Protein design: a NP-hard problem in bioinformatics

Hugo Bazille

Research Master Internship

Master Thesis Report

Protein design: a NP-hard problem in bioinformatics

Author:
Hugo Bazille

Supervisor:
Jacques Nicolas
DYLISS team

Abstract

Proteins are biological macromolecules made of a chain of simple molecules called amino-acids. The three-dimensional folding of a protein greatly determines its function. Due to the combinatorial nature of the space of possible protein spatial conformations, computer-aided protein study is a major research field in bioinformatics. The problem of *computational protein design* aims at finding the best protein conformation to perform a given task. In this internship, this problem is reduced to an optimization problem, looking for the minimization of an energy function depending on the amino-acid interactions in the protein.

The goal of this internship is to develop a new solver for this problem based on Answer Set Programming. ASP is a paradigm of declarative programming based on non monotonic reasoning. The computational protein design problem may be easily modeled as an ASP program but a practical implementation able to work on real-sized instances has never been achieved. This report presents the state-of-the-art approaches in the domain (*A*, ILP, weighted-CSP...*) and proposes an in-depth study in ASP that shows opportunities and pitfalls of this approach. We 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 outperforms a previous ASP implementation and has comparable performances with respect to SAT-based approaches.

Contents

1 Introduction

Proteins are essential compounds of living organisms. Indeed, they are implied in almost every structural, catalytic, sensory, and regulatory functions of these organisms [7]. Proteins are amino-acid sequences that may have many different functions, which are mostly determined by their three dimensional structure. The study of these structures is thus an important field in biology with many applications in various fields such as medicine, biotechnology, synthetic biology... [19, 36]. Computer-assisted study of proteins leads to new possibilities: instead of "merely" understanding the system functioning, it offers opportunities to mimic the evolution and create new mutations in proteins or even totally new structures [34].

The goal of *computational protein design (CPD)* is precisely to find among a collection of proteins the one that are most likely to match a function. As they are twenty possible amino-acids for each position in a protein sequence and each of them accepts several structural variants, the number of possible combinations to be tested is out of range of any experimental approach, even for very short sequences. Consequently, a significant amount of efforts has been put into computational studies of proteins, and numerous works have demonstrated the power of computer-aided protein design [1, 8, 20, 21, 26, 33]. Among the most striking results of this approach, one can mention the design of retro-aldol enzymes [22], or the production of antimalarial drugs from the engineered bacteria *Escherichia coli* [31].

In CPD, choosing the best sequences of amino-acids to perform some function is formulated as an optimization problem. It can be seen as the dual problem of another standard problem in bioinformatics, protein, which tries to find the structure (and then the function) of a protein from a given a sequence of amino-acids. 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. Better techniques are needed in order to improve the exploration of the search space and to enable more realistic models. 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.

The goal of this internship was to develop a specific approach to solve the CPD problem. It uses a form of non monotonic logic: Answer Set Programming [30]. This work is based on the previous results of [18], with many enrichments that are developed all along this internship report. In order to evaluate this work, two comparison studies were carried out: the first one quantifies the progress made during this internship with respect to results of [18], and the second one is a more demanding task addressing the results of the most recent advances in the domain [1, 2, 40]. The ASP model performances do not reach the level of the best current approaches and a discussion on the advantages and drawbacks of the ASP approach for solving the CPD problem is provided.

We begin by giving the biological background of this work and defining the CPD problem with a rigid backbone and discrete sets of possible rotamers in proteins. Then, we give an overview of state of the art, with various techniques and different paradigms. Next, we introduce Answer Set Programming in section 4: it details the basics of non monotonic reasoning and how to encode the CPD problem. Then, section 5 describes the algorithms and encodings used in the different steps of the search. Results are described in section 6, together with the analysis of the different encodings. Finally, the reports concludes by different perspectives that seem interesting to be studied in a future work.

2 The protein design problem

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 bounds*. There are mainly 20 different amino acids that follow the same organisation illustrated in figure 1. Each amino acid has an amine (NH_2) and a carboxyl group (COOH). The 'R' represents the *side-chain* that makes the amino acid unique. An amino acid with a given side chain is called a residue. The sequence of carbon, oxygen and nitrogen atoms without the side chain defines a three-dimensional structure called *backbone*.

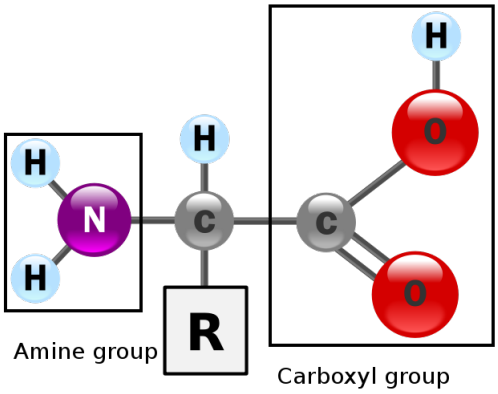


Figure 1: Structure of an amino acid

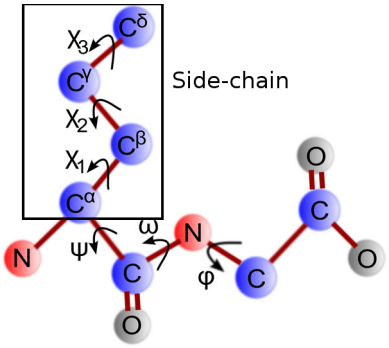


Figure 2: Dihedral angles in the backbone and in the side chain[2]

The geometry of the protein is relatively rigid and is generally defined by three *dihedral angles* (φ, ψ and ω) for each amino acid of the sequence along the backbone. A dihedral angle is the angle between two planes. Here, these two planes are defined by a sequence of four atoms and dihedral angles represent the angles between two amino acids. Furthermore, each side-chain has up to four degrees of freedom: the dihedral angles χ_1 to χ_4 (see figure 2).

The backbone structure is highly constrained. Indeed, four successive atoms form a plane, as shown in figure 3: it partly explains the rigidity of the backbone: there are fewer freedom degrees than one might expect. Some proteins also exhibit some symmetries, as shown in figure 4.

The different possible conformations of a side-chain are called *rotamers*. Whenever there is a degree of freedom, there are an infinite number of possible rotamers, as the molecules can continuously rotate around the axis. However, it is most often sufficient to consider a representative finite set of rotamers (see figure 5), which can be determined by a statistical analysis of the actual conformations present in protein structure databanks. For each amino acid, there are from one to

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

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

[3] `http://biowiki.ucdavis.edu/Biochemistry/Proteins/Protein_Conformation`

[4] `http://www.pdb.org/pdb/explore/images.do?structureId=2EGN`

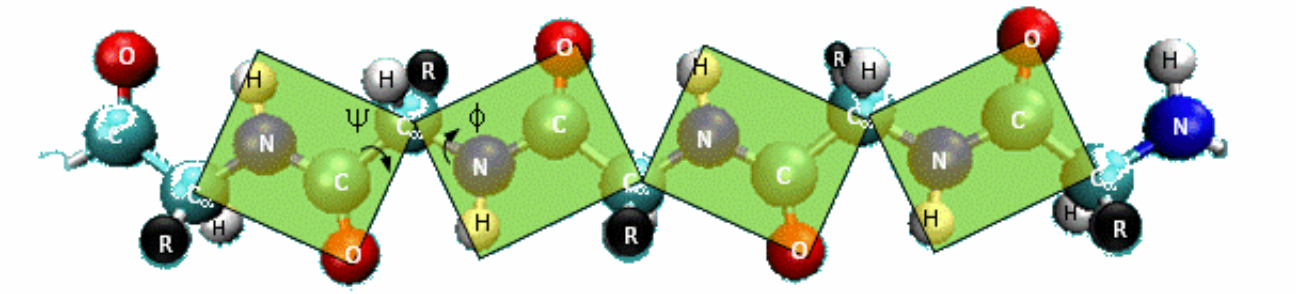


Figure 3: Topology of a backbone: a succession of planes[3]

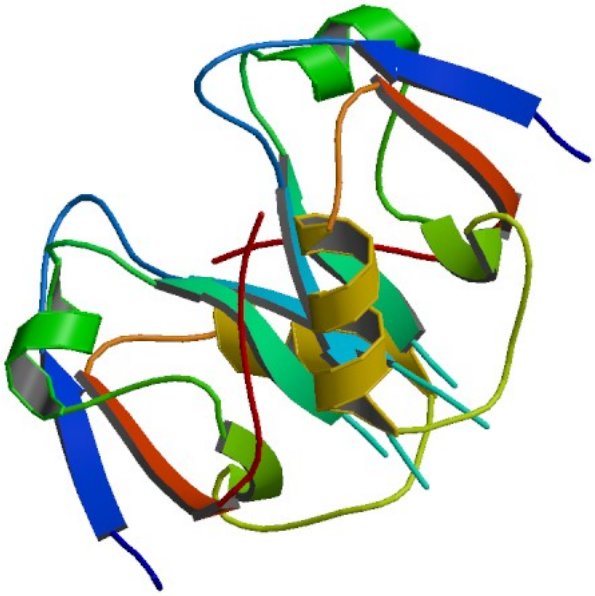


Figure 4: An abstract view of the 2EGN protein domain backbone structure[4]

several tens of most common rotamers. These common rotamers are listed in public libraries such as the Dunbrack backbone-dependant and backbone-independant rotamer libraries [24].

2.2 Protein design through energy minimization

The aim of computational protein design (CPD) is to find a protein that will perform a desired function. This function depends both on the backbone structure and on the rotamer configuration taken by each aminoacid. For a given backbone structure, the protein tends to adopt a global configuration of minimal energy, which leads to a more stable structure. Therefore, the practical goal of protein design is to find the conformation of the sequence of residues that folds into a defined backbone and minimizes the energy of the protein, which can be expressed as an optimization

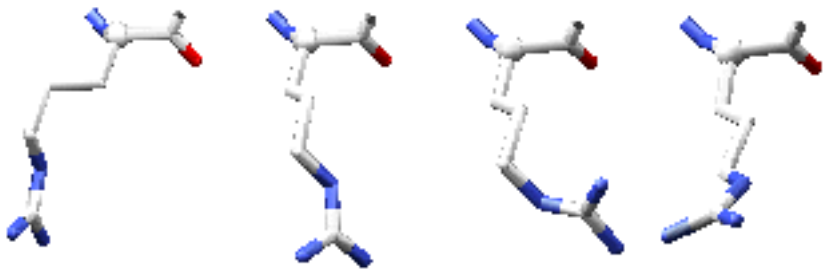


Figure 5: Different rotamers: the side chain has different possible geometries

problem.

The energy of the protein is given by the energy of the backbone, the energy of interaction between the rotamers and the backbone, and the energy of interaction between different rotamers. When the backbone is fixed, the energy minimization depends only on the energy of interaction between the rotamers and either the backbone or the other rotamers. 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 becomes:

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

The energy functions $E(i_r)$ and $E(i_r, j_{r'})$ are based on empirical measures. In practice, they depend on many parameters such as van der Waals potentials, electrostatics, internal coordinate terms [17]. Then, different energy functions may be chosen and lead to slightly different results.

Protein design is a NP-hard problem [35] 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 $(220)^{100} \approx 10^{234}$ possible solutions.

Variants of this problem are also studied, such as protein design with continuous rotamers [8, 20] or with backbone flexibility [16, 20]. The backbone flexibility can also be represented with a discrete or continuous space of possible conformations.

2.3 Benchmarking libraries

In any approach used to solve the protein design problem, benchmarking data are needed for parameter tuning and comparison purpose. Here we quickly describe two common protein databases used by most researchers in the domain.

Protein Data Bank Created in 1971, the Protein Data Bank (PDB)[1] is a repository for the three dimensional geometry of large biological molecules such as proteins. In PDB the structure description of thousands of proteins can be downloaded in different formats (PDB, XML...). These

[1] http://www.pdb.org

files include the sequences of amino acids and rotamers that form the protein and the three-dimensional coordinates of the residues.

PDB works collegially: everyone can submit a macro-molecule structure, however it has to be validated by a curator before being added to the database.

Backbone dependant rotamer library As numerous approaches use discrete sets of rotamers to solve the protein design problem, it is important to work with standard libraries of rotamers. The Dunbrack backbone-dependent rotamer library [24] developed in 1993 is one of the main libraries used in protein design. It is updated regularly and we have used the 2010 version[2] [39]. The goal of rotamer libraries is to find the most common rotamers for each amino-acid. They summarize the existing knowledge on the experimentally determined structures. Initially, rotamer libraries took only into account the nature of the amino-acid: they were backbone-independant rotamer libraries. However, this does not allow to consider the specificities of the studied protein. On the contrary, backbone dependant libraries take into account the different angles φ and ψ of the backbone and have been demonstrated to give more accurate results [24]. In these libraries, the backbone form is taken into account, leading to better prediction of interactions. To construct the libraries, several thousands of proteins are analyzed and probability densities are estimated as a function of the backbone angles φ and ψ. Figure 6 shows how the probabilities differ for different φ, ψ values and how the results are discretized according to the version of the library. Given a continuous density profile, discretization gives more precise results in the recent versions. To make the schema more understandable, only one angle of the side chain is represented in the figure (varying from -180 to 180.

Note that Dunbrack libraries are not the only existing ones. Among recent ones, one can cite the protein-dependent side chain rotamer library [4]: in order to calculate the possible rotamers, the library takes into account not only the angles of the backbone but also local space information such as position of other amino-acids. In theory, these libraries give more accurate results, however, since they are very recent, it is hard to find easily usable software using them.

3 State of the art

In this section, we introduce a few state of the art algorithms to solve this problem. The presentation is divided into two parts. In the first one, algorithms are allowed to find approximations of the best solution and are supposed to scale to larger proteins. In the second one, algorithms are designed to find exactly the best solution.

3.1 CPD with approximations

Genetic algorithms: Different approaches based on genetic algorithms have been tried on the CPD problem [38]. An initial population of rotamers is created. Each element of the population is a sequence of amino-acids with their conformations. A cross-over operation is defined by exchanging two amino-acids at a position in two different elements of the population. A mutation operation 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 cross-over operators on the remaining rotamers. Some mutations are randomly added in order to escape from the local

[2] http://dunbrack.fccc.edu/bbdep2010

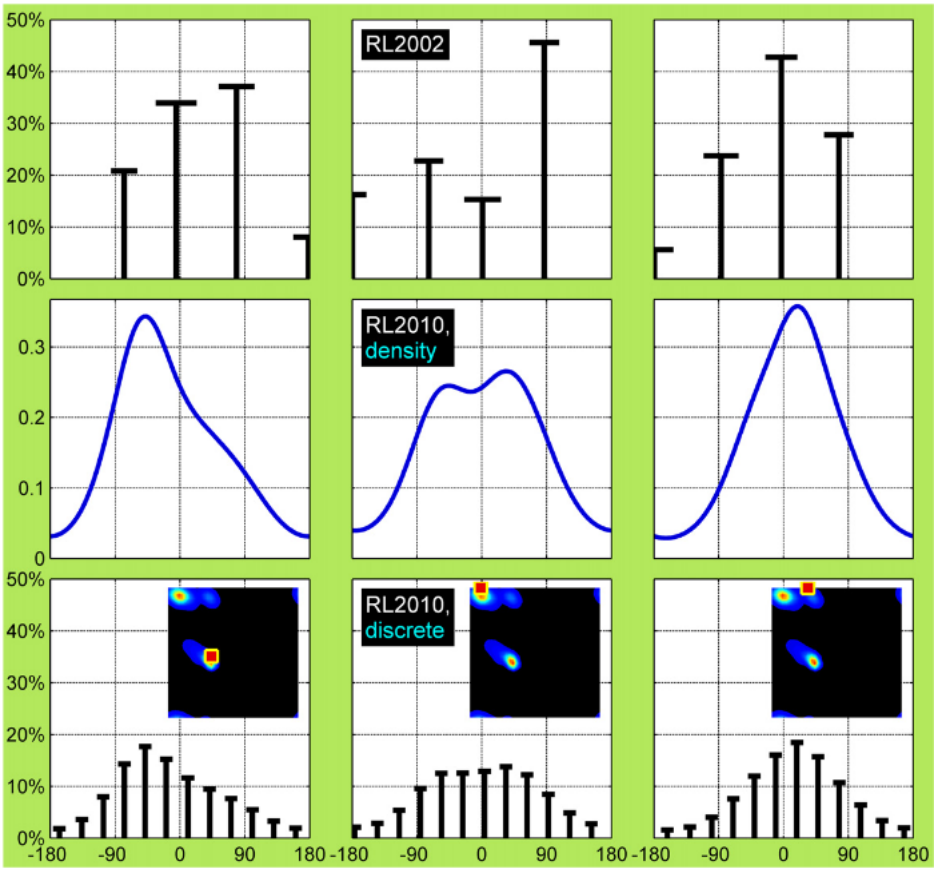


Figure 6: Probability density distribution in the 2010 version of the Dunbrack library and the corresponding discretization in the 2002 and 2010 versions for three (φ, ψ) values

minima. This knowledge-poor approach was one of the first to be applied on CPD [23]. However, this is not widely used anymore, since other methods such as Monte Carlo are preffered.

SCWRL4: SCWRL4 [26] 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 conformations (i.e. the geometry of the side chain) is varying. This assumption greatly reduces the search space. An interaction graph that represents the side-chain placement problem is created during the preprocessing. Then, the resolution algorithm is based on a tree decomposition algorithm: the graph is decomposed into trees on which a branch-and-bound search can be performed. It also includes a collision detection of rotamers, reducing the risk to produce physically impossible solutions.

Rosetta: Rosetta [29] is a large open source package that can be used to get an approximated solution to the protein design problem. It uses the Dunbrack backbone-dependant library to restrict the different possible conformations and uses a Monte-Carlo based method to sample the space of conformations [21, 25]. 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 was useful in our first series of experiments to compute the needed interaction energy values. Indeed, Rosetta offers many functionalities: it can also predict the interaction between two proteins or between a protein and a ligand, and the main function that interests us in this study is the prediction of energy of interaction and the possibility to give it as an input to another program.

3.2 Exact search algorithms for CPD

A*: 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 algorithms such as A* [28]. 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

$$E(i_r) + \sum_{j_s \text{ assigned}} E(i_r, j_s) + \sum_{k \text{ unassigned}} \min_{r'} E(i_r, k'_r)$$

. Heuristics have been improved many times since then and the interested reader may consult [20] to look at the current results achieved by such methods.

Optimization as an integer linear program: The problem of minimizing the total free energy of a protein conformation can be represented as a Linear Integer Program. Indeed, one can use binary variables to represent the presence of rotamers: $q_i(r_i) = 1$ if the rotamer r is chosen at place i, else $q_i(r_i) = 0$. The fact that there is exactly one rotamer at each position can be expressed by the formula

$$\sum_i q_i(r_i) = 1$$

The energy minimization is then represented by a constraint

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

Where E denotes the energy scoring function. This approach is extended in some works such as [41] where only some variables are constrained to be integers (Mixed Integer Linear Programming).

SAT based protein design: The protein design problem can also be formulated as a satifiability 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 then to have very good performances since this domain is very competitive and benefits from a strong and active community regularly enhancing the solvers. Effective encodings with improved branch and bounds algorithms are presented in [33].

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 (pairs that do not allow a minimal global energy). In practice,

hierarchical decisions are made: first the amino acid is assigned, then the different dihedral angles are chosen (i.e the different configurations of the side chain). These programs are quite efficient, provided they take into account only three possibilities by dihedral angle (called *gauche*$^+$, *gauche*$^-$ and *trans*).

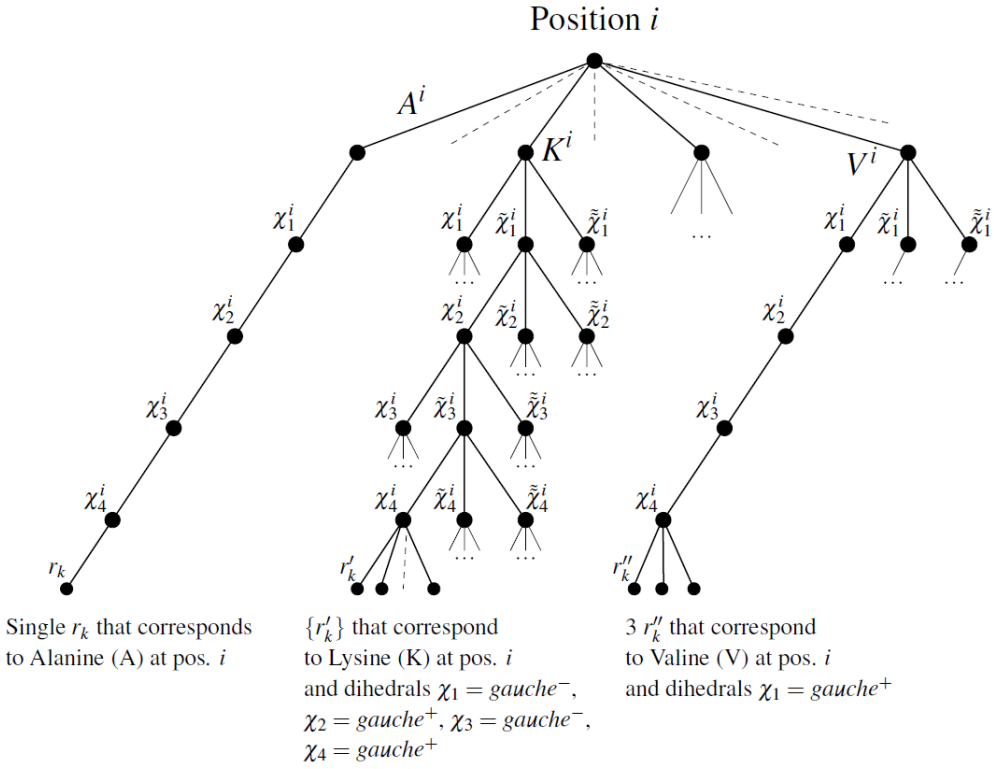


Figure 7: Search tree in ProSAT

Cost function networks: The most recent advance in solving the CPD problem proposes to model it as a weighted constraint satisfaction problem [1, 2, 40], using cost function networks.

Definition 3.1. Cost function networks

A Cost Function Network (CFN) is a pair (X, W) where $X = \{1, \ldots, n\}$ 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 that can be assigned to it. 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. For a given tuple of values t, $t[S]$ denotes the projection of t over S. A cost function $w_S \in W$, with scope $S \subseteq X$, is a function $w_S : D_S \to [0, k]$ 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 between a rotamer and the fixed backbone, and each binary cost function $w(i_r, j_{r'})$

represents the energy of interaction between two rotamers. The cost functions with higher arity are fixed to 0 since the CPD problem only includes binary interactions.

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.

This review cannot pretend to be exhaustive but it provides already a good overview of the main approaches that have been used so far to solve the CPD problem. In this internship, we have studied another framework that has been tried only once in the literature with little success, Answer Set Programming (ASP). Since ASP has exhibited interesting performances in a number of combinatorial optimization problems, we have tried to understand the reasons of this relative failure. We have emphasized the specific features of the problem that make it difficult and looked at better encodings, achieving clear progress with respect to previous work.

4 Answer Set Programming (ASP)

4.1 A quick presentation of ASP

4.1.1 Programming paradigm

Answer Set Programming (ASP) is a form of declarative programming used for combinatorial search problems. An ASP program consists of logical rules which look like Prolog rules, but with a different semantics.

The solutions of an ASP program are computed under the stable models semantics[15]: a stable model can be seen as a classical logical model (a set of true atoms making the program true) in which an atom is true if and only if it can be proven from the rules. ASP is designed not only for Boolean Constraints Solving but also for complex optimization: solvers for ASP programs implement heuristics in order to solve efficiently minimization queries over weighted atoms.

4.1.2 Elements of syntax and semantics

The syntax adopted in this document corresponds to the current standard of ASP language[3]. It differs substantially from the syntax of previous grounders such as Gringo 3.

Let us first introduce normal logic programs. Given a set of atoms A, a *normal logic program* over A is a finite set of normal rules:

$$a_0 \leftarrow a_1; \ldots; a_m; \text{not } a_{m+1}; \ldots; \text{not } a_n$$

where $a_i \in A$ for all i. A *normal rule* r is made of a *head* $head(r) = a_0$ and a *body* $body(r) = \{a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n\}$. Each atom can be provably true or false. This means that if an atom cannot be proven it is considered as false. The intuitive reading of a rule is that if $body(r)$ holds (ie $a_1 \ldots a_m$ are provably true and $a_{m+1} \ldots a_n$ are false) then $head(r)$ has to hold. If $body(r) = \emptyset$ then r is called a *fact*. It intuitively means that $head(r)$ always holds.

In the following, we will use $body^+(r) = \{a_1, \ldots, a_m\}$ and $body^-(r) = \{a_{m+1}, \ldots, a_n\}$. A rule is said to be positive if $body^-(r) = \emptyset$. A program is said to be positive iff all its rules are positive.

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

Definition 4.1. Model

A set $X \subseteq A$ is a model of a logic program P if for all rules $r \in P$, $head(r) \in X$ whenever $body^+(r) \subset X$ and $body^-(r) \cap X = \emptyset$.

For instance, the following program P_1 has three models -$\{p\}$, $\{p, q\}$ and $\{p, q, r\}$-:

$$P_1 = \left\{ \begin{array}{c} p. \\ q \leftarrow r. \end{array} \right\}$$

However, $\{p, q\}$ and $\{p, q, r\}$ are not provable: no rule has head r and thus r is not provable and the only rule with head q requires r to be provable. For a positive program, the unique model which has this property is the $\subseteq$-minimal model [14]. For P_1, it is $\{p\}$.

Definition 4.2. Stable model for a positive program

Given a positive program P, a stable model is the $\subseteq$-minimal model of P.

For a positive program, the stable model is unique [14].

For the following program P_2, the only stable model is $\emptyset$. Indeed, intuitively we can see that to prove p, we need to prove q, and to prove q, we need to prove p.

$$P_2 = \left\{ \begin{array}{c} p \leftarrow q. \\ q \leftarrow p. \end{array} \right\}$$

To compute the stable models for general programs, we need to introduce the reduct P^X of a program P relative to a set X of atom.

$$P^X = \{head(r) \leftarrow body^+(r) | body^-(r) \cap X = \emptyset\}$$

P^X is a positive program and then has a unique stable model.

Definition 4.3. Stable model for a normal program

Given P a program, and X a model of P, X is a stable model of P if X is the stable model of P^X[15].

For instance, let us consider the following program P_3:

$$P_3 = \left\{ \begin{array}{c} p. \\ q \leftarrow \text{not } r; \text{not } s. \\ r \leftarrow p; \text{not} q. \end{array} \right\}$$

This program has six models ($\{p, r\}$,$\{p, r, s\}$, $\{p, q\}$, $\{p, q, r\}$, $\{p, q, s\}$, $\{p, q, r, s\}$). Let us calculate the reduct over two models:

$$P_3^{\{p,r\}} = \left\{ \begin{array}{c} p. \\ r \leftarrow p. \end{array} \right\}$$

The only minimal model of $P_3^{\{p,r\}}$ is $\{p, r\}$, then $\{p, r\}$ is a stable model.

The reduct for $P_3^{\{p,q,r,s\}}$ is:

$$P_3^{\{p,q,r,s\}} = \left\{ \begin{array}{c} p. \end{array} \right\}$$

The only minimal model of $P_3^{\{p,q,r,s\}}$ is $\{p\}$, then $\{p, q, r, s\}$ is not a stable model. Following the same reasoning, one can prove that the only stable models of P_2 are $\{p, q\}$ and $\{p, r\}$.

A normal program can have one, several or no stable model.

4.1.3 Extensions

First order variables It is possible to use predicates of any arity and first order variables in ASP, provided that this 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. In practice, programs generally use facts to define domains.

For instance, the program:

```
node(1).  node(2).
edge(X,Y) :- node(X); node(Y).
```

yields the grounded program

```
node(1).  node(2).
edge(1,1) :- node(1); node(1).
edge(1,2) :- node(1); node(2).
edge(2,1) :- node(2); node(1).
edge(2,2) :- node(2); node(2).
```

Integrity constraints It is possible to write rules with no head. An integrity constraint c is a rule of the form

$$\emptyset \leftarrow a_1; \ldots; a_m;\ \text{not}\ a_{m+1}; \ldots;\ \text{not}\ a_n.$$

It means that $body(c)$ cannot hold simultaneously. This addition does not add expressive power: it can be expressed in normal logic by introducing a new atom *error* with the rule

$$error \leftarrow a_1; \ldots; a_m;\ \text{not}\ a_{m+1}; \ldots;\ \text{not}\ a_n;\ \text{not}\ error.$$

However, it allows the code to be cleaner: constraints can be more easily identified.

Cardinality rules, choice rules A cardinality rule is of the form

$$a_0 \leftarrow l\{a_1; \ldots; a_n; \text{not}\ a_{n+1}; \ldots; \text{not}\ a_m\}u$$

It allows to control the cardinality of sets of atom, with lower bound l and upper bound u. That means that the head has to hold if at least l and at most u literals of the body hold. It does not add expressivity, it can be simulated by a number of normal rules quadratic in m. However, it greatly helps to reduce the size of programs.

A choice rule is of the form

$$l\{a_1; \ldots; a_m\}u \leftarrow a_{m+1}; \ldots; a_n; \text{not}\ a_{n+1}; \ldots; \text{not}\ a_o$$

It means 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 rules quadratic in m.

Optimization statements In order to solve multi-criteria optimization problems, ASP systems include statements to express cost functions [12]: $\#minimize$ and $\#maximize$. The minimize statement has the following form (the maximize statement is similar):

$$\text{minimize}\{w_1@p_i : l_1 \ldots; w_n@p_n : l_n\}$$

Each w_i represents the weight of the literal l_i and p_i represent its priority level. The solver minimizes the sum of the weighted literals, beginning by the highest priority level.

Other extensions such as disjunctive logic programming add expressivity to the language, however they will not be useful here: ASP with the above extensions is powerful enough to capture the protein design problem.

4.1.4 ASP solvers

Most recent ASP solvers are based on enhancements of the *Davis-Putman-Logemann-Loveland* (*DPLL*) algorithm: they proceed to a search with backtracking on a binary tree representing all the possible assignments of the program atoms. Each node of the tree corresponds to a three-valued (true, false, unknown) interpretation.

The search tree is derived by two operations:

- *expand*: derives new knowledge thanks to the logical relations and detect conflicts;
- *choose*: makes one choice for an atom that has value *unknown*

These choices are made using different heuristics. In case of failure, the solver analyzes the conflict and comes back to the most recent choice that does not imply the conflict. Conflicts are also learned in order to avoid them in other branches of the search.

ASP solvers have other useful features, such as enumeration of all the solutions, intersection of all possible solutions...

4.2 The clingo4 environment: ASP+control

The first ASP solvers were quite "monolithic". Indeed, they followed a fixed two step process: first the grounding generated a finite propositional program. Then, a solver computed the stable models of this program. It was quite difficult to implement an iterative or interactive form of reasoning: no possibility to play with the different computed models at one step of an algorithm in the next step, no possibility to add or remove some information "on-the-fly". Specific controllers were developed to allow incremental reasoning (*iclingo* [11]) and reactive reasoning (*oclingo* [10]). However, these systems were still rigid. Control was always given to the system and users had only the possibility to describe the logic of their algorithm through the ASP program.

In the recent versions of the ASP system *clingo*, two scripting languages have been integrated with ASP in a common environment in order to achieve complex reasoning processes: *Python* and *Lua* [13]. On the declarative side, it becomes possible to defines procedures allowing to instantiate different logic programs with different parameters, which are differentiated by a directive **#program**. External rules may also be added/removed in a program thanks to the directive **#external**. In the following, we explain the basics of ASP scripting with the example of *Python*. This choice is arbitrary, as *Lua* provides the same possibilities. Scripting has been used in our work to combine the search for approximate solutions and local exploration.

Basics of scripting: ASP programs can be divided in different subprograms:

```
a(1).
#program acid(k).
b(k).
#program base.
a(2).
```

The subprogram acid takes one argument (k). *base*, in addition to rules in the scope of a base program gathers all rules not preceded by a **#program** directive.These subprograms can be grounded at any time in the process. Here is an example of script calling the previous ASP program with its output:

```
#script(python)
  def main(prg):
    prg.ground("base",[])
    prg.solve()
    prg.ground("acid",[42])
    prg.solve()
#end.

Solving...
Answer: 1
a(1) a(2)
Solving...
Answer: 1
a(1) a(2) b(42)
```

In the first part, only the program *base* has been grounded, leading to the answer set $\{a(1), a(2)\}$. In the second part, the program *acid* has been grounded with the parameter 42. Then, a rule has been added to the previous one, leading to the answer set $\{a(1), a(2), b(42)\}$.

External statements: Once a program has been grounded, it cannot be "ungrounded". Therefore, another mechanism is needed for volatile statements. This is especially needed for reactive and incremental solving. The directive **#external** provides flexible handling of atoms. The following program shows some possibilities of the **#external** directive.

```
a(1).
#external b(1).
#external c(X) : a(X).
control :- b(1), c(1).
```

And a script calling this program with its output:

```
#script (python)
from gringo import Fun

def main(prg):
    prg.ground("base", {})
    prg.assignExternal(Fun("b",[1]),True)
    prg.assignExternal(Fun("c",[1]),True)
    prg.solve()
    prg.releaseExternal(Fun("b",[1]))
    prg.solve()
#end.
```

```
Solving...
Answer: 1
a(1) b(1) c(1) control
Solving...
Answer: 1
a(1) c(1)
```

First, the program is grounded. Then, two external statements are invoked. The first one switches the truth value of $b(1)$ to *true*. The second one switches the truth value of $c(1)$ to *true only if* $a(1)$ is *true*. A first solve call is made, leading to the answer set $\{a(1), b(1), c(1), control\}$. Then, the external statement $b(1)$ is released and another solve call is made, leading to another answer set: $\{a(1), c(1)\}$.

Possibilities: Among other possibilities, it is possible with simple scripts [13] to simulate the behaviour of specific solvers, such as *iclingo* and *oclingo*. It also allows the user to easily design its program by incremental refinements: as the user knows some specific features of its problem, he may improve its code to solve the problem more efficiently.

It is also possible to produce high level interactive solving. A client-server ASP solving process encoded in Python is presented in [13]. It allows more reactive solving and then may be implemented on distant infrastructures.

5 Implementation

In this section, we describe the different algorithms developed and/or implemented during the internship. We also give the main elements of actual ASP encoding. The presentation follows the historical order of our work: first, algorithms using the DEE heuristics are presented, then a first naive ASP encoding and finally the enhancements that we have designed.

5.1 DEE

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. It is necessary to reduce the number of possibilities while remaining sure to keep the best solution in the remaining space. The goal of *Dead End Elimination* (*DEE*) algorithms is to remove the choice of some residue at some place because it always give worse results than other residues at this place. They can also be extended to eliminate pair of residues, triplets...

There are numerous DEE algorithms that have different complexity and efficiency. The efficiency is measured by the number of pruned residues (or pairs, triplets...). The difficulty is to manage the trade-off between pruning and solving times.

Usually, data are processed through several runs, starting with the simplest algorithms and continuing with the most complex ones. The description of a few state-of-the-art algorithms follows, together with a short analysis of their advantages and drawbacks.

Simple DEE: The first Dead End Elimination algorithm was presented in [6]. It was designed especially for the protein design problem in 1992. It relies on a simple principle: "if the best

possible case when i_r is chosen is worse than the worst case when i_t is chosen instead, then i_r can be eliminated". Formally, i_r can be eliminated if there exists some t that satisfies

$$E(i_r) + \sum_{j \neq i} \underset{u}{min} E(i_r, j_u) > E(i_t) + \sum_{j \neq i} \underset{u}{max} E(i_t, j_u),$$

where j is a position and u is a rotamer.

In our case, it means that if a rotamer i_r has a higher score (sum of interactions energies) in its best case than another rotamer i_t in its worst case, then i_r can be safely eliminated. The pseudo-code of the algorithm is presented in algorithm 11 in annex).

If n is the total number of rotamers and p the number of positions, the complexity of algorithm 11 to test one rotamer is $O(pn^2)$. Thus, to test all the rotamers at all positions, complexity rises up to $O(p^2n^3)$.

Goldstein DEE: The previous algorithm is quite simple but is not very efficient. A better criterion, Goldstein DEE, can be summed up by "if i_r contribution to a solution costs always more than i_t then i_r can be eliminated". Formally, i_r can be eliminated if:

$$\exists i_t, E(i_r) - E_{j \neq i} \underset{u}{min} [E(i_r, j_u) - E(i_t, j_u)] > 0 \tag{1}$$

Algorithm 1 Goldstein DEE

```
for all position i do
    for all possible rotamer r at position i do
        for all candidate rotamer t ≠ r at position i do
            X ← E(i_r) - E(i_t)                          ▷ store the energy difference
            for all position j ≠ i do
                for all possible rotamer u at position j do
                    Y ← min_u[E(i_r, j_u) − E(i_t, j_u)]   ▷ minimal energy difference
                end for
                X = X+Y                                  ▷ sum of minimum differences
            end for
            if X>0 then
                eliminate i_r and break
            end if
        end for
    end for
end for
```

The complexity of algorithm 1 is the same as the Simple DEE: $O(pn^2)$ to test one rotamer and $O(p^2n^3)$ to test all rotamers at all places. The Golstein criterion is stronger than the simple DEE criterion: the set of rotamers it prunes includes the set of rotamers pruned by simple DEE application.

Simple split: Simple split is a more complex DEE algorithm which relies on the idea that considering only one rotamer may not be sufficient to eliminate another one. The principle of this

algorithm is illustrated in figure 8: there are 3 possible rotamers at position i: i_r, i_{t_1} and i_{t_2}. The horizontal axis represents the conformation space for all other positions, i.e all the possible combinations of rotamers at position $j \neq i$. The curves represent the energy of the conformations. On the configurations on the left of the graph, i_{t_2} is better than i_r, and on the right, i_{t_1} is better. Thus i_r will never be the best. For each possible conformation, there exists an i_t giving a better result than i_r. Formally, i_r can be eliminated if

$$\exists k, \forall v, \exists i_t, E(i_r) - E(i_t) + \sum_{j \neq k} \min_u [E(i_r, j_u) - E(i_t, j_u))] + (E(i_r, k_v) - E(i_t, k_v)) > 0 \qquad (2)$$

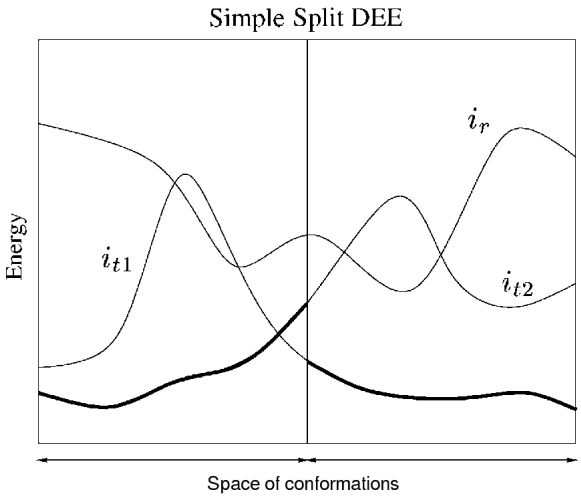


Figure 8: Principle of simple split DEE algorithm

The complexity of the algorithm 2 is $O(p^2n^3)$. This algorithm is then more interesting than the previous ones, as it has the same complexity and is more powerful. In practice, it eliminates more rotamers than the two previous ones, but it is a bit slower, due to its higher number of operations.

Double Goldstein: The Double Goldstein DEE shows how the different algorithms can be adapted to eliminate pair of rotamers. A pair $(i_{1_{r_1}}, i_{2_{r_2}})$ can be eliminated if there exists a pair $(i_{1_{t_1}}, i_{2_{t_2}})$ such that

$$\begin{aligned}(E(i_{1_{r_1}}) + E(i_{2_{r_2}}) + E(i_{1_{r_1}}, i_{2_{r_2}})) - (E(i_{1_{t_1}}) + E(i_{2_{t_2}}) + E(i_{1_{t_1}}, i_{2_{t_2}})) \\ + \sum_{j \neq i_1 j \neq i_2} \min_u (E(i_r, j_u) - E(i_t, j_u)) > 0 \end{aligned} \qquad (3)$$

The complexity of algorithm 12 (in annex) is $O(p^3n^5)$, a noticeable increase of complexity. In practice, this algorithm is launched after the previous one, in order to reduce the size of the input. Even if it allows to eliminate a high number of pairs, its complexity makes it too slow for many reasonable instances.

Algorithm 2 Simple split DEE

```
for all position i do
    for all possible rotamer r at position i do
        for all candidate rotamer t ≠ r at position i do
            for all position j ≠ i do
                for all possible rotamer u at position j do
                    store Y_{i,j,rt} = min_u[E(i_r, j_u) − E(i_t, j_u)]  ▷ store the minimal energy differences
                end for
            end for
        end for
        for all splitting position k do
            set elim_v = false ∀ rotamer v at position k
            for all candidate rotamer t ≠ r at position i do
                X ← E(i_r) − E(i_t)                                  ▷ store the energy difference
                for all position j ≠ i, j ≠ k do
                    X ← X + Y_{i,j,rt}                               ▷ sum of minimum differences
                end for
                for all rotamer v at position k do
                    if X + [E(i_r, k_v) − E(i_t, k_v)] > 0 then
                        elim_v ← true
                    end if
                end for
            end for
            if elim_v = true ∀v then
                eliminate i_r and break
            end if
        end for
    end for
end for
```

5.2 Search for exact results

The CPD problem can be specified in only a few lines of ASP. In this subsection, we present an encoding based on the one in [18]. This encoding does not include all possible optimizations but is quite effective on very short proteins.

First, the set of positions, of possible residues and possible rotamers at each position have to be defined. They are represented by facts using predicates `position/1` (4), `possibleResidue/2` (5) and `possibleRotamer/4` (6)(/n means with n arguments).

```
position(Position_in_the_protein).                                   (4)

possibleResidue(Position,Aminoacid_identifier).                      (5)

possibleRotamer(Position,Aminoacid_identifier,
Rotamer_identifier,Energy_backbone_interaction).                     (6)
```

Then the interaction energies between rotamers have to be tabulated from data given by software such as *Rosetta* or *Osprey*. This is coded using predicate `interEnergy/7` (7).

```
interEnergy(Position1,Aminoacid_ident1,Rotamer_ident1,
Position2,Aminoacid_ident2,Rotamer_ident2,Energy_interaction_12).
```
(7)

For each position, one residue is chosen to be in the solution. It is recorded in the predicate `residue/2` (8).

```
residue(Position,Aminoacid_identifier).
```
(8)

In the same way, one rotamer is chosen to be in the solution for each chosen residue. It is recorded in the predicate `rotamer/3` (9).

```
rotamer(Position,Aminoacid_identifier,Rotamer_identifier).
```
(9)

Given as input data the facts describing the sets of positions, possible rotamers and interaction energies, the CPD problem can be expressed in three rules.

Algorithm 3 `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{ Energy@1,energyB,Pos,Id:
           rotamer(Pos,Amin,Id),possibleRotamer(Pos,Id,Energy) ;
           Energy@1,energyR,Pos1,Pos2:
           rotamer(Pos1,Amin1,Id1),rotamer(Pos2,Amin2,Id2),
           interEnergy(Pos1,Amin1,Id1,Pos2,Amin2,Id2,Energy)}.
```

- Rule (a) states that at each position the solver has to choose a residue among the possible residues.
- Rule (b) states that for each chosen residue, the solver has to choose a rotamer among possible rotamers.
- Rule (c) minimizes the sum of the energies recorded in predicates `possibleRotamer` (energy of interaction between rotamers and the backbone) and `interEnergy` (energy of interaction between rotamers). 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 may be added to this program. An eliminated pair is taken into account by an integrity constraint:

```
:- rotamer(Pos1,Id1), rotamer(Pos2,Id2).
```

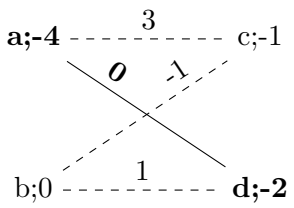
This rule means that the rotamers `Id1` at position `Pos1` and `Id2` at position `Pos2` cannot be chosen simultaneously in a solution.

This short program computes exact results but is limited to small instances as runtime may quickly become too long. The main goal of our internship was to understand the limitations and study how the code could be improved.

5.3 Transformation in an equivalent problem with better properties

5.3.1 Search for a lower bound

The optimization step is crucial in CPD and we have studied the way it is managed by clingo. 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$\{V : p\}$. with $V < 0$, the solver transforms it in #minimize$\{-V : \mathit{not}\ p\}$. Of course, this algorithm does not take into account the specificities of our problem. It appears that the transformation is far from being optimal. Let us consider a short example:



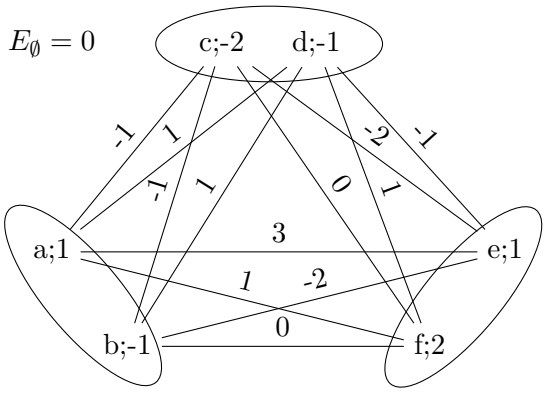
In this problem, it is possible to choose rotamer a or b at position 1 and rotamer c or d at position 2. The minimal solution of this problem is the pair (a, d), a configuration with total energy -6 (-4+-2+0). Solving this problem with (an adaptation of) the naive encoding given in section 5.2 leads 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:

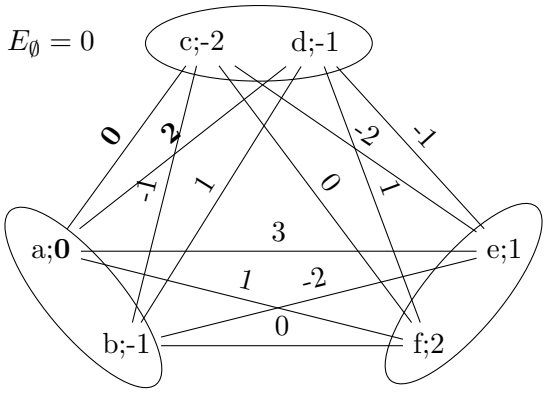
- 4 if a is **not** chosen
- 0 if b is chosen
- 1 if c is **not** chosen
- 2 if d is **not** chosen
- 3 if the pair (a, c) is chosen
- 0 if the pair (a, d) is chosen
- 1 if the pair (b, c) is **not** chosen
- 1 if the pair (b, d) is chosen

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 thousands of values may be added because the choice of a rotamer entails that all others are not chosen. It may be problematic: it is more difficult for the solver to find good lower bounds if there are many different values added at each assignment of a rotamer.

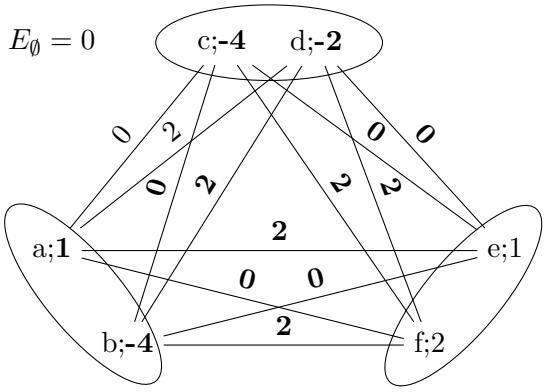
To bypass this problem, a transformation is made before giving the problem to the solver. Then, a much better lower bound is estimated and minimization is carried over only positive values. The principle is inspired from the transformations used in the cost function network approach [37, 5, 27]. The idea is the following one: for each position i, let $N_{i,1}, \ldots, N_{i,r}$ be the energy of interaction with the backbone of the possible rotamers $i_1, \ldots, i_r$, 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



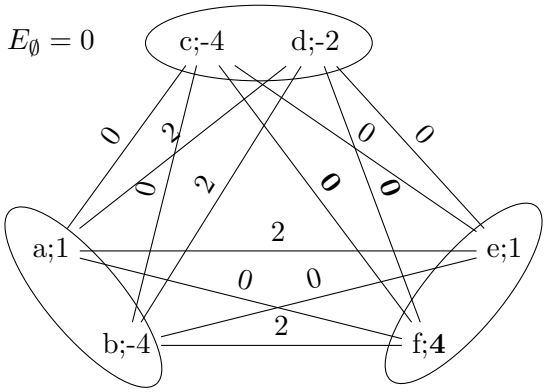
(a) Initial network



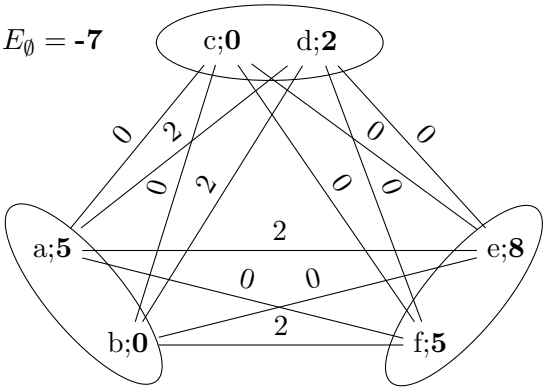
(b) Enforcing arc consistency for a



(c) Enforcing arc consistency for a, b, c, d



(d) Enforcing arc consistency for e, f



(e) Enforcing node consistency for all nodes

Figure 9: Transformation of a network into an equivalent one

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. It is further detailed in section 5.3.2.

For instance, the previous problem is such that the minimum energy of interaction with the backbone at position 1 is -4 and the minimum energy at position 2 is -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

$$possibleRotamer(Position, Aminoacid\_id, Rotamer\_id, Energy).$$

will be changed in

$$possibleRotamer(Position, Aminoacid\_id, Rotamer\_id, EnergyMinEnergy\_at\_Position).$$

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. In fact, the estimation of the lower bound may be stated as a constraint satisfaction problem and we are applying a technique imported from this field, constraint propagation to reduce the set of possible values for the minimum. This is achieved by looking for two local consistency properties, arc consistency, which is propagating energy constraints on pairs of rotamers, and node consistency, which is propagating energy constraints on a single rotamer with the backbone. A more complete example is provided in the figure 9. Initially, a network with three positions and two possible nodes by position is given. For each node there is an energy cost, and for each pair of nodes at different position there is another energy cost. First, enforcing arc consistency is applied between the node a and the nodes c and d (figure 9b): if a is chosen, exactly one of the two arcs (a, c) or (a, d) will be chosen. Then one unit of cost is transferred from a to its arcs going to position 2 such that both have a positive cost and one has a cost of zero. Then, the same principle is applied on the remaining arcs that concern a, and then with b, c and d (figure 9c). Next, the same is done with e and f: even if all arcs have been used in the algorithm with a,b,c,d, it does not ensure the arc consistency for e and f.

Finally, node consistency is enforced on all nodes: for each position, the minimal cost is added to $E_\emptyset$ and subtracted to the nodes, leading to a lower bound of -7.

5.3.2 Add a hierarchy to the encoding

In a branch and bound search, the sooner a cut can be made, the better it is. Then, we may transform the encoding in order to be able to do more cuts sooner. The idea is to cluster rotamers at some position. In any partition, rotamers must have similar properties in order to make it more effective. Fortunately, there is a natural way to do this clustering: regroup the rotamers by amino-acid: even if they do not have exactly the same properties, they have in general similar energies of interaction with other rotamers and with the backbone. Then, this partitioning can be made at a constant computational cost.

Let i be a position, $i_{A_1} \ldots i_{A_r}$ be rotamers that are different conformations of an amino-acid A, $E(A_1) \ldots E(A_r)$ be their energies of interaction with the backbone, and N be min $E(A_1) \ldots E(A_r)$.

Then we associate the energy N to the choice of A and the energies $E(A_1) - N \dots E(A_r) - N$ to the choice of $i_{A_1} \dots i_{A_r}$. These new energies are denoted $E'(A_1) \dots E'(A_r)$. Then, some of the energy is added at the choice of the amino-acid. It is denoted $E'(A)$. It allows to perform cuts on amino-acids and then to avoid to make r cuts successively.

The same principle may be applied for pairs of rotamers. For each amino-acid A at position i and B at position j, it is possible to define $(i_A, j_B) = \underset{r,r'}{argmin} E(i_r, j'_r)$ where r is a conformation of amino-acid A and r' is a conformation of amino-acid B. Then, in the minimization process, the energies $E(i_r, j'_r)$ are replaced by the new energies $E'(i_A, j_B)$ and $E'(i_r, j'_r) = E(i_r, j'_r) - E(i_A, j_B)$. Therefore, when two amino-acids are chosen at two different positions, it is possible to add energy in the objective function before choosing the rotamers. Then, the formula to be minimized becomes

$$E = \sum_i (E'(A) + E'(i_{A_r})) + \sum_i \sum_{j,j<i} (E'(i_A, j_B) + E'(i_{A_r}, j_{B_{r'}}))$$

The algorithms 4, 5, 6, 7, 4, 8 and 9 describe transformations of the problem that keep it equivalent. They correspond to the different steps described in the section 5.3.1 but with an adaption for the hierarchical encoding. Algorithm 10 describes how to improve the lower bound. The idea is that the lower bound is successively improved as much as possible for every position. Its complexity is $O(p^2 n^3)$ (with p the number of position and n the number of rotamers by position). Thus, it has the same complexity that most used DEE algorithms. More details on arc consistency enforcing algoirthms may be found in [3].

Algorithm 4 Energy transfer from residue to rotamers

Input: position i and amino-acid i_A
for all rotamers A_r at position i **do**
  $E(i_{A_r}) \leftarrow E(i_{A_r}) + E(i_A)$
end for
$E(i_A) \leftarrow 0$

Algorithm 5 Energy transfer from rotamer to pairs

Input: positions i, j, rotamer i_{A_r}
for all rotamers $B_{r'}$ at position j **do**
  $E(i_{A_r}, j_{B_{r'}}) \leftarrow E(i_{A_r}, j_{B_{r'}}) + E(i_{A_r})$
end for
$E(i_{A_r}) \leftarrow 0$

5.4 Looking for solutions close to the optimum

Finding the optimal solution is not the only point of the CPD problem. As said before, it may be interesting to produce solutions that are close to the optimum. We have studied two different objectives in this context, either looking for a speed-up of the search by a heuristic approach that offers no guarantee on the approximation of the minimum, or looking for a set of solutions with a controlled degradation with respect to the optimal solution.

Algorithm 6 Energy transfer from rotamers to a residue

Input: position i and amino-acid i_A
$E \leftarrow \min E(A_r)$
for all rotamers A_r at position i **do**
  $E(i_{A_r}) \leftarrow E(i_{A_r}) - E$
end for
$E(i_A) \leftarrow E$

Algorithm 7 Energy transfer from pairs to rotamers

Input: position i, rotamer i_{A_r}
for all position j **do**
  $E \leftarrow \min E(i_{A_r}, j_{B_{r'}})$
  for all rotamers $B_{r'}$ at position j **do**
    $E(i_{A_r}, j_{B_{r'}}) \leftarrow E(i_{A_r}, j_{B_{r'}}) - E$
  end for
  $E(i_{A_r}) \leftarrow E(i_{A_r}) + E$
end for

Algorithm 8 Energy transfer from pairs of rotamers to pairs of residues

Input: positions i, j, amino-acids i_A, j_B
$E \leftarrow \min E(i_{A_r}, j_{B_{r'}})$
for all rotamers A_r at position i **do**
  for all rotamers $B_{r'}$ at position j **do**
    $E(i_{A_r}, j_{B_{r'}}) \leftarrow E(i_{A_r}, j_{B_{r'}}) - E$
  end for
end for
$E(i_A, j_B) \leftarrow E(i_A, j_B) + E$

Algorithm 9 Energy transfer from residues to the lower bound

Input: positions i, lower bound $E_\emptyset$
$E \leftarrow \min E(i_A)$
for all amino-acids A at position i **do**
  $E(i_A) \leftarrow E(i_A) - E$
end for
$E_\emptyset \leftarrow E_\emptyset + E$

5.4.1 Approximations

The main issue in combinatorial problems is to represent and contain within reasonable limits the huge space of possible solutions. The idea of the DEE algorithms is to reduce the size of this space without removing the best solution(s). Ultimately, the cost of this strategy may equals the cost of solving directly the initial problem. To go further, it is necessary to drop the strong commitment to reach the optimum.

We have tried to find some constraints powerful enough to eliminate a high number of possibili-

Algorithm 10 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
                Energy_transfer_from _residue_to_rotamers(j,B)
            end for
            for all rotamer r' at position j do
                Energy_transfer_from _rotamer_to_pairs(j,r',i)
            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
        Energy_transfer_from_residues_to_lower_bound(i,A)
    end for
end for
```

ties but sufficiently well chosen to avoid loosing too much quality. The main idea in our encoding is to implement local searches. It proceeds by building a first model without requiring the minimum to be reached. Then, the search space is limited to a "small" neighborhood: all possible solutions that differ by at most k different rotamers are tried, with k small (say maximum 3). Once the optimal solution in this neighborhood has been found, a new search centered on this last solution is launched, and the whole process is repeated until no better solution can be found: this means that a local optimum has been reached.

Even if this method does not guarantee optimality, it allows to get an upper bound of the optimal value. With the transformations of the problem explained in section 5.3.1, we only deal with positive values. Having a good upper bound should allow to perform cuts quicker in the search.

5.4.2 Enumeration of ε-solutions

As we work on an abstraction of a biological problem, some approximations are made and some solutions of the optimization problem may not be solutions of the protein design problem. It may be because of space limitations and some rotamers need too much space, or for chemical incompatibility... For these reasons, it can be more useful to have a set of good solutions rather

than one global minimum solution.

The first step is to find a minimum. It may be an exact minimum (or an approximated one) E_{best} with the previous methods. After obtaining this minimum, it is necessary to restart the algorithm, beginning with a modified DEE. Indeed, in order not to eliminate ε solutions, the right members of the equations (1), (2) and (3) have to be changed by ε.

It is also possible to adapt DEE algorithms for a rotamer i_r to be eliminated if a lower bound of the energy $E_{lower}(i_r)$ when i_r is chosen is such that $E_{lower}(i_r) > E_{best} + \varepsilon$. As for all DEE algorithms, the cost of obtaining lower bounds increase when the required precision increased. 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 a complexity of $O(n^3p^3)$ with n the total number of rotamers and p the number of positions. Similar criteria may be used to eliminate pairs, triplets... of rotamers. The complexity to compute E_{lower} for pairs with this criterion is $O(n^5p^4)$.

After pruning the search space, the search itself can be launched. An ASP program doing this task may be constituted of the rules `(a)`, `(b)`, and `(c)` of the program `Exact encoding` with the additional rule:

```
:- freeenergy #sum{Energy,Pos,Id :
                  energy(Pos,Id,Pos,Id,Energy), possibleRotamer(Pos,Id,Energy) ;
                    Energy,Pos1,Id1,Pos2,Id2 :
                  energy(Pos1,Id1,Pos2,Id2,Energy), iE(Pos1,Id1,Pos2,Id2,Energy)}.
```

This rule states that the sum of energies must not be greater than `freeenergy`. In the program, `freeenergy` has to be replaced by $E_{best} + \varepsilon$.

Give ε values around 1 or 2% of E_{best} increases the number of solutions to several hundred or even thousands. Although we have had no time left to study in depth the results on our benchmarks, it might be interesting to get various statistics on the set of solutions, for instance to compare the frequence of apparition of rotamers and amino acids.

6 Results

Benchmarks have been run on two different datasets: the first one is extracted from the one used by [18], which contained initially 120 instances. They are based on 10 proteins (1BE9, 1I92, 1MFG, 1N7F, 1QAU, 1RZX, 1TP3, 2EGN, 2FNE, 2GZV). For each of these proteins, twelve instances of the problem have been generated, considering 10, 12, 15 and 17 positions and three different sets of amino-acids (fixed, only hydrophobic or all aminoacids). We retained the 4O most difficult problems. It serves to compare the progress made between [18] 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. All our runs have been performed on a Intel Xeon W3520 quad-core, 2.66 GHz.

Protein	Positions	AAs	# Rotamers before	# Rotamers after	Pairs eliminated	Time
1BE9	17	All	2547	1720	270811	2919
		Hydro	529	130	1903	3.2
	15	All	2063	1023	122977	1063
		Hydro	435	56	270	1.3
1I92	17	All	2648	1728	288332	3262
		Hydro	593	186	4318	6.0
	15	All	2055	816	86194	806
		Hydro	455	93	1042	2.1
1MFG	17	All	2608	1798	299418	3145
		Hydro	560	188	3717	5.3
	15	All	1972	854	106835	759
		Hydro	410	74	485	1.3
1N7F	17	All	2548	1525	211868	2473
		Hydro	544	136	1811	4.2
	15	All	1934	648	59369	477
		Hydro	402	57	202	1.2
1QAU	17	All	2643	1776	288165	1598
		Hydro	562	196	4177	6.2
	15	All	2023	895	101761	1228
		Hydro	421	97	1021	1.8
1RZX	17	All	2722	1748	257158	3398
		Hydro	554	167	3044	4.5
	15	All	2121	941	94310	883
		Hydro	404	56	205	1.1
1TP3	17	All	2582	1732	282007	3145
		Hydro	536	142	2253	3.8
	15	All	2142	1037	135604	1057
		Hydro	447	63	409	1.7
2EGN	17	All	2548	1570	240733	2478
		Hydro	546	150	2516	3.7
	15	All	2005	804	78377	573
		Hydro	413	75	655	1.4
2FNE	17	All	2736	1909	325100	3927
		Hydro	577	197	4639	6.1
	15	All	2144	1060	132397	1158
		Hydro	438	98	1093	1.7
2GZV	17	All	2934	1991	348785	5243
		Hydro	613	189	3823	7.5
	15	All	2247	1122	132910	1340
		Hydro	456	87	696	2.1

Table 1: Results of DEE algorithms performed on the first series of instances

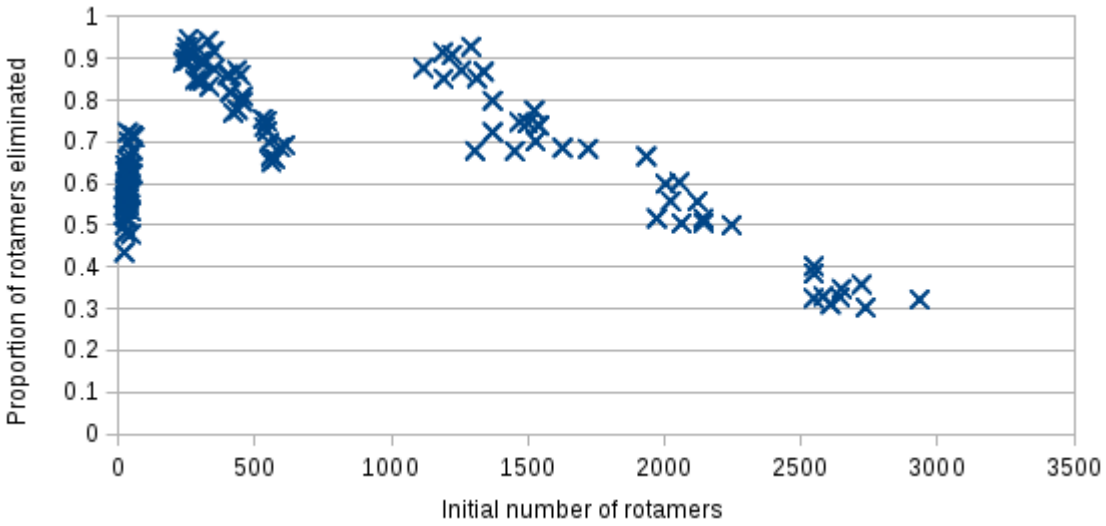


Figure 10: Bigger the instances are, smaller the proportion of eliminated rotamers is

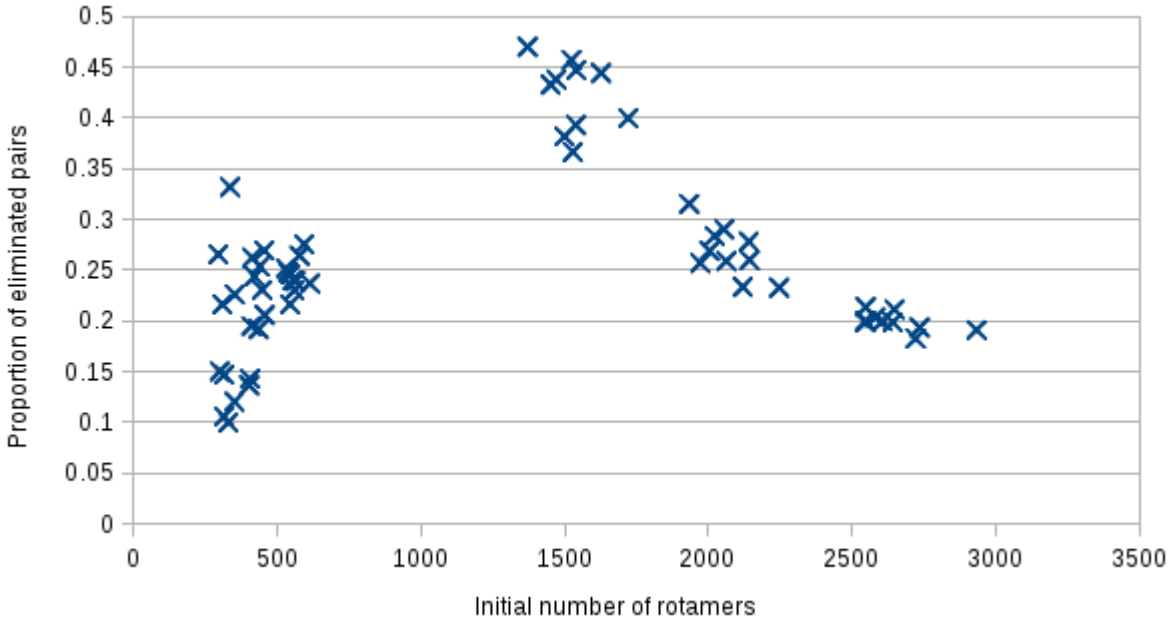


Figure 11: Bigger the instances are, smaller the proportion of eliminated pairs is

6.1 DEE

In order to evaluate the efficiency of DEE algorithms, 40 inputs are presented in table 1. For each instance, we perform three successive passes of Goldstein DEE and then 5 passes of Simple Split.

Several facts are interesting here:

- First, the percentage of eliminations decrease with the size of the instance, as shown in figure 10 representing the percentage of eliminations for various instances, sorted by the number of initials rotamers. The same phenomena applies for the elimination of pairs. Even if on small instances the proportion of eliminated pairs is small, we have a general tendency: bigger the instances are, smaller is the proportion asymptotically. This may be explained by the fact that there is a "forall" in the elimination criteria: adding possibilities leads to add more risk not to satisfy the "forall".

- The second fact is that elimination takes a long time on bigger instances: due to the cubic complexity, the time begins to explode for instances with all amino acids. Then, it is not possible to launch the double Goldstein DEE on bigger instances in a reasonable time: its complexity is too big to be executable where it could bring many information.

- The third one is that even if double Goldstein cannot be applied, the Simple Split DEE gives many results on the pairs. The sixth column gives the number of pairs eliminated.

6.2 Exact searches

The first series of results illustrates the progress made over [18]: table 2 contains the result with the same instances. For each amino acid, it shows the running time needed to solve the instances with 15 and 17 positions, with only hydrophobic amino acids and with the 20 amino acids. The total number of possible rotamers is indicated in the column #Rotamers. In order not to spend to much time on insoluble problems, a time limit (3hours) has been added in the specifications of the problem. A "-" denotes a problem that could not been solved before the time limit and a "!" denotes a problem that could not been solved due to insufficient memory.

We can observe the improvement on every instance: many more may be solved, and none crashes because of memory. Now, the gap is when there are around 1800 rotamers. Before, it was when there were more than 100 rotamers. Then, the difference is huge. It may be explained by the fact that with the new pre processing the ASP solver is able to infer a good lower bound and then dramatically reduce the search space.

The second series of instances is taken from [1]. They have been built using Osprey2.0 [9], with a different rotamer library than the first serie. These instances are far more complexes, as they deal with several tens of positions to redesign.

The second series of instances has a bigger difficulty, due to a different topology of the network. It may be explained by the fact that we use another rotamer library than for the first serie. Then, the threshold for instances that could not be solved is lower.

The solver *Toulbar2* presented in [1] has by far the best performances. It is the one that can solve the bigger number of instances. Some (not in this table) that could not be solved with Toulbar could not be solved with other solvers. However, we can compare to different methods. We wanted to compre with *Protsat*[4] [33], but the depot is empty and we could not contact the authors. Then, we compared to *maxhs*, another core-based SAT-solver, also used in benchmarks in [1]. *Maxhs* is not very effective: he could not solve many instances.

The most interesting comparison is with *Cplex*, a sophisticated solver for Integer Linear Programing. On some instances, our ASP-based solver outperforms *Cplex*, however on larger instances,

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

Protein	Positions	Amino acids	# Rotamers	Solving time in [18](s)	Solving time in our work(s)
1BE9	17	All	1720	!	-
		Hydro	130	-	0.66
	15	All	1023	!	155
		Hydro	56	1.719	0.05
1I92	17	All	1728	!	-
		Hydro	186	-	1.26
	15	All	816	!	1174
		Hydro	93	-	0.10
1MFG	17	All	1798	!	-
		Hydro	188	-	0.51
	15	All	854	!	336
		Hydro	74	679	0.07
1N7F	17	All	1525	!	-
		Hydro	136	-	0.21
	15	All	648	!	15.6
		Hydro	57	66	0.047
1QAU	17	All	1776	!	-
		Hydro	196	-	0.73
	15	All	895	!	98
		Hydro	97	2295	0.10
1RZX	17	All	1748	!	-
		Hydro	167	-	0.37
	15	All	941	!	906
		Hydro	56	36	0.05
1TP3	17	All	1732	!	-
		Hydro	142	-	0.19
	15	All	1037	!	403
		Hydro	63	463	0.054
2EGN	17	All	1570	!	-
		Hydro	150	-	0.32
	15	All	804	!	1038
		Hydro	75	280	0.039
2FNE	17	All	1909	!	-
		Hydro	197	-	0.49
	15	All	1060	!	154
		Hydro	98	4976	0.10
2GZV	17	All	1991	!	-
		Hydro	189	-	0.62
	15	All	1122	!	1242
		Hydro	87	-	0.09

Table 2: Summary of the advances in ASP protein design

Protein	Positions	# Rotamers	ASP(s)	Toulbar2(s)	Cplex(s)	MaxSat(s)
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 3: Comparison with the last tools in CPD

Cplex takes the advantage. It means that something in our approach makes it less scalable. The amelioration of our method goes through the identification of what makes it less scalable.

6.3 Search of approximated solutions

We did a search for an approximate result on 10 instances we could not solve. These 10 instances come from the first series. For each of them, there are 17 positions redesigned. First, we launch a solve during 20 seconds in order to have a model, and then we launch the local searches. In figure 12, y-axis represents the distance between the best model found and the lower bound calculated previously. X-axis represents the size of the local neighborhood, and 0 stands for the initial search that timed out at 10800 seconds in the previous subsection. Figure 13 represents the time of search in function of the size of the neighborhood for each of the 10 instances.

We notice that for all instances except one, a succession of local searches with the possibility to change one amino-acid allows to significantly improve the accuracy of the model found, with a time altogether reasonable: between 200 and 400 seconds. Increase to a neighborhood of size 2 allows to significantly improve the quality of the solution. Furthermore, the time of search does not explode. However, increasing the size of the neighborhood to 3 does not ameliorate the result for 8 of the 10 instances and when it does, it was a really small improvement. Besides, the search time starts to explode.

Doing it with other unsolved instances (from the second series) gives the same kind of behaviours: doing local searches gives good improvements, but increasing the size of the neighborhood negates these advances. However, these results show that it can be a good idea to perform at first a succession of local search until lock in a local minima. Then, the upper bound found may be used to perform cut during the global search.

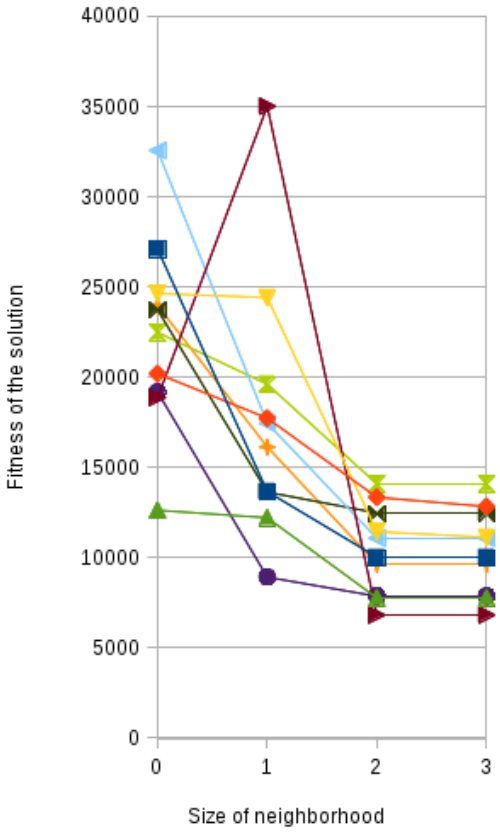


Figure 12: Accuracy of solutions depending on the size of the neighborhood

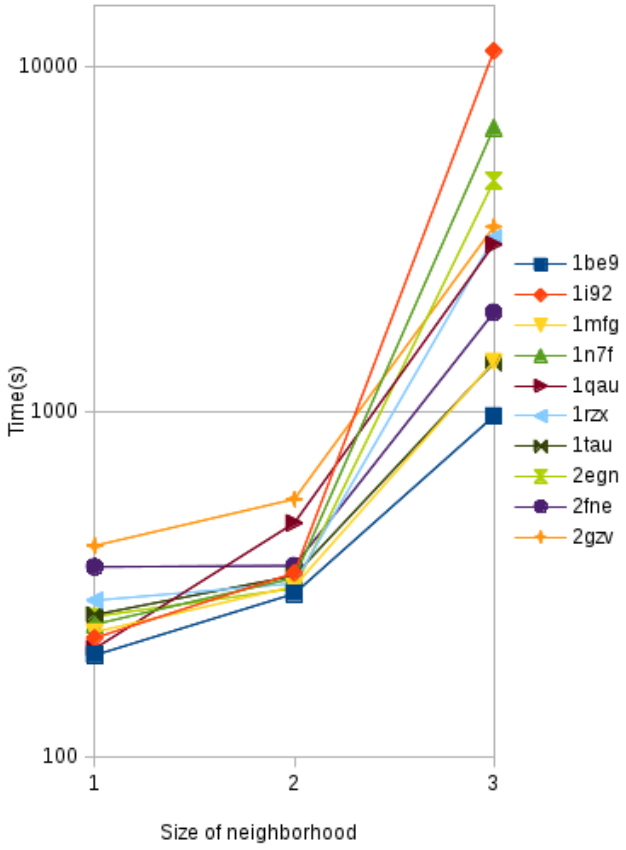


Figure 13: Solving time depending on the size of the neighborhood

6.4 Enumeration of neighbouring solutions

We did benchmarks on another instances from the first series. They were characterized by 17 positions to be redesigned and between 500 and 700 rotamers. All these instances could be solved to get an exact result. Then, for each instance, we have enumerated the solutions for ε ranging from 500 to 4500 with a step of 500. It represents about 0.5 to 8% of the free energy for these instances.

Figures 14 and 15 represent the number of solutions and the time needed to produce them all. We can see that the number of solutions grow exponentially with ε and so the running time does. However, the principal fact is that it is easy to have a big set of candidates proteins in a reasonable time if the optimal solution is is easily found. However, on bigger instances (that could not/harder be solved), the search for a set of proteins takes a very long time.

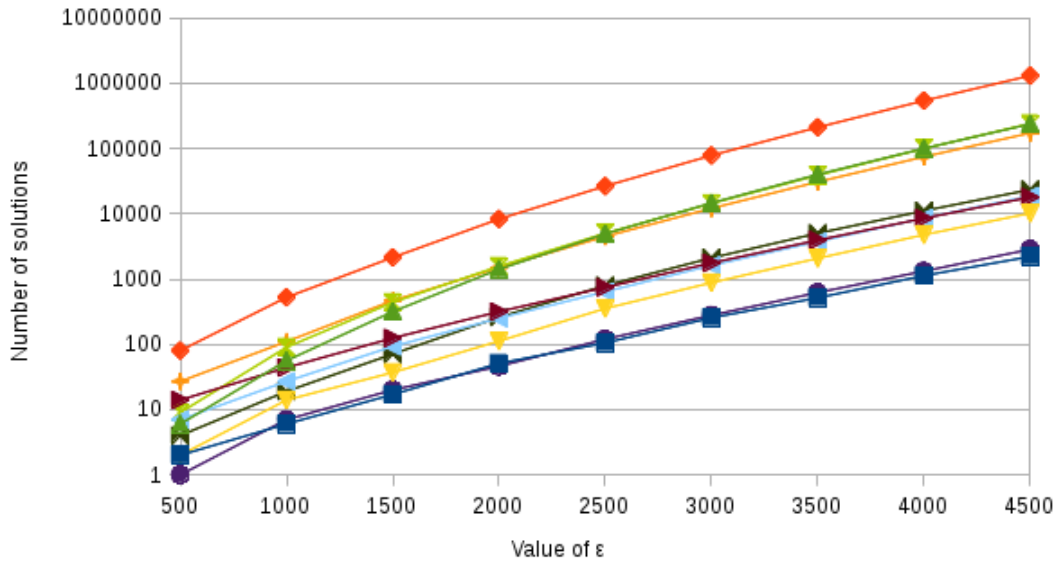


Figure 14: Number of solutions depending on ε

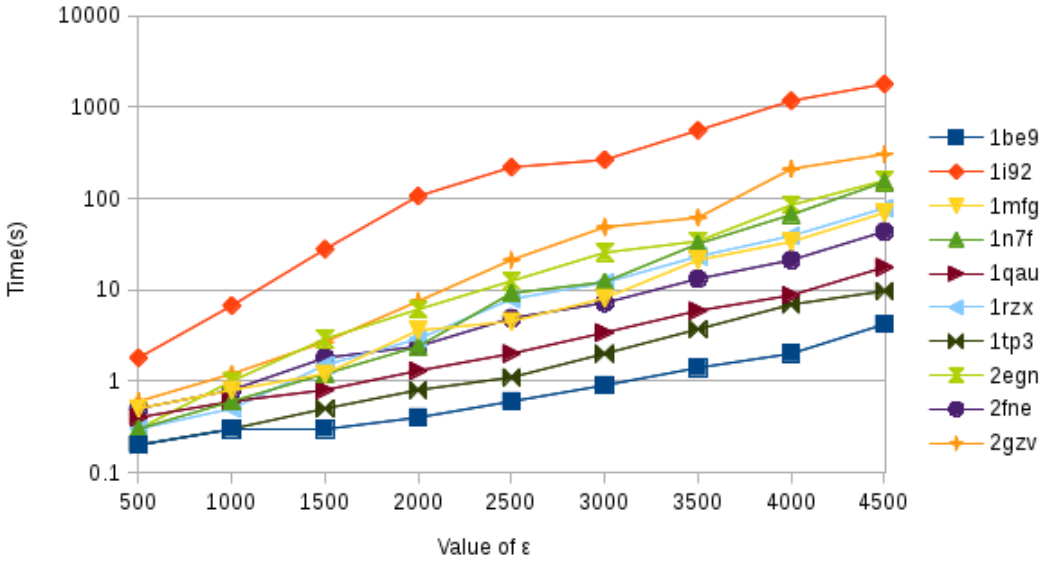


Figure 15: Solving time depending on ε

6.5 Explaining the differences

The first problem is the search for a lower bound. It was partially solved by applying a few algorithms. However, it is not sufficient: on big instances, the lower bound is far to the optimal solution. In other frameworks such as *Toulbar2*, the lower bound found is very close to the optimal

solution. This is why many instances are solved in less than one second: due to the good lower bound, almost all rotamers are eliminated quickly during the search, and only few rotamers are left.

Another problem is that ASP is a kind of black box. Even if the user can parameterize a lot of features (such as the optimization strategy, the policy of restart...), it is not possible to implement some procedure, such as enforcing the node and arc consistencies each time a choice is made. Even if systems such as *Toulbar* are quite slow to explore the search space, they have for them a better preprocessing, and the possibility to interact more accurately with the solver.

Finally, we did not have enough time to study the influences of heuristics. They may dirige the search in a way that allows to find the optimal value quicker, and hen to perform more cut more quickly. Some systems previously mentioned use different heuristics that allow to accelerate the search.

7 Conclusion

The goal of this internship was to evaluate the potential of Answer Set Programming in solving the CPD problem. Indeed, CPD belongs to the category of difficult problems and its combinatorial nature falls into the scope of ASP [32]. Furthermore, we saw this problem can be easily expressed in ASP language. Indeed, every possible residue, rotamer and all interaction energies can be coded as input logical facts, and the search part, including minimization, takes only three rules.

Our work is in the scope of exact solutions search, and can be compared to many different techniques: ILP, 01QP, Markov Random Field, A*-based searches, Cost Functions Networks... We have chosen for the benchmarks some of the most recent and best approaches and tried to import some of their features to improve our solution.

We pointed some drawbacks in our approach and give some hints for a future work in the rest of this section, whether on the processing side (lower bound, hierarchy...) or from the solver side.

We have brought a first contribution by greatly improving a previous work on the subject, but we are confident that this work can be improved in many ways. Trying to reach performances of CFN constitutes 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, and also developed the enumeration of "good" solutions. Nevertheless, we did not have enough time to develop all the ideas we had, such as using the ASP solver heuristics, and even not enough time to study the search for lower bounds. It leaves many research avenues open.

We can also pinpoint that in practice the goal of CPD is not only to find one optimal solutions, but a set of good candidates: due to the approximations and the choice of energy function, the most stable structure may slightly differ from the computed optimum. Moreover, it appeared that some solutions may be so stable that they lack the flexibility needed to perform the task they were designed for. The extension to neighbouring solutions can be easily implemented in ASP by launching another solve process with an extended search after the search of the optimal solution finished, however this strategy shares the disadvantages of the 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. From a methodological point of view, we may also explore the modularity of the approach, the difficulty to go from a variant of the problem to another one, and the performances of ASP in these different variants. The most difficult challenge would be to solve the CPD problem with a continuous space of rotamers. At first sight, it seems out of the range of ASP solvable problems due to the necessity to manage finite domains.

Finally, we believe that a great interest in trying different approaches to solve problems such as CPD may lead to new general ideas to solve difficult problems. Moreover, evaluating the strengths and weaknesses of different kinds of competing solvers has the potential to ameliorate all of them by merging the best ideas of each approach. As an interesting by-product of this internship, we have identified some weaknesses of the *clingo*'s minimization process and we informed the authors about these drawbacks and the interest of lower bound estimation in case of negative values.

References

[1] D. Allouche and al. Computational protein design as an optimization problem. *Artificial Intelligence*, 2014.

[2] David Allouche, Seydou Traoré, Isabelle André, Simon de Givry, George Katsirelos, Sophie Barbe, and Thomas Schiex. Computational protein design as a cost function network optimization problem. *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming*, pages 840–849, 2012.

[3] Christian Bessire, Jean-Charles Rgin, Roland H.C. Yap, and Yuanlin Zhang. An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165(2):165 – 185, 2005.

[4] Md Shariful Bhuyan and Xin Gao. A protein-dependent side-chain rotamer library. *BMC Bioinformatics*, 12(Suppl 14):S10, 2011.

[5] Martin Cooper and Thomas Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154(12):199 – 227, 2004.

[6] Johan Desmet, Marc D. Maeyer, Bart Hazes, and Ignace Lasters. The dead-end elimination theorem and its use in protein side-chain positioning. *Nature*, 356(6369):539–542, April 1992.

[7] Alan Fersht. *Structure and mechanism in protein science: a guide to enzyme catalysis and protein folding*. Macmillan, 1999.

[8] Pablo Gainza, Kyle E. Roberts, and Bruce Randall Donald. Protein design using continuous rotamers. *PLoS Computational Biology*, 8(1), 2012.

[9] Pablo Gainza, Kyle E Roberts, Ivelin Georgiev, Ryan H Lilien, Daniel A Keedy, Cheng-Yu Chen, Faisal Reza, Amy C Anderson, David C Richardson, Jane S Richardson, et al. Osprey: protein design with ensembles, flexibility, and provable algorithms. *Methods in enzymology*, 523:87, 2013.

[10] M. Gebser, T. Grote, R. Kaminski, and T. Schaub. Reactive answer set programming. In J. Delgrande and W. Faber, editors, *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, volume 6645 of *Lecture Notes in Artificial Intelligence*, pages 54–66. Springer-Verlag, 2011.

[11] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele. Engineering an incremental ASP solver. In M. Garcia de la Banda and E. Pontelli, editors, *Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08)*, volume 5366 of *Lecture Notes in Computer Science*, pages 190–205. Springer-Verlag, 2008.

[12] 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.

[13] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Clingo= asp+ control: Preliminary report. *arXiv preprint arXiv:1405.3694*, 2014.

[14] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. pages 1070–1080. MIT Press, 1988.

[15] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.

[16] Ivelin Georgiev and Bruce Randall Donald. Dead-end elimination with backbone flexibility. In *ISMB/ECCB (Supplement of Bioinformatics)*, pages 185–194, 2007.

[17] D Benjamin Gordon, Shannon A Marshall, and Stephen L Mayot. Energy functions for protein design. *Current opinion in structural biology*, 9(4):509–513, 1999.

[18] Joao Filipe Rosado Gouvela. Protein design using answer set programming. *Master Dissertation Instituto superior tecnico Lisboa*, 2012.

[19] Ingo Grunwald, Klaus Rischka, Stefan M Kast, Thomas Scheibel, and Hendrik Bargel. Mimicking biopolymers on a molecular scale: nano (bio) technology based on engineered proteins. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367(1894):1727–1747, 2009.

[20] 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.

[21] 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.

[22] Lin Jiang, Eric A Althoff, Fernando R Clemente, Lindsey Doyle, Daniela Röthlisberger, Alexandre Zanghellini, Jasmine L Gallaher, Jamie L Betker, Fujie Tanaka, Carlos F Barbas, et al. De novo computational design of retro-aldol enzymes. *science*, 319(5868):1387–1391, 2008.

[23] David T Jones. De novo protein design using pairwise potentials and a genetic algorithm. *Protein Science*, 3(4):567–574, 1994.

[24] Dunbrack RL Jr and Karplus M. Backbone-dependent rotamer library for proteins application to side-chain prediction. *Journal of Molecular Biology*, 230(2):543 – 574, 1993.

[25] 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. PMID: 20235548.

[26] Georgii G. Krivov, Maxim V. Shapovalov, and Roland L. Dunbrack. Improved prediction of protein side-chain conformations with scwrl4. *Proteins: Structure, Function, and Bioinformatics*, 77(4):778–795, 2009.

[27] Javier Larrosa and Thomas Schiex. Solving weighted {CSP} by maintaining arc consistency. *Artificial Intelligence*, 159(12):1 – 26, 2004.

[28] 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.

[29] Andrew Leaver-Fay, Michael Tyka, Steven M. Lewis, Oliver F. Lange, James Thompson, Ron Jacak, Kristian W. Kaufman, P. Douglas Renfrew, Colin A. Smith, Will Sheffler, Ian W. Davis, Seth Cooper, Adrien Treuille, Daniel J. Mandell, Florian Richter, Yih-En Andrew Ban, Sarel J. Fleishman, Jacob E. Corn, David E. Kim, Sergey Lyskov, Monica Berrondo, Stuart Mentzer, Zoran Popovi, James J. Havranek, John Karanicolas, Rhiju Das, Jens Meiler, Tanja Kortemme, Jeffrey J. Gray, Brian Kuhlman, David Baker, and Philip Bradley. Chapter nineteen - 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.

[30] Vladimir Lifschitz. What is answer set programming? pages 1594–1597, 2008.

[31] 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.

[32] Ilkka Niemelä. Answer set programming: A declarative approach to solving search problems. In *Logics in Artificial Intelligence*, pages 15–18. Springer, 2006.

[33] Noah Ollikainen, Ellen Sentovich, Carlos Coelho, Andreas Kuehlmann, and Tanja Kortemme. Sat-based protein design. In *ICCAD*, pages 128–135. IEEE, 2009.

[34] Sergio G Peisajovich and Dan S Tawfik. Protein engineers turned evolutionists. *Nature methods*, 4(12):991–994, 2007.

[35] Niles A. Pierce and Erik Winfree. Protein design is np-hard. *Protein Engineering*, 15(10):779–782, 2002.

[36] Jürgen Pleiss. Protein design in metabolic engineering and synthetic biology. *Current opinion in biotechnology*, 22(5):611–617, 2011.

[37] Thomas Schiex. Arc consistency for soft constraints. *Principles and Practice of Constraint Programming CP 2000*, 1894:411–425, 2000.

[38] Luis P.B. Scott, Jorge Chahine, and Jos R. Ruggiero. Using genetic algorithm to design protein sequence. *Applied Mathematics and Computation*, 200(1):1 – 9, 2008.

[39] 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.

[40] Seydou Traoré, David Allouche, Isabelle André, Simon de Givry, George Katsirelos, Thomas Schiex, and Sophie Barbe. A new framework for computational protein design through cost function network optimization. *Bioinformatics*, 29(17):2129–2136, 2013.

[41] Yushan Zhu. Mixed-integer linear programming algorithm for a computational protein design problem. *Industrial and Engineering Chemistry Research*, 46(3):839–845, 2007.

Appendices

Algorithm 11 Simple DEE

```
for all position i do
    for all possible rotamer r at position i do
        for all candidate rotamer t ≠ r at position i do
            X ← E(i_r)
            Y ← E(i_t)
            for all position j ≠ i do
                for all possible rotamer u at position j do
                    Z ← min_u E(i_r, j_u)        ▷ best case for r
                    W ← max_u E(i_t, j_u)        ▷ worst case for t
                end for
                X = X+Z                          ▷ sum of the best cases for r
                Y = Y+W                          ▷ sum of the worst cases for t
            end for
            if X > Y then
                eliminate i_r and break
            end if
        end for
    end for
end for
```

Algorithm 12 Double Goldstein DEE

for all position i_1 **do**
  for all position i_2 **do**
    for all possible pair (r_1, r_2) at position (i_1, i_2) **do**
      for all concurent pair (t_1, t_2) at position (i_1, i_2) **do**
        $X \leftarrow [E(r_1) + E(r_2) + E(r_1, r_2)] - [E(t_1) + E(t_2) + E(t_1, t_2)]$
        ▷ store the energy difference
        for all position j, $j \neq i_1$ and $j \neq i_2$ **do**
          for all possible rotamer u at position j **do**
            $Y \leftarrow \min_u [E(i_{r_1}, j_u) + E(i_{r_2}, j_u) - E(i_{t_1}, j_u) - E(i_{t_2}, j_u)]$
            ▷ minimal energy difference
          end for
          $X \leftarrow X + Y$ ▷ sum of minimum differences
        end for
        if $X > 0$ **then**
          eliminate the pair (i_{1r_1}, i_{2r_2}) and break
        end if
      end for
    end for
  end for
end for