Protein Design using Answer Set Programming
(Extended Abstract)

João Filipe Gouveia
Instituto Superior Técnico / INESC-ID
joao.f.gouveia@ist.utl.pt

October 2012

Abstract

Protein design is a problem that can be useful in different areas like the understanding of some diseases or the production of biofuels. Answer set programming is a very recent approach to declarative problem solving. The goal of this work is to solve the protein design problem using answer set programming.

This paper presents an answer set programming implementation to solve the protein design problem. The protein design problem consist in determining the amino acids to form a specific protein. In this work, three dead-end elimination algorithms were implemented. Results showed that the solving time of the ASP approach increases exponentially with the increase of the number of rotamers considered.

1 Introduction

Proteins are biochemical compounds that take an important role in living cells. There are many different types of proteins each having a different function in the world. Some of the proteins types are enzymes, antibodies, structural components, receptors and transporters. The function of a protein is determined by its structure and therefore it is important to determine the best protein structure for a specific function.

There are many problems related to proteins. Some of these problems are structure prediction, protein design, energy calculation and mutation prediction. In this work we will focus on the protein design problem. The protein design problem consists in determining the best components of a protein for a given protein structure. This way it is possible to design the best protein for a specific function given that the function of a protein greatly depends on its structure.

The goal of this work is to develop a program that solves the protein design problem using answer set programming. Answer Set Programming (ASP)[1, 2] is a very recent approach to declarative problem solving. Difficult search problems, like protein design problems, can be solved using ASP.

2 Proteins

Proteins have a very important role in life. Each protein has a different function in the world. Some of the proteins types are enzymes, antibodies, structural components, etc. *Proteins* are biological compounds that are formed by a sequence of *amino acids*, also called residues. The sequence of amino acids in a protein defines the three-dimensional structure into which the protein folds and this structure is called the *backbone*. Each protein has

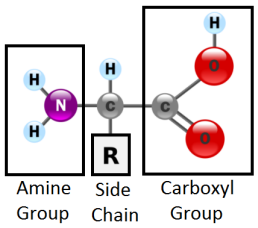

Figure 1: Amino Acid Structure

a specific function and that function greatly depends on its structure.

There are 20 essential amino acids. Each amino acid has an amine and a carboxyl functional group that are common to every amino acid and a *side-chain* that varies. The side-chain is specific for each amino acid, varying in physical properties. Figure 1[1] illustrates the general structure of an amino acid.

The side-chain of an amino acid can have up to four dihedral angles, χ_1, χ_2, χ_3 and χ_4. A *dihedral angle* is measured on a sequence of four atoms where the first three atoms form a plan and the last three atoms form another plan. Not all amino acids have the four degrees of freedom.

Each conformation of the side-chain of an amino acid is called a *rotamer*[3, 4]. There are infinite conformations, or rotamers, for each amino acid side-chain since the dihedral angles can be given with arbitrary precision. It is commonly considered only a representative discrete set of dihedral angles and this set is determined by a statistical analysis of the conformations that occur in nature[5].

2.1 Protein Design

Each protein has a specific function that is greatly determined by its structure. Predicting the sequence of amino acids that form a protein with a specific function can be very useful in different areas such as therapeutics, treatment of diseases, the digestive process of animals or even the production of biofuels.

The protein design problem consists in determining a set of amino acids and respective side-chain conformations to form a protein. The set of amino acids of the protein folds into a well-defined three-dimensional structure[6]. It is relevant to design a protein for a specific function, i.e., determining the set of amino acids that folds into a specific structure.

Many computational methods have been developed to solve the protein design problem[4, 6, 7, 8, 9]. The objective of protein design is to determine the set of amino acids that folds into a defined backbone structure minimizing the energy of the protein. In the protein design problem is common to keep the backbone of the protein fixed and determine the best side-chain conformation of each amino acid. So the structure of the protein, the amine and the carboxyl groups of each amino acid are fixed and given as input. The problem therefore becomes to determine the set of rotamers that minimizes the total energy of a protein, i.e., identify the global minimum energy conformation (GMEC). Hence, protein design becomes an optimization problem.

The energy of a protein can be given by the sum of the energy of the backbone, the interaction energy between two rotamers at different positions and the interaction energy between each rotamer and the backbone of the protein[10]. Given that it is common to keep the backbone fixed, the term related to the energy of the backbone becomes irrelevant for the optimization problem and the energy function can be expressed as[11]:

$$E = \sum_i E\left(r(i)\right) + \sum_i \sum_{j,j<i} E\left(r(i), r(j)\right)$$

where $E\left(r(i)\right)$ is the energy of the interaction

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

between the rotamer $r(i)$ at position i and the backbone of the protein, and $E(r(i), r(j))$ is the energy of the interaction between rotamer $r(i)$ at position i and rotamer $r(j)$ at position j.

The protein design problem, which is *NP-hard*[12], has an enormous solution space. Many approaches have been developed to obtain the optimal or near optimal solutions for the protein design problem. These approaches use different methods such as the Monte Carlo search algorithm[7], dead-end elimination[8, 13], genetic algorithms[6, 14] and the branch-and-terminate algorithm[9].

In the recent past have been developed programs that are related to the protein design problem, like SCWRL4[15], SHARPEN[16], Rosetta[17] and ProtSAT[10]. SCWRL4[15] is a program used for prediction of protein side-chain conformations. SHARPEN[16] is a scalable distribution of an open-source library for protein design and structure prediction. Rosetta[17] is a well known open-source program that can be used in several areas like structure prediction and protein design. ProtSAT[10] is a SAT-based program for protein design.

3 Answer Set Programming

Answer Set Programming (ASP) is an approach to declarative problem solving[1, 2]. ASP is commonly applied to difficult, NP-Hard, search problem. The idea is to create a program that is the definition of the problem, the constraints and facts, instead of creating a program that is an algorithm to solve the problem. Knowledge representation and nonmonotonic reasoning, logic programming with negation, databases and Boolean constraint solving are the roots of ASP[18]. In the ASP approach, a problem is solved by representing the problem as a logic program such that the solutions of the program correspond to the solutions of the problem. An ASP solver can then be applied to the logic program to compute (i.e. search for) the solutions.

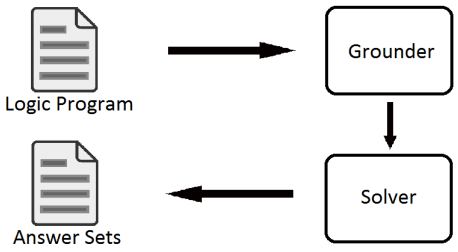

Figure 2: General solving process

An ASP *program*[19] is a set of *rules*. A *rule* is a representation of some knowledge. ASP programs can have several answer sets. An *answer set*[19, 20, 21], or stable model, of a logic program is a set of literals that satisfies all the rules of the program, i.e., that make all the rules of the program true.

Most ASP solvers use grounders as front-end. A grounder is a tool that for a given logical program computes an equivalent variable-free version of the program (a ground program), i.e., replaces the variables in the rules with all possible instantiations. The grounders *lparse*[2] and *gringo*[3] are the most commonly used grounders for ASP. Figure 2 shows the general process of solving an ASP program, i.e., shows the input (logic program) accepted by the grounder which gives its results to the solver and the solver outputs the answer sets of the input program.

Answer set programming can be applied not only to decision problems but also to optimization problems. There are two functions in ASP for optimization problems: *minimize* and *maximize*. It is possible to compute the optimal solution, if there is one, applying ASP to optimizations problems. If the problem does not

[2] `http://www.tcs.hut.fi/Software/smodels/`
[3] `http://potassco.sourceforge.net/`

have a solution, then the information that the program is unsatisfiable is returned. ASP can also be applied to multi-criteria optimization problems[22].

The input languages of *lparse* and *gringo*, the two grounders mentioned before, are very similar to the *Prolog*[4] language. In ASP it is common to used predicates. The arguments of a predicate can start with capitalized letters or lower letters. Arguments starting with capitalized letters mean that they are not instantiated. Arguments starting with lower letters mean that they are instantiated with objects of the world. ASP is considered a closed world, i.e., only the known facts are considered to be true and only the objects that appear in some fact are considered to exist. Everything else is considered to be false or do not exist.

There are several Answer Set Programming solvers. In this work we used *clasp*[23, 18]. As ASP grounder we used *gringo*.

4 Implementation

Two approaches were developed to solve the protein design problem. The protein design problem consists in determining the set of amino acids and respective side-chain that form a protein with a specific function, minimizing the total energy of the protein. In our case we made a simplification to the problem. It is considered that the backbone of the protein is fixed and only the side-chains are designed. It is also considered only a discrete set of rotamers using the Dunbrack Backbone-Dependent Rotamer Library[24]. These rotamers are given as input. For the energies is used the source code of Rosetta[17] to compute the score of the interactions between each possible rotamer and the backbone and the interactions between each pair of rotamers at different positions. The positions to design are also given as input, therefore it is possible to design the entire protein or only a subset of specific positions. Figure 3 illustrates the considered protein design problem.

In a first approach was developed an ASP program to determine the optimal set of rotamers for a given protein. The program takes as input the backbone positions of the protein to design, the residues of each position and the energy values of all rotamers considered using the score function of Rosetta and the rotamer library mentioned before. The program produces as output a side-chain conformation for each protein position designed. In this first approach, only a side-chain prediction is made, i.e., the backbone of the given protein is kept fixed and the residues (or amino acids) of the protein received by the protein backbone are kept fixed as well. Only the best set of rotamers is determined.

The second approach to the problem is similar to the first approach. The difference between the two approaches is that the second one does not keep the residues fixed. The backbone of the protein given as input is fixed and the amino acids and respective side-chains are flexible. The objective is to determine the set of amino acids and respective side-chain that minimize the total energy of the protein.

Two codifications of the problem were developed: a simple codification and codification with two optimization criteria. The first codification is very simple and the set of rules of the program represents the problem as it was described. In a first phase we created the predicates to represent the information of the problem. Then the rules to represent the problem restrictions were implemented, for example, a rule that says that each position of the protein must have exactly one amino acid. Finally, the optimization rule was implemented, which minimizes the total energy of the solution.

In the second codification we modified the predicates used so that the negative energies

[4] `http://www.swi-prolog.org/`

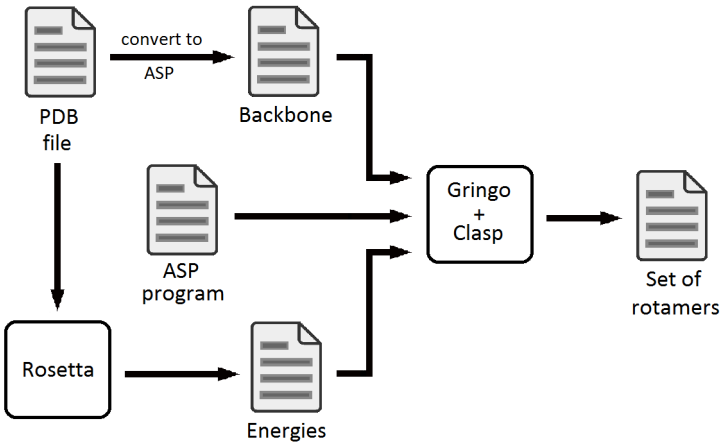

Figure 3: Schema of the problem

of the interactions are represented with different predicates. The idea of this approach is to use a multi-criteria optimization. In these new predicates the energies have the absolute value once the information about being a negative energy is in the name of the predicate. This way we can have two optimization criteria: minimize the positive energy values and maximize the negative energy values. In this approach the two optimization criteria have the exact same priority so that the optimal solution can be found.

Besides the two codifications of the problem, were implemented three dead-end elimination algorithms. The dead-end elimination (DEE) method consists in eliminating the rotamers that cannot belong to the optimal solution[8, 13]. Three algorithms were implemented : Original DEE[25], Simple Goldstein[25, 8] and Simple Split[25, 8]. The three algorithms were implemented using Java. The Original DEE algorithm was also implemented using ASP.

5 Results

To test the DEE algorithms implemented and the codifications made to solve the protein design problem we used 10 proteins. The Protein Data Bank (PDB) id[5] of the proteins are: 1MFG, 1BE9, 2GZV, 2EGN, 2FNE, 1RZX, 1N7F, 1QAU, 1TP3 and 1I92. To restrict the input information data in order to understand the capacities of the codifications made, only a subset of positions were designed for each protein. We choose to design, for each protein, 17, 15, 12 and 10 positions. The chosen positions are positions with a low solvent accessibility, i.e., positions near the core of the protein where most of the amino acids are hydrophobic[14]. For each set of positions we decided to consider three sets of amino acids. For the first approach where only the side-chain conformations are determined, the amino acids of each position are given as input. For the second approach where not only the side-chain but also the amino acids of each position must be determined, we consider two sets of amino acids: the 20 essential amino acids; and only the hydrophobic amino acids.

For each protein, the PDB files were extracted from the Protein Data Bank. We used an adapted source code of Rosetta[17] to compute the discrete set of possible rotamers given

[5] `http://www.pdb.org`

the PDB file of each protein and respective positions to be designed. Rosetta was also used to compute all the necessary energy interaction scores.

Regarding the results of the Original DEE algorithm, the Java approach is better than the ASP approach. The results also showed that in some cases the Simple Split algorithm is faster than the Original DEE algorithm implemented using ASP. The Original DEE algorithm eliminates on average 11,93% of the rotamers. The Simple Split algorithm eliminates 62,52% of the rotamers on average.

In the problem codifications tests we used the rotamers and interactions of each protein with and without using a DEE algorithm first. In other words, we tested the codifications using the possible rotamers given by Rosetta and also tested the codifications using the result of applying the Simple Split algorithm to the original rotamers. We choose to use the Simple Split algorithm because it is the most powerful, i.e., is the algorithm that eliminates the largest number of rotamers.

For the instances designing 17 positions only for the approach in which the backbone of the protein is kept fixed, i.e., the amino acids in each position are given as input, were computed the optimal solutions. The results showed that both codifications, the simple and the double optimization, do not differ much. It also showed that using the DEE algorithm first, as it decreases the number of rotamers, allows to solve a larger number of instances. For the instances that are solved without DEE we can see that using DEE does not increase significantly the total time.

There are 120 instances (12 for each of the 10 proteins). Of these 120 instances, 40 consider the 20 essential amino acids, other 40 consider only hydrophobic amino acids and the remaining 40 consider the amino acids given as input. Table 5 shows the percentage of solved instances with and without using the Simple Split DEE algorithm.

Amino Acids	Without DEE	With DEE
All	0%(0/40)	0%(0/40)
Hydrophobic	0%(0/40)	70%(28/40)
Fixed	100%(40/40)	100%(40/40)

Table 1: Percentage of Solved Instances

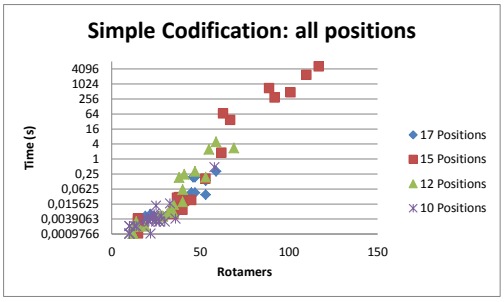

Figure 4: Graphs of Simple Codification Results

Figure 4 shows the relation between the number of rotamers and the time required to compute the optimal solution. The time scale is logarithmic in order to better illustrate the results. It is possible to verify that the time increases exponentially with the number of rotamers given as input. Also, figure 4 shows that above around 110 rotamers is not possible to compute the optimal solution under 3 hours (the limit imposed).

6 Conclusion

The main goal of this work was to verify the capacity of answer set programming when applied to the protein design problem.

Our work has two main differences from the existent tools. First, our approach guarantees the optimal solution for the given input information. Most of the existent tools do not guarantee the optimality of the solution. Second,

our approach uses ASP, a language never used before to solve the protein design problem.

In this work were developed two codifications for the protein design problem. The two approaches do not differ much regarding the number of solutions and the solving time. We also implemented dead-end elimination methods[8, 13] using ASP and Java.

With this work we can conclude that ASP is not very good when there is a large amount of input rotamers. Results showed that for more than around 110 rotamers no solution is given to the protein design problem (see figure 4). In figure 4 it is possible to verify that the solving time of ASP increases exponentially with the increasing of the number of rotamers.

Moreover, the results showed that the implementation of the Original DEE algorithm in ASP for a larger number of rotamers is worse than the implementation in Java of the same algorithm. In some cases, the Simple Split algorithm, which is more powerful than the Original DEE algorithm, if implemented in Java is faster than the Original DEE algorithm implemented in ASP.

However, it is also possible to conclude that, for a small number of rotamers, ASP is very fast. In the codifications of the protein design problem, results showed that for less 50 rotamers, it takes less than one second to compute the optimal solution. Regarding the Original DEE algorithm implemented, it is possible to see that for a small number of rotamers ASP is faster than Java.

ASP can be very good with search and optimization problems with a small amount of input information. When applied to the protein design problem, as it has a large amount of information regarding interactions between rotamers, ASP may not be the best approach.

References

[1] V. Lifschitz, "What is answer set programming?," in *AAAI*, pp. 1594–1597, AAAI Press, 2008.

[2] I. Niemelä, "Answer set programming: A declarative approach to solving search problems," in *JELIA* (M. Fisher, W. van der Hoek, B. Konev, and A. Lisitsa, eds.), vol. 4160 of *Lecture Notes in Computer Science*, pp. 15–18, Springer, 2006.

[3] B. I. Dahiyat and S. L. Mayo, "De novo protein design: Fully automated sequence selection," *Science*, vol. 278, no. 5335, pp. 82–87, 1997.

[4] B. I. Dahiyat and S. L. Mayo, "Protein design automation," *Protein Science*, vol. 5, pp. 895–903, May 1996.

[5] R. L. Dunbrack and F. E. Cohen, "Bayesian statistical analysis of protein side-chain rotamer preferences.," *Protein Science*, vol. 6, pp. 1661–1681, Aug. 1997.

[6] D. T. Jones, "De novo protein design using pairwise potentials and a genetic algorithm," *Protein Science*, vol. 3, pp. 567–574, 1994.

[7] B. Kuhlman and D. Baker, "Native protein sequences are close to optimal for their structures," *Proceedings of the National Academy of Sciences*, vol. 97, pp. 10383–10388, Sept. 2000.

[8] D. B. Gordon, G. K. Hom, S. L. Mayo, and N. A. Pierce, "Exact rotamer optimization for protein design," *Journal of Computational Chemistry*, vol. 24, no. 2, pp. 232–243, 2003.

[9] D. Gordon and S. Mayo, "Branch-and-terminate: a combinatorial optimization

algorithm for protein design," *Structure*, vol. 7, no. 9, pp. 1089–1098, 1999.

[10] N. Ollikainen, E. Sentovich, C. Coelho, A. Kuehlmann, and T. Kortemme, "Sat-based protein design," in *ICCAD*, pp. 128–135, IEEE, 2009.

[11] C. A. Voigt, D. Gordon, and S. L. Mayo, "Trading accuracy for speed: a quantitative comparison of search algorithms in protein sequence design," *Journal of Molecular Biology*, vol. 299, no. 3, pp. 789 – 803, 2000.

[12] N. A. Pierce and E. Winfree, "Protein design is NP-hard," *Protein Engineering*, vol. 15, no. 10, pp. 779–782, 2002.

[13] R. F. Goldstein, "Efficient rotamer elimination applied to protein side-chains and related spin glasses," *Biophysical Journal*, vol. 66, no. 5, pp. 1335–1340, 1994.

[14] J. R. Desjarlais and T. M. Handel, "De novo design of the hydrophobic cores of proteins," *Protein Science*, vol. 4, pp. 2006–2018, Oct. 1995.

[15] G. G. Krivov, M. V. Shapovalov, and R. L. Dunbrack, "Improved prediction of protein side-chain conformations with SCWRL4," *Proteins: Structure, Function, and Bioinformatics*, vol. 77, pp. 778–795, Dec. 2009.

[16] I. V. Loksha, J. R. M. III, C. W. Hong, A. H. Ng, and C. D. Snow, "Sharpen - systematic hierarchical algorithms for rotamers and proteins on an extended network," *Journal of Computational Chemistry*, vol. 30, no. 6, pp. 999–1005, 2009.

[17] C. A. Rohl, C. E. M. Strauss, K. M. S. Misura, and D. Baker, *Protein Structure Prediction Using Rosetta*, vol. 383 of *Methods in Enzymology*, pp. 66–93. Department of Biochemistry and Howard Hughes Medical Institute, University of Washington, Seattle, Washington 98195, USA.: Elsevier, 2004.

[18] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, and M. T. Schneider, "Potassco: The potsdam answer set solving collection," *AI Communications*, vol. 24, no. 2, pp. 107–124, 2011.

[19] V. Lifschitz, "Answer set programming and plan generation," *Artificial Intelligence*, vol. 138, no. 1-2, pp. 39–54, 2002.

[20] M. Gelfond and V. Lifschitz, "The stable model semantics for logic programming," in *ICLP/SLP*, pp. 1070–1080, 1988.

[21] C. Baral, *Knowledge Representation, Reasoning, and Declarative Problem Solving*, pp. 11–39. New York, NY, USA: Cambridge University Press, 2003.

[22] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, "Multi-criteria optimization in answer set programming," in *ICLP (Technical Communications)* (J. P. Gallagher and M. Gelfond, eds.), vol. 11 of *LIPIcs*, pp. 1–10, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[23] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, "clasp: A conflict-driven answer set solver," in *LPNMR'07*, pp. 260–265, Springer, 2007.

[24] M. V. Shapovalov and R. L. D. Jr., "A smoothed backbone-dependent rotamer library for proteins derived from adaptive kernel density estimates and regressions," *Structure*, vol. 19, no. 6, pp. 844 – 858, 2011.

[25] N. A. Pierce, J. A. Spriet, J. Desmet, and S. L. Mayo, "Conformational splitting: A more powerful criterion for dead-end elimination," *Journal of Computational Chemistry*, vol. 21, no. 11, pp. 999–1009, 2000.