difficult and looked at better encodings, achieving clear progress with respect to the way its resolution could be improved.

4 Modeling CPD using Answer Set Programming (ASP)

4.1 A bit of syntax and the solver clingo4

Answer Set Programming (ASP) is a form of declarative logic programming using the the stable models semantics. It is a Boolean constraint solving framework designed for Knowledge Representation and reasoning and for complex optimization problems: solvers for ASP programs implement heuristics in order to solve efficiently minimization queries over weighted atoms.

This document follows the current standard of ASP language[1].

Given a set of atoms A, a *normal logic program* over A is a finite set of normal rules (a_0 is called the *head* and the rest is the *body*):

$a_0 :- a_1; \ldots; a_m; not\ a_{m+1}; \ldots; not\ a_n$, where $a_i \in A$ for all i, .

It is possible to use predicates of any arity and first order variables in ASP, provided that these variables are defined over a finite domain. Each variable will be replaced during a grounding phase by its possible values in the Herbrand universe of the program, leading to a fully instantiated program.

It is possible to write integrity constraints, which are rules with no head: $:- a_1; \ldots a_m;\ not\ a_{m+1}; \ldots\ not\ a_n$.

A cardinality rule is of the form $a_0 :- l\{a_1; \ldots; a_m;\ not\ a_{m+1}; \ldots;\ not\ a_n\}u$ It allows to control the cardinality of sets of atoms, with lower bound l and upper bound u. Similarly, a choice rule is of the form:
$l\{a_1; \ldots; a_m\}u :- a_{m+1}; \ldots; a_n;\ not\ a_{n+1}; \ldots;\ not\ a_p$,
meaning that if the body holds, at least l and at most u atoms of the head have to hold. Such rules are expanded in a number of normal rules quadratic in m.

ASP systems include statements to express cost functions [8] and multi-criteria optimization. With w_i the weight of literal l_i and p_i its priority level, the minimize statement writes #minimize $\{w_1@p_i : l_1 \ldots; w_n@p_n : l_n\}$. The solver minimizes the sum of the weighted literals, starting by the highest priority level.

The first ASP solvers were quite "monolithic": first the grounding generated a propositional program, then a solver computed its stable models. The solvers use enhancements of the *Davis-Putman-Logemann-Loveland* (*DPLL*) algorithm,

[1] https://www.mat.unical.it/aspcomp2013/ASPStandardization

analyzing and learning conflicts in case of failure to prune the search space. In the recent versions of the ASP system *clingo4*, two scripting languages have been integrated with ASP in a common environment in order to achieve complex reasoning processes: *Python* and *Lua* [9]. On the declarative side, it becomes possible to define procedures allowing to instantiate different logic programs with different parameters (directive #program) and they may be grounded and solved at any time using built_in procedures ground() and solve(). External rules ("i. e. "volatile rules ") may also be added/removed in a program (#external).

We have chosen clingo4 because it is one of the most efficient ASP solver to date and because it offers large possibilities to adjust the solving strategy. The user can easily design its program by incremental refinements or produce high level interactive solvers. A client-server ASP solving process encoded in Python is presented in [9]. Scripting has been used in our work to combine the search for approximate solutions and local exploration. Since clingo4 is still evolving, this kind of study can help its designers to improve the optimization component.

4.2 Algorithms and ASP models to solve the CPD

DEE pruning algorithms Like in every combinatorial problem, the main issue in CPD is the very fast increase of the size of the search space with respect to the size of the protein. The goal of *Dead End Elimination* (*DEE*) algorithms is to remove the choice of some residue at some place because it always gives worse results than other residues at this place. They can also be extended to eliminate pairs of residues, triplets... Numerous DEE algorithms are available that have different complexities and efficiencies We have used two of them, *simple split* and *double Golstein*, offering a good tradeoff with respect to these parameters.

Simple split eliminates a rotamer i_r if $\exists k, \forall v, \exists i_t$,

$$E(i_r) - E(i_t) + (E(i_r, k_v) - E(i_t, k_v)) + \sum_{j \neq k \neq i} min_u[E(i_r, j_u) - E(i_t, j_u)] > 0 \quad (1)$$

If p is the number of positions and n the number of rotamers at some position, the complexity of the algorithm is $O(p^2n^3)$. The Double Goldstein DEE eliminates a pair of rotamers (i_{1r_1}, i_{2r_2}) if there exists a pair (i_{1t_1}, i_{2t_2}) such that

$$(E(i_{1r_1}) + E(i_{2r_2}) + E(i_{1r_1}, i_{2r_2})) - (E(i_{1t_1}) + E(i_{2t_2}) + E(i_{1t_1}, i_{2t_2})) + \sum_{j \neq i_1 j \neq i_2} min_u(E(i_r, j_u) - E(i_t, j_u)) > 0 \quad (2)$$

The complexity of the algorithm is $O(p^3n^5)$, a noticeable increase of complexity that allows to eliminate a high number of pairs. In practice, this algorithm is launched after the first one to decrease the value of n.

Search for exact results The basic ASP code for the CPD problem is introduced as in [10] in order to serve as a reference point for the improvements

that are tested thereafter. The set of positions, of possible residues and possible rotamers at each position have been represented by the following facts :
`position(Position_in_the_protein).`
`possibleResidue(Position,Aminoacid).`
`possibleRotamer(Position,Aminoacid,Rotamer,EnergyB).`

EnergyB is the interaction energy with the backbone. Then the interaction energies between rotamers are tabulated from *Rosetta* or *Osprey* computations:
`interEnergy(Pos1,AA1,Rot1,Pos2,AA2,Rot2,EnergyR).`

Given these facts as input data, the CPD problem can be expressed in 3 rules. For each position, one residue and one rotamer are chosen to be in the solution. It is recorded in predicate residue(Pos,AA) and rotamer(Pos,AA,Rotamer).

Algorithm 1 `Exact encoding`

```
% Exactly one residue must be assigned to each position                (a)
1{residue(Pos,Amin) : possibleResidue(Pos,Amin)}1 :- position(Pos).

% Exactly one rotamer must be assigned to each residue                 (b)
1{rotamer(Pos,Amin,Id) : possibleRotamer(Pos,Amin,Id,Energy)}1
                              :- residue(Pos,Amin), position(Pos).

% Minimize the sum of the energies                                     (c)
#minimize{ EnergyB@1,Pos,Id:
           rotamer(Pos,Amin,Id),possibleRotamer(Pos,Id,EnergyB) ;
           EnergyR@1,Pos1,Pos2:
           rotamer(Pos1,Amin1,Id1),rotamer(Pos2,Amin2,Id2),
           interEnergy(Pos1,Amin1,Id1,Pos2,Amin2,Id2,EnergyR)}.
```

- Rule (a) chooses a residue among the possible residues at each position;
- Rule (b) chooses a rotamer among possible rotamers for each chosen residue.
- Rule (c) minimizes the sum of the recorded energies predicates. The weight of each predicate is the value of its *Energy* parameter.

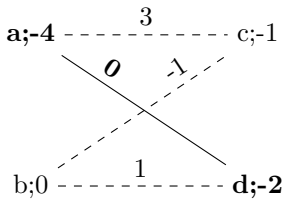
As for the SAT specification of CPD, a file containing the list of pruned rotamer pairs computed by DEE algorithms before the search is added to this program. An eliminated pair is taken into account by an integrity constraint:
`:- rotamer(Pos1,Id1), rotamer(Pos2,Id2).`

This short program computes exact results but is limited to very small instances as runtime may quickly become too long. The optimization step is crucial in CPD and we have studied the way it is managed by clingo to understand the limitations and improve the code.

Transformation in an equivalent problem with better properties

Different minimization algorithms are implemented in the ASP solver (branch and bound, unsatisfiable core...) but they all require positive values. This means

that programs are automatically rewritten to fulfill this constraint in order to transform the minimization problem into an equivalent problem with only positive values and lower bound 0. This rewriting algorithm is quite simple: each time there is a minimization statement of the form `#minimize { V:p }.`, $V < 0$, the solver transforms it in `#minimize { -V: not p}`. It appears that the transformation is far from being optimal in our problem. Let us consider a short example:



In this schema, it is possible to choose rotamer a or b at position 1 and rotamer c or d at position 2. The minimal solution of this problem is the pair (a, d), a configuration with total energy -6 (-4+-2+0). Solving this problem with (an adaptation of) the naive encoding given in section 4.2 leads to the good answer, but the optimization value reached in the transformed system is 2.

In details, weights are added to the energy of the solution depending on the following conditions: 4 if a is **not** chosen, 0 if b is chosen, 1 if c is **not** chosen, 2 if d is **not** chosen 3 if the pair (a, c) is chosen, 0 if (a, d) is chosen, 1 if (b, c) is **not** chosen, 1 if (b, d) is chosen. On this example, the final optimization value is not too far from the lower bound 0. On real instances with many negative values it is no more the case in general and since the choice of a rotamer entails that all others are not chosen and it is difficult for the solver to find good lower bounds.

To bypass this issue, the problem is transformed such that minimization is carried over positive values only and the solver can estimate a much better lower bound. This issue may be stated as a constraint satisfaction problem and we apply a technique imported from this field, constraint propagation, to reduce the set of possible values. This is achieved by looking for two local consistency properties, soft arc consistency, which is propagating energy constraints on pairs of rotamers, and soft node consistency, which is propagating energy constraints on a single rotamer with the backbone. The principle is used in the cost function network approach [4, 17] where it appeared to be a major source of efficiency in solving the CPD problem. The idea is the following one: for each position i, let $N_{i,1}, \ldots, N_{i,r}$ be the interaction energy of rotamers $i_1, \ldots, i_r$ with the backbone, and let $N_i = min_j N_{i,j}$. Then, the energy N_i may be considered as a fixed cost $E_\emptyset$ associated to any choice in the position and it is only necessary to search for the minimization on the energies $N_{i,1} - N_i, \ldots, N_{i,r} - N_i$ that are all positive. A similar process can be done for the energies of interaction between rotamers (see also the hierarchical approach below).

In the previous example, the minimum energy of interaction with the backbone is -4 at position 1 and -2 at position 2. Thus any choice over positions 1 and 2 includes a fixed cost of interaction with the backbone equal to -4+-2= -6, which is the best possible lower bound in this case, and the cost of a, b, c and d may be respectively replaced by 0, 4, 1 and 0 for the minimization.

In terms of the ASP program, this means that all predicates $possibleRotamer(Pos, AA, Rotamer, Energy)$. will be changed in $possibleRotamer(Pos, AA, Rotamer, Energy - MinEnergy\_at\_Pos)$.

It is a bit more complex since the energies of interaction between rotamers have also to be transformed. For instance in the previous problem, the minimum energy of interaction between rotamers once b has been chosen is -1 and thus the interaction energy -1 may be added as a fixed cost for the choice of b.

Add a hierarchy to the encoding

In a branch and bound search, the sooner a cut can be made, the better it is. We propose to cluster rotamers with similar properties at some position in order to be able to do more cuts sooner. Fortunately, there is a natural way to achieve this clustering in constant time: regroup the rotamers by amino-acid since they share in general similar interaction energies.

Let i be a position of an amino-acid A with rotamers $i_{A_1} \dots i_{A_r}$, having energies of interaction with the backbone $E(A_1) \dots E(A_r)$. If $N = \min E(A_1) \dots E(A_r)$. Then energy $E'(A) = N$ is associated to the choice of A and energies $E'(A_1) = E(A_1) - N \dots E'(A_r) = E(A_r) - N$ to the choice of $i_{A_1} \dots i_{A_r}$. It allows to replace r cuts at the rotamer level by a single cut at the amino-acid level. The same principle may be applied for pairs of rotamers. For amino-acid A at position i with conformations r and B at j with conformations r',one defines $(i_A, j_B) = argmin_{r,r'} E(i_r, j_{r'})$. Then energies $E(i_r, j_{r'})$ are replaced by energies $E'(i_A, j_B)$ and $E'(i_r, j_{r'}) = E(i_r, j_{r'}) - E(i_A, j_B)$.

Algorithms 2 and 3 correspond to a sample of transformation steps described previously to get an equivalent program, adapted for the hierarchical encoding. Algorithm 4 gives an overview of the whole process. The lower bound is iteratively improved as much as possible for every position. Its complexity is $O(p^2n^3)$ (with p the number of positions and n the number of rotamers by position), the same complexity that most used DEE algorithms. More on arc consistency enforcing algorithms may be found in [3].

Algorithm 2 Energy transfer from rotamers to a residue.

```
E ← min E(A_r)
for all rotamers A_r at position i do
    E(i_{A_r}) ← E(i_{A_r}) − E
end for
E(i_A) ← E
```

4.3 Enumeration of ε-solutions

As we work on an abstraction of a biological problem, some approximations are made and solutions of the optimization problem may not be solutions of the CPD problem. Thus solutions close to the optimum are interesting too. In order to produce a set of good solutions with a fixed divergence from the global minimum solution, we assume the minimum value E_{best} has been produced and restart the algorithm with a modified DEE pruning including a parameter ε in the right

Algorithm 3 Energy transfer from pairs to rotamers.

```
for all position j do
    E ← min E(i_{A_r}, j_{B_{r'}})
    for all rotamers B_{r'} at position j do
        E(i_{A_r}, j_{B_{r'}}) ← E(i_{A_r}, j_{B_{r'}}) − E
    end for
    E(i_{A_r}) ← E(i_{A_r}) + E
end for
```

Algorithm 4 Enforcing consistency

```
for all position i do
    for all amino-acid A at position i do
        for all position j ≠ i do
            for all amino-acid B at position j do
                for all rotamers A_r at position i do ▷ from _residue_to_rotamers(j,B)
                    E(i_{A_r}) ← E(i_{A_r}) + E(i_A)
                end for
                E(i_A) ← 0
            end for
            for all rotamer r' at position j do
                for all rotamers B_{r'} at position j do ▷ from _rotamer_to_pairs(j,r',i)
                    E(i_{A_r}, j_{B_{r'}}) ← E(i_{A_r}, j_{B_{r'}}) + E(i_{A_r})
                end for
                E(i_{A_r}) ← 0
            end for
        end for
        for all rotamer A_r at position i do
            Energy_transfer_from_pairs_to_rotamer(i,A_r)
        end for
        Energy_transfer_from_rotamers_to_residue(i,A)
        for all position j ≠ i do
            for all rotamer B_{r'} at position j do
                Energy_transfer_from_pairs_to_rotamer(j,B_r)
            end for
            for all amino-acid B at position j do
                Energy_transfer_from_rotamers_to_residue(j,B)
            end for
        end for
        E ← min E(i_A)
        for all amino-acids A at position i do    ▷ from residues to the lower bound
            E(i_A) ← E(i_A) − E
        end for
        E_∅ ← E_∅ + E
    end for
end for
```

members of the equations in 4.2. It is also possible to adapt DEE algorithms to discard rotamer i_r if a lower bound of the energy $E_{lower}(i_r)$ when i_r is chosen is greater than $E_{best} + \varepsilon$. The cost of obtaining lower bounds increases when the required precision increases. A possible lower bound is

$$E_{lower}(i_r) = E(i_r) + \sum_{j \neq i} min_{r'}[E(j_{r'}) + E(i_r, j_{r'})] + \sum_{j \neq i} \sum_{k > j, k \neq i} min_{r',r"}[E(k_{r"}, j_{r'})]$$

Computing this lower bound for all rotamers at all possible place has complexity $O(n^3p^3)$ with n the number of rotamers and p the number of positions. Similar criteria to eliminate pairs of rotamers have complexity $O(n^5p^4)$. After pruning the search space, the search itself can be launched. An ASP program doing this task may be constituted of the program `Exact encoding` with an additional rule stating that the sum of energies must not be greater than $E_{best}+\varepsilon$. A 2% variation of E_{best} may increase the number of solutions to several thousands.

5 Results

Benchmarks have been run on two different datasets: the first one is extracted from [10], which contained initially 120 instances. They are based on 12 instances of 10 proteins , considering 10, 12, 15 and 17 positions and three different amino-acid sets (fixed, only hydrophobic or all aminoacids). We retained the 40 most difficult problems. It serves to compare the progress made between [10] and our own work. The second dataset is extracted from [1] and contains 47 instances. It allows us to compare our results to the last advances in the domain. The solver clingo4 has been run with default parameters. Trying dfferent solver options for clasp (particularly for the choice of heuristics) has shown no significant improvement. All runs have been performed on a Intel Xeon W3520 quad-core, 2.66 GHz. All reported times are in seconds.

5.1 DEE

The efficiency of DEE algorithms has been evaluated on the first dataset. Results are presented in table 1 (columns 4 to 7). For each instance, we perform 3 successive runs of Goldstein DEE (a fast algorithm) that allows to run then 5 times Simple Split (a complex algorithm) on a highly reduced dat set. Note that more runs can be necessary to reach a fix point but it would not further improve the total search time. Several facts are interesting here:

– First, except for very small instances, the percentage of eliminations decreases with the size of the instance, as shown in figure 3. The same phenomena applies for the elimination of pairs . This may be explained by the fact that there is a "forall" in the elimination criterion increasing the stringency of the criterion with the number of possibilities.

Protein	Size	AA clusters	# Rotamers before	# Rotamers after	Discarded pairs	DEE time	Time in [10]	Time in our work
1BE9	17	All	2547	1720	270811	2919	!	-
		Hydro	529	130	1903	3.2	-	0.66
	15	All	2063	1023	122977	1063	!	155
		Hydro	435	56	270	1.3	1.719	0.05
1I92	17	All	2648	1728	288332	3262	!	-
		Hydro	593	186	4318	6.0	-	1.26
	15	All	2055	816	86194	806	!	1174
		Hydro	455	93	1042	2.1	-	0.10
1MFG	17	All	2608	1798	299418	3145	!	-
		Hydro	560	188	3717	5.3	-	0.51
	15	All	1972	854	106835	759	!	336
		Hydro	410	74	485	1.3	679	0.07
1N7F	17	All	2548	1525	211868	2473	!	-
		Hydro	544	136	1811	4.2	-	0.21
	15	All	1934	648	59369	477	!	15.6
		Hydro	402	57	202	1.2	66	0.047
1QAU	17	All	2643	1776	288165	1598	!	-
		Hydro	562	196	4177	6.2	-	0.73
	15	All	2023	895	101761	1228	!	98
		Hydro	421	97	1021	1.8	2295	0.10
1RZX	17	All	2722	1748	257158	3398	!	-
		Hydro	554	167	3044	4.5	-	0.37
	15	All	2121	941	94310	883	!	906
		Hydro	404	56	205	1.1	36	0.05
1TP3	17	All	2582	1732	282007	3145	!	-
		Hydro	536	142	2253	3.8	-	0.19
	15	All	2142	1037	135604	1057	!	403
		Hydro	447	63	409	1.7	463	0.054
2EGN	17	All	2548	1570	240733	2478	!	-
		Hydro	546	150	2516	3.7	-	0.32
	15	All	2005	804	78377	573	!	1038
		Hydro	413	75	655	1.4	280	0.039
2FNE	17	All	2736	1909	325100	3927	!	-
		Hydro	577	197	4639	6.1	-	0.49
	15	All	2144	1060	132397	1158	!	154
		Hydro	438	98	1093	1.7	4976	0.10
2GZV	17	All	2934	1991	348785	5243	!	-
		Hydro	613	189	3823	7.5	-	0.62
	15	All	2247	1122	132910	1340	!	1242
		Hydro	456	87	696	2.1	-	0.09

Table 1. Results of algorithms performed on the first series of instances

– The second fact is that elimination takes a long time on larger instances: due to the cubic complexity it is not possible to launch the double Goldstein DEE on largest instances in a reasonable time. However, even if double Goldstein cannot be applied, simple split DEE discards many pairs (see column 6).

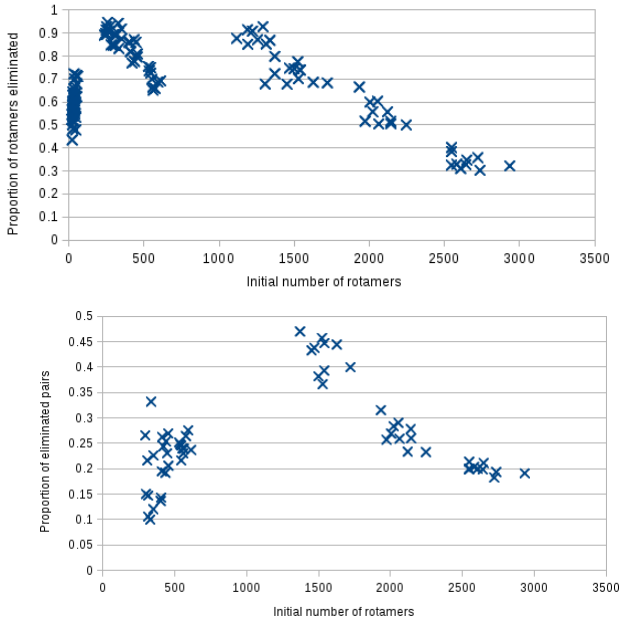


Fig. 3. Bigger the instances are, smaller the ratio of eliminated rotamers or pairs is

5.2 Exact searches

The first series of results illustrates the progress made over [10] (columns 8 and 9 of table 1). A time limit has been set to 3 hours. A "-" denotes a problem that could not been solved due to time limit and a "!" denotes a problem that could not been solved due to insufficient memory. Many more instances have be solved, and none crashes because of memory. The difficult problems occur around 1800 rotamers instead of 100 in the previous approach. With the preprocessing we have implemented, the ASP solver is able to infer a good lower bound and it dramatically reduces the search space. Note that the simple idea consisting of reducing the energy computation to a spatial neighborhood of residues does not work in practice to break the complexity.

The second series of instances is taken from [1] and has been built using Osprey2.0 [6]. These instances are far more complex, as they deal with several tens of positions to be redesigned and use a different library of rotamers that cannot be easily eliminated by DEE. Then, the threshold for instances that can not be solved is lower.

Protein	Positions	# Rotamers	ASP(s)	Toulbar2(s)	Cplex(s)	maxhs
2TRX	11	410	0.4	0.1	2.6	4086
1HZ5	12	427	0.3	0.1	7.6	5695
1PGB	11	438	2.7	0.1	3.6	5209
1MJC	28	440	151	0.1	4.1	3698
1UBI	13	498	209	0.2	139	-
1CSK	30	508	2030	0.1	9.6	-
1SHF	30	527	-	0.1	8.6	-
2PCY	18	598	2589	0.2	26.9	-
1SHG	28	613	366	0.2	39.4	-
1NXB	24	625	-	0.2	17	-
1FNA	38	887	-	0.5	121	-
1CSP	30	1026	146	0.84	1264	-
1BK2	24	1089	6.8	0.65	125	-
1LZ1	59	1202	-	1.5	1084	-
1FYN	23	2110	-	2.8	3136	-
1CM1	17	2242	-	3.3	473	-

Table 2. Comparison with the most recent tools in CPD

The solver *Toulbar2* presented in [1] has by far the best performances. Instances (not in this table) that could not be solved with it could not be solved with other solvers. We wanted to compare ASP with *Protsat*[2] [22], but the repository is empty and we could not reach the authors. Then, we compared to *maxhs*, another core-based SAT-solver, also used in benchmarks in [1]. *Maxhs* is not very effective: many instances could not be solved. The most interesting comparison is with *Cplex*, a sophisticated solver for ILP. On some instances, our ASP-based solver outperforms *Cplex*, however on larger instances, *Cplex* takes the lead. It means that something in our approach makes it less scalable.

5.3 Enumeration of neighboring solutions

We ran solvable instances from the first series characterized by 17 positions to be redesigned and between 500 and 700 rotamers. For each instance, we have enumerated the solutions for ε ranging from 500 to 4500 with step 500. It represents about 0.5 to 8% of the free energy for these instances.

Figure 4 represents the number of solutions and time needed to produce them. They grow exponentially with ε. The main fact is that it is easy to get a big set of candidate proteins in small time if the optimal solution is easily found. On larger instances, the search for a set of proteins becomes hard to solve.

5.4 Explaining the differences

The first problem is the search for a lower bound. Our transformation algorithms are not the best possible ones: on big instances, the lower bound is far from the

[2] http://sourceforge.net/projects/protsat/

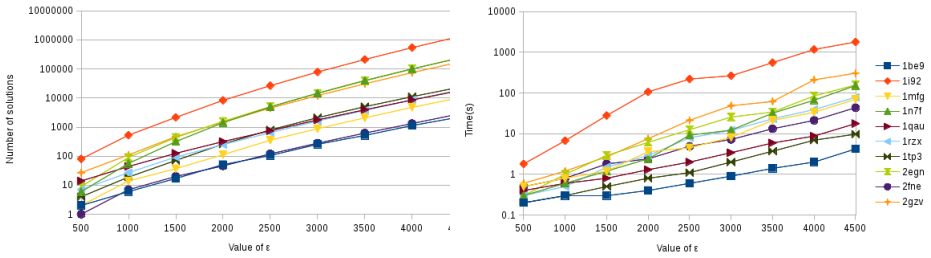


Fig. 4. Number of solutions and solving time depending on ε

optimal solution unlike in *Toulbar2* that solves many instances in less than a second since almost all rotamers are eliminated at first stages of the search. More sophisticated lower bound estimation are needed.

The ASP solver clingo4 chosen for this study had the advantage of proposing primitives to access the grounding and solving phases through a dedicated API. It is no more a black box but it still remains a "grey" box in the version we used (4.3.0): even if the user can parameterize a lot of features (optimization strategy, policy of restart...), enforcing some procedure such as node and arc consistencies each time a choice is made was not possible. Systems such as *Toulbar* are relatively slow in exploring the search space, but they offer a better preprocessing and the possibility to interact more accurately with the solver.

Finally, it would require more work to study the influence of built-in ASP configurations on the search (frumpy, handy, crafty...).

6 Conclusion

The goal of this study was to evaluate the potential of ASP in solving the CPD problem. Our work lies in the scope of exact searches and can be compared to many different techniques: ILP, 01QP, MMRF, A* searches, CFN ... We have chosen for the benchmarks some of the most recent and best approaches, which inspired our own work. We have established a first milestone on the subject, and are confident that it can still be largely improved.

Trying to reach performances of CFN is an interesting challenge, and may mobilize different competences. We also studied different approaches and proposed different techniques to quickly get approximated results that could be redirected as new constraints in the solver. Refining the use of ASP solver heuristics and improving the search for lower and upper bounds are the main perspectives of this research. We can also pinpoint that in practice, due to the approximations and the choice of energy function, the most stable structure may slightly differ from the computed optimum.The extension to neighboring solutions can be easily implemented in ASP by launching another solve process with an extended search after the search of the optimal solution, but this strategy suffers from

the disadvantages of exact search: the search for a lower bound is worse than in many other approaches and then the search time grows much faster.

Among other variants, it may also be interesting to study the modeling of the CPD problem with a flexible backbone in ASP. On this problem, the best results are given by software such as *Osprey*. The most difficult challenge for ASP would be to solve the CPD problem with a continuous space of rotamers.

A great interest in trying different approaches to solve problems such as CPD may lead to new general ideas to solve difficult problems and to ameliorate solvers by merging the best ideas of each approach. As an interesting by-product of this work, we have identified some current limitations of the *clingo*'s minimization process and this should help improving the optimization part in the solver.

Acknowledgments

Authors would like to thank I. Lynce and F. Gouveia at INESC-ID Lisboa and T. Schiex, S. Barbe and D. Allouche at INRA Toulouse for having made this study possible with their own work and having kindly provided their benchmarks. We also thank the reviewers for their constructive suggestions.

References

1. D. Allouche and al. Computational protein design as an optimization problem. *Artif. Intel.*, 2014.
2. D. Allouche, S. Traoré, I. André, S. de Givry, G. Katsirelos, S. Barbe, and T. Schiex. Computational protein design as a cost function network optimization problem. *Proceedings of the 18th Int. Conf. on Principles and Practice of Constraint Programming*, pages 840–849, 2012.
3. C. Bessière, J.C. Régin, R. H.C. Yap, and Y. Zhang. An optimal coarse-grained arc consistency algorithm. *Artif. Intel.*, 165(2):165 – 185, 2005.
4. Martin Cooper and Thomas Schiex. Arc consistency for soft constraints. *Artif. Intel.*, 154(12):199 – 227, 2004.
5. Agostino Dovier, Andrea Formisano, and Enrico Pontelli. A comparison of clp (fd) and asp solutions to np-complete problems. In *Proc of International Conference on Logic Programming*, volume 3668, pages 67–82. Springer Berlin Heidelberg, 2005.
6. P. Gainza, KE. Roberts, I. Georgiev, RH. Lilien, DA. Keedy, CY. Chen, F. Reza, AC. Anderson, DC. Richardson, JS. Richardson, et al. Osprey: protein design with ensembles, flexibility, and provable algorithms. *Methods Enzymol.*, 523:87, 2013.
7. Pablo Gainza, Kyle E. Roberts, and Bruce Randall Donald. Protein design using continuous rotamers. *PLoS Computational Biology*, 8(1), 2012.
8. Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. On the implementation of weight constraint rules in conflict-driven asp solvers. In *Logic Programming*, pages 250–264. Springer, 2009.
9. Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Clingo= asp+ control: Preliminary report. *arXiv preprint arXiv:1405.3694*, 2014.
10. Joao Filipe Rosado Gouvela. Protein design using answer set programming. *Master Dissertation Instituto superior tecnico Lisboa*, 2012.

11. Mark A. Hallen, Daniel A. Keedy, and Bruce R. Donald. Dead-end elimination with perturbations (deeper): A provable protein design algorithm with continuous sidechain and backbone flexibility. *Proteins: Structure, Function, and Bioinformatics*, 81(1):18–39, 2013.
12. Xiangqian Hu, Hao Hu, David N. Beratan, and Weitao Yang. A gradient-directed monte carlo approach for protein design. *Journal of Computational Chemistry*, 31(11):2164–2168, 2010.
13. David T Jones. De novo protein design using pairwise potentials and a genetic algorithm. *Protein Science*, 3(4):567–574, 1994.
14. Dunbrack RL. Jr and Karplus M. Backbone-dependent rotamer library for proteins application to side-chain prediction. *J. Mol. Biol.*, 230(2):543 – 574, 1993.
15. Kristian W. Kaufmann, Gordon H. Lemmon, Samuel L. DeLuca, Jonathan H. Sheehan, and Jens Meiler. Practically useful: What the rosetta protein modeling suite can do for you. *Biochemistry*, 49(14):2987–2998, 2010.
16. GG. Krivov, MV. Shapovalov, and RL. Dunbrack. Improved prediction of protein side-chain conformations with scwrl4. *Proteins: Struct., Funct., Bioinf.*, 77(4):778–795, 2009.
17. Javier Larrosa and Thomas Schiex. Solving weighted {CSP} by maintaining arc consistency. *Artif. Intel.*, 159(12):1 – 26, 2004.
18. Andrew R Leach, Andrew P Lemon, et al. Exploring the conformational space of protein side chains using dead-end elimination and the A* algorithm. *Proteins Structure Function and Genetics*, 33(2):227–239, 1998.
19. A. Leaver-Fay, M. Tyka, S. M. Lewis, O. F. Lange, et al. Chapter 19 - rosetta3: An object-oriented software suite for the simulation and design of macromolecules. In Michael L. Johnson and Ludwig Brand, editors, *Computer Methods, Part C*, volume 487 of *Methods in Enzymology*, pages 545 – 574. Academic Press, 2011.
20. Vladimir Lifschitz. What is answer set programming? pages 1594–1597, 2008.
21. Vincent JJ Martin, Douglas J Pitera, Sydnor T Withers, Jack D Newman, and Jay D Keasling. Engineering a mevalonate pathway in escherichia coli for production of terpenoids. *Nature biotechnology*, 21(7):796–802, 2003.
22. Noah Ollikainen, Ellen Sentovich, Carlos Coelho, Andreas Kuehlmann, and Tanja Kortemme. Sat-based protein design. In *ICCAD*, pages 128–135. IEEE, 2009.
23. S. Peisajovich and D. Tawfik. Protein engineers turned evolutionists. *Nature methods*, 4(12):991–994, 2007.
24. Niles A. Pierce and Erik Winfree. Protein design is np-hard. *Protein Engineering*, 15(10):779–782, 2002.
25. Jürgen Pleiss. Protein design in metabolic engineering and synthetic biology. *Current opinion in biotechnology*, 22(5):611–617, 2011.
26. HK. Privett, G. Kiss, TM. Lee, R. Blomberg, RA. Chica, LM. Thomas, D. Hilvert, KN. Houk, and SL. Mayo. Iterative approach to computational enzyme design. *Proc. Natl. Acad. Sci. U. S. A.*, 109(10):3790–3795, 2012.
27. Luis P.B. Scott, Jorge Chahine, and Jos R. Ruggiero. Using genetic algorithm to design protein sequence. *Appl.Math.Comput.*, 200(1):1 – 9, 2008.
28. Maxim V. Shapovalov and Roland L. Dunbrack Jr. A smoothed backbone-dependent rotamer library for proteins derived from adaptive kernel density estimates and regressions. *Structure*, 19(6):844 – 858, 2011.
29. S. Traoré, D. Allouche, I. André, S. de Givry, G. Katsirelos, T. Schiex, and S. Barbe. A new framework for computational protein design through cost function network optimization. *Bioinformatics*, 29(17):2129–36, 2013.
30. Yushan Zhu. Mixed-integer linear programming algorithm for a computational protein design problem. *Ind. Eng. Chem. Res.*, 46(3):839–845, 2007.