

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *01/03/2021* **par :**

MANON RUFFINI

Models and Algorithms for Computational Protein Design

JURY

MARTIN COOPER	Professeur d'Université	Président du Jury
FRÉDÉRIC CAZALS	Directeur de Recherche	Rapporteur
BRUCE DONALD	Professeur titulaire	Rapporteur
RINA DECHTER	Professeur titulaire	Rapporteur
THOMAS SCHIEX	Directeur de recherche	Directeur
SOPHIE BARBE	Directrice de recherche	Directrice

École doctorale et spécialité :
MITT : Domaine STIC : Intelligence Artificielle

Unité de Recherche :
Mathématiques et Informatique Appliqués de Toulouse (INRAE)

Directeur(s) de Thèse :
Thomas SCHIEX et *Sophie BARBE*

Rapporteurs :
Frédéric CAZALS, *Rina DECHTER* et *Bruce DONALD*

À mes parents,

Acknowledgments

Ce manuscrit est le résultat de plusieurs années de travail. Rien n'aurait été possible sans toutes les personnes qui m'ont entourée : des remerciements s'imposent.

D'abord, je tiens à remercier mon directeur Thomas Schiex et ma directrice Sophie Barbe. Merci de m'avoir fait confiance et confié ce sujet de thèse. J'ai énormément appris, et vous m'avez fait réaliser des choses dont je ne me serais pas crue capable.

Thank you to the jury members : Frédéric Cazals, Rina Dechter, Bruce Donald, and Martin Cooper. Thank you for your interest in my work and your helpful comments.

Merci également aux membres de mon comité de thèse : Frédéric Cazals, Martin Cooper, Emmanuel Hebrard et Cédric Pralet.

Merci à l'ENS de Rennes d'avoir financé cette thèse. Merci également à Jérôme Fehrenbach et Emmanuel Paul de m'avoir confié des enseignements à l'IUT d'informatique. Le contact avec les étudiants a été très enrichissant, et l'expérience très formatrice, j'y ai beaucoup gagné.

Merci à l'unité MIAT pour son accueil plus que chaleureux pendant ces 3 années. Merci à tous les gestionnaires : Nathalie, Benjamine, Papa ; et tout particulièrement merci à Fabienne, pour ta gentillesse et ta patience, et à Alain, rarement absent pour rigoler, conseiller un film, et remonter le moral. J'en profite pour remercier Agnès Requis, gestionnaire de l'école doctorale MITT pour sa disponibilité et sa bienveillance. Merci à tous ceux avec qui j'ai pu échanger scientifiquement, de près ou de loin. Un grand merci à Simon, qui m'a beaucoup aidée tout au long de la thèse.

Merci à tous ceux que j'ai croisé, stagiaires, doctorants, cdd, permanents, avec qui j'ai partagé pause thé ou café (quelle température la bouilloire?) ; et ceux avec qui j'ai partagé un bureau (avant la folie Covid) : Clément, Fulya, Malo, Léo, Camille, Pierre et Valentin. Merci à Sylvain, directeur d'unité, ou plutôt grand maître vénéré. Merci pour tes conseils, ton écoute, et ta bienveillance (je pourrais dire que tu fais peur mais il paraît qu'on n'obtient rien par la flatterie...). Merci Malo, Émilien, Étienne, Jelena, pour les midis à la cantine, les fous rires, les sorties, les randos, le vélo, le ski, et SURTOUT les portraits de René Coty. Et merci Lise, merci beaucoup, vraiment, de veiller au grain, de parler tout le temps et de partager tes passions (et tes bras aussi #1000déménagements).

Enfin, merci à tous ceux qui, de plus loin, m'ont conduite jusqu'ici. Merci à mes profs, qui m'ont inspirée et poussée : Florence Ferré, Lucien Dupuy, Alexandre Gauvin et Valérie Desvalois. Merci à David Pichardie, qui m'a

accompagnée dans ma dérive vers l'informatique à l'ENS.

Merci à mes amis, la team KL en particulier. Après le BDE, les Inters, la commuz' et l'agreg, c'est l'étape de la thèse qui se termine, et vous êtes toujours là (malgré mon obsession pour les selfies ridicules). Un merci tout spécial à Marmie : binôme un jour binôme toujours, et Léa : pour la commuz, la zumba, les potins, la gestion de mes crises de nerfs, les crises de rire, et le Brésil.

Merci à Alain et Martine pour leur accueil et leur soutien, en particulier pendant la (difficile) période de la rédaction.

Merci à ma famille, les Ginguay et Ruffini Families, pour leur présence (virtuelle) infaillible et quotidienne. Merci à Maël, Loïs & Florine, de me rappeler que mes lunettes sont moches. Et surtout, merci Papa & Maman, de toujours être là, surtout dans les accès de stress, de fatigue, et aussi de joie. Merci aussi de toujours me rappeler avec beaucoup d'objectivité que de toutes façons je suis la meilleure :).

Last but not least, merci Damien, d'être entré dans ma vie, d'avoir transformé l'imprévu des confinements et du télétravail en des supers souvenirs. Merci pour tes encouragements, ton soutien, et tout le reste...

Contents

Introduction **1**

1 Cost Function Networks & Optimization **5**
1.1 Constraint Satisfaction Problem 6
1.1.1 Definition 6
1.1.2 Resolution strategies 11
1.1.3 Complexity 15
1.2 Weighted Constraint Satisfaction Problem 18
1.2.1 Graphical models and Cost function networks 19
1.2.2 Examples of valuation structures 21
1.2.3 Cost function networks 24
1.3 Tree search 27
1.3.1 Depth-first search 28
1.3.2 Best-first search 28
1.3.3 Other exploration strategies 29
1.4 Variable Elimination 29
1.5 Local consistency and constraint propagation 32
1.5.1 Equivalence preserving transformations 34
1.5.2 Local consistencies 36
1.5.3 Propagation & Global cost functions 46
1.6 Other tools to improve search 49
1.6.1 Branching heuristics 49
1.6.2 Soft neighborhood substitutability 50
1.6.3 Other strategies 50

2 Computational protein design **53**
2.1 Proteins: from sequence to 3D-structure 54
2.1.1 The different structure levels of a protein 54
2.1.2 The protein data bank 59
2.1.3 Protein folding 59
2.1.4 Protein structure prediction: the folding problem . . . 60
2.2 Computational protein design 61
2.2.1 The ultimate goal of CPD 61
2.2.2 Input: the target structure 62
2.2.3 Search space: sequences and conformations 62
2.2.4 Objective: the energy function 64
2.2.5 Optimization methods for the CPD 65
2.2.6 In silico quality estimation of predicted sequences . . . 68

2.3 CPD as a WCSP . . . 69
2.4 Success stories . . . 71
2.5 Limits and challenges . . . 73

3 Guaranteed solution diversity and quality 75
3.1 Ensuring diversity and quality of solutions . . . 76
3.1.1 Mesuring diversity . . . 77
3.1.2 Sets of diverse and good quality solutions . . . 79
3.2 Relation with existing work . . . 81
3.3 Representing the diversity constraint . . . 82
3.3.1 Attempt at the Lagrangian relaxation . . . 82
3.3.2 Using automata . . . 83
3.3.3 Multi-valued decision diagram encoding cost function 85
3.3.4 Exploiting automaton function decomposition . . . 88
3.3.5 Compressing the encoding . . . 89
3.4 Combining the dissimilarity constraints . . . 92
3.5 Greedy DiverseSeq . . . 95
3.6 First experiments on Bayesian networks . . . 97
3.7 Experiments on CPD instances . . . 99

4 Positive and negative multi-state design 111
4.1 Multi-state protein design . . . 112
4.1.1 Multi-state CPD as an optimization problem . . . 112
4.1.2 Existing methods for multistate protein design . . . 114
4.1.3 Complexity . . . 115
4.2 Positive multi-state design as a WCSP . . . 118
4.2.1 Positive MSD as a CFN . . . 119
4.2.2 Experiments . . . 120
4.2.3 CPU-time analysis . . . 121
4.3 Negative multi-state design . . . 123
4.3.1 Negative MSD as a bilevel WCSP . . . 123
4.3.2 Preliminary tests . . . 129

Conclusion 133

Appendices 135

A French Summary 137

Bibliography 159

Introduction

Sudoku is a well-known combinatorial puzzle. In its classic form, a 9×9 grid must be filled with digits, with very simply expressed rules: each row, column, and 3×3 subgrid must contain all the digits from 1 to 9. Yet, given a pre-filled grid, exploring all filling possibilities, until finding one that satisfies the rules could require a time greater than the age of the universe, even if considering a solution only takes one microsecond. The Sudoku is a perfect example of a *Constraint Satisfaction Problem*, where variables must be assigned values, that satisfy a set of simply expressed constraints.

When the problem requires the expression of preference or uncertainty, constraint satisfaction generalizes to *Weighted Constraint Satisfaction*. Variable assignments are associated with a joint cost function, that decomposes into a combination of many simple functions. Simple functions typically involve a small number of variables or can be expressed in a dedicated language, in a very concise manner. The aim is not only to find a solution that satisfies all the constraints but to also ensure that it has minimum cost. Weighted constraint satisfaction can express a wide range of real-world problems, including the design of proteins.

Proteins are complex molecules that govern much of how cells work, in humans, plants, and microbes. They are made of a succession of simple molecules called amino acids. In solution, most proteins fold into a 3D shape, determined by the physicochemical properties of their amino acids. Because of their large variety of functions, and their potentials for applications in medicine, biofuels, green chemistry, etc, new protein sequences are sought, that present desired new or enhanced properties and functions. As function is closely related to structure, computational design methods aim at finding a sequence that folds into a target structure, that corresponds to the desired properties and functionnalities. A general formulation of this problem being highly intractable, simplifying assumptions have been made, that also enable its modeling as a cost function network and its efficient resolution.

Despite successful new protein designs, this so-called single state design process remains unreliable: simplifying assumptions on the backbone and amino acid positions ignore protein flexibility, and the optimized criterion only approximates the actual molecule energy and does not always precisely capture the function. Therefore, the predicted sequence might not meet the targeted properties and functionalities. Computational methods are aimed at producing small libraries of highly relevant sequence candidates, for further experimental testing.

The main goal of this thesis is to develop algorithms to improve the perfor-

mance of the existing cost function network based method for computational protein design. This manuscript is structured into four chapters.

Chapter 1 introduces the basic concepts of weighted constraint programming. After a brief overview of constraint satisfaction and its solving strategies, valuation structures and graphical models are presented. Cost function networks correspond to graphical models with an additive valuation structure. They are fully described, as well as algorithmic strategies that aim at efficiently solving the associated optimization problems.

Chapter 2 provides basic notions on proteins, structures, and folding mechanisms. It then sets the complete computational protein design framework, in its single state form, with input, search space, and optimization criterion descriptions. The cost function modeling is presented, along with existing solving methods.

Chapter 3 tries to provide a better exploration of the sequence search space by providing diversity and quality guarantees for the generation of a library of solutions. A diversity constraint was developed, that enables the greedy production of provably diverse good sequences. Experiments on CPD instances show that sets of diverse sequences have a better probability of providing a good quality sequence, compared to the unconstrained minimum energy solutions.

Chapter 4 shows how search algorithms for single state design can be adapted to both positive and negative multi-state design, thus integrating flexibility information in the model. In positive multi-state design, the input structure is described with a set of backbone conformations (instead of one). The target structure is further specified in negative design, with the information of both desired and undesired states.

Thesis contributions

Implementation of a diversity constraint Controlling diversity regarding sequences of interest is a significant contribution to protein design tools. Indeed, designs should remain dissimilar from patented sequences. Moreover, designs often aim at improving a specific characteristic in an existing protein, by mutating its sequence. Few mutations are to be introduced, in order to reach the target, while maintaining the molecule's other properties.

A diversity constraint was developed that enables the integration of this sequence knowledge in the cost function network model. This constraint also enabled the incremental generation of libraries of diverse good quality sequences. To our knowledge, this is the first combinatorial optimization method that provides both diversity and quality guarantees on the returned set of solutions. This work was published in [Ruffini *et al.* 2019a, Ruffini

et al. 2019b].

Complexity study of multi-state design Single-state design is known to be NP-complete. Multi-state design can be intrinsically harder than single state design, but this additional complexity only appears when negative states are considered. Positive multi-state design can capture flexibility and distinct conformational states of backbones while remaining "only" NP-complete. Negative design is required when the target is to improve specificity or to minimize binding energy. It is however NP^{NP}-complete. The complexity proof has been published in [Vucinic *et al.* 2020].

Bilevel optimization for negative design The NP^{NP}-complete negative design problem can be written as a bilevel query on a cost function network. Existing minimization mechanisms, such as branch-and-bound and local consistencies, were adapted to bilevel optimization. Although further testing is necessary to assess the efficiency of the algorithm and the design quality, preliminary tests are rather promising.

Chapter 1

Cost Function Networks & Optimization

Contents

1.1 Constraint Satisfaction Problem 6
1.1.1 Definition 6
1.1.2 Resolution strategies 11
1.1.3 Complexity 15
1.2 Weighted Constraint Satisfaction Problem 18
1.2.1 Graphical models and Cost function networks 19
1.2.2 Examples of valuation structures 21
1.2.3 Cost function networks 24
1.3 Tree search 27
1.3.1 Depth-first search 28
1.3.2 Best-first search 28
1.3.3 Other exploration strategies 29
1.4 Variable Elimination 29
1.5 Local consistency and constraint propagation 32
1.5.1 Equivalence preserving transformations 34
1.5.2 Local consistencies 36
1.5.3 Propagation & Global cost functions 46
1.6 Other tools to improve search 49
1.6.1 Branching heuristics 49
1.6.2 Soft neighborhood substitutability 50
1.6.3 Other strategies 50

Introduction

This chapter introduces the basic concepts of weighted constraint programming, that will be used throughout this thesis. This paradigm aims at

concisely representing and answering queries on a function of many discrete variables, expressed as a combination of simple functions.

First, Section 1.1 introduces constraint networks, and the constraint satisfaction problem (CSP) and its basic solving strategies. Section 1.2 defines valuation structures and graphical models and gives several examples, that are embedded in this general framework. Two specific cases are particularly developed: Markov Random Fields and Cost Function Networks. The following sections 1.3, 1.4, 1.5 and 1.6 describe algorithmic strategies aimed at efficiently solving the optimisation problem associated to cost function networks.

1.1 Constraint Satisfaction Problem

In its most famous form, the Sudoku consists of a 9×9 grid, where each cell ought to be filled with a digit between 1 and 9. The game's rules are that each row, column, and 3×3 subgrid should present one of all digits from 1 to 9. Some cells are already set, and the aim is to find an assignment of the rest of the grid that is consistent with all the rules.

The game can be extended to the n-Sudoku, where a $n^2 \times n^2$ grid is to be filled with digits from 1 to n^2, with the rows, columns, and $n \times n$ sub-grid rules. There are $(n^2)^{n^4}$ possibilities. Even with $n = 3$, the number of possibilities is higher than the number of atoms in the observed universe.

Constraint programming [Marquis *et al.* 2020] is a paradigm for combinatorial decision making. There are *decision variables* (the Sudoku cells), and *constraints* that express restrictions on combinations of values that sets of decision variables can take (the column, row, and subgrid rules). The task at hand is to find values for the variables that satisfy all the constraints.

The representation of constrained problems as networks has emerged in the 1970's [Montanari 1974], along with solving strategies [Freuder 1978, Freuder 1982, Borning 1981] and constraint-based solvers [Lauriere 1978].

Constraint programming has been used to model and solve many problems [Dechter 2003], including image labelling [Waltz 1975], RNA structure prediction [Gaspin & Westhof 1994], job scheduling [Hasle *et al.* 1996, Adhikary *et al.* 1997], resource allocation [Dincbas & Simonis 1991, Chow & Perett 1997], genetic analysis [Marinescu & Dechter 2006], or planning [Perrett 1991, Chan *et al.* 1998].

1.1.1 Definition

Definition 1 (Constraint network)**.**
A constraint network is a tuple $\mathcal{X} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ that consists in:

- a set $\mathbf{X} = (X_1, \ldots, X_n)$ of n variables.
- a set $\mathbf{D} = (\mathbf{D}_1, \ldots, \mathbf{D}_n)$ of domains.

  Each variable $X_i \in \mathbf{X}$ can be assigned a value from its finite domain $\mathbf{D}_i$. The size of domain $\mathbf{D}_i$ is denoted d_i. The maximum domain size is denoted $d = \max_i d_i$.

  For a set of variables $\mathbf{S} \subset \mathbf{X}$, the set $\mathbf{D_S} = \prod_{X_i \in \mathbf{S}} D_i$ denotes the Cartesian product of the domains of the variables of $\mathbf{S}$. For a tuple of values $\mathbf{t} \in \mathbf{D_Y}$ where $\mathbf{S} \subset \mathbf{Y} \subset \mathbf{X}$, the tuple $\mathbf{t}[\mathbf{S}]$ denotes the projection of $\mathbf{t}$ over the variables of $\mathbf{S}$.

- a set $\mathbf{C}$ of constraints.

  A constraint $\mathbf{c_S} \in \mathbf{C}$ with scope $\mathbf{S}$ is a relation over tuples of $\mathbf{D_S}$. It is the set of allowed combinations of values for the variables in $\mathbf{S}$. Tuples that belong to $\mathbf{c_S}$ satisfy the constraint. Other tuples violate the constraint, they are forbidden. The arity of $\mathbf{c_S}$ is the size of $\mathbf{S}$, denoted $|\mathbf{S}|$.

Solving the Constraint Satisfaction Problem (CSP) corresponds to finding an assignment $\mathbf{t} \in \mathbf{D_X}$ of the variables of $\mathbf{X}$ that satisfy all the constraints, *i.e.*, for all $\mathbf{c_S} \in \mathbf{C}$, the tuple $\mathbf{t}[\mathbf{S}]$ satisfies $\mathbf{c_S}$.

Constraint representation Constraints can be expressed in many ways. The essence of constraint programming is that this representation should remain simple, concise. Here are a few examples of constraints:

- If domains are boolean ($\mathbf{D}_i = \{t, f\}$), a constraint can be represented with a clause. A clause is a disjunction of literals, and a literal is either a variable or the negation of a variable.
- The relation $X_i = X_j$ represents a constraint.
- If only a few variables are involved, the constraint can very simply be represented as the list of valid tuples; or as a tensor (multi-dimensional table) with value t if the corresponding tuple is valid and f otherwise.

  For instance, the equality constraint $x_i = x_j$, with domains $\mathbf{D}_i = \mathbf{D}_j = \{a, b\}$ could be represented as a matrix or a tensor

$$c_{ij} = \{(a,a),(b,b)\} \qquad c_{ij} = \begin{array}{c|cc} & a & b \\ \hline a & t & f \\ b & f & t \end{array}$$

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Figure 1.1: Example of partially filled Sudoku grid

- Dedicated descriptions can be used to describe specific constraints. For example, the constraint *AllDifferent*$(X_1, \ldots, X_i)$ states that variables $X_1, \ldots, X_i$ must all take different values.

Definition 2 (Constraint Satisfaction Problem)**.**
The Constraint Satisfaction Problem (CSP) is defined as follows:

Instance A constraint network $\mathcal{X} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$

Question Does there exist an assignment $\mathbf{t}$ that satisfies all the constraints, *i.e.*, such that for each $\mathbf{c_S} \in \mathbf{C}$, $\mathbf{t}[\mathbf{S}]$ satisfies the constraint $\mathbf{c_S}$?

Such an assignment is called a solution to the CSP.

Example 1 (Sudoku)**.**
The 9×9 *Sudoku problem described in the introduction (see Figure 1.1), can be expressed as a constraint network:*

- *There is one variable per box to fill (81 variables).*

  Let us denote X_{ij} *the variable corresponding to the box on the* i*-th row and* j*-th column. Variable domains are integers from* 1 *to* 9*: for each* i, j, $\mathbf{D}_{X_{ij}} = \{1, \ldots, 9\}$

- *The constraints describe the rules. Variables from the same row, column, or sub-grid must take different values. For example, for each row* $i \in \{1..9\}$*:*

$$\mathbf{c}_i = \mathit{AllDifferent}(X_{i1}, \ldots, X_{i9})$$

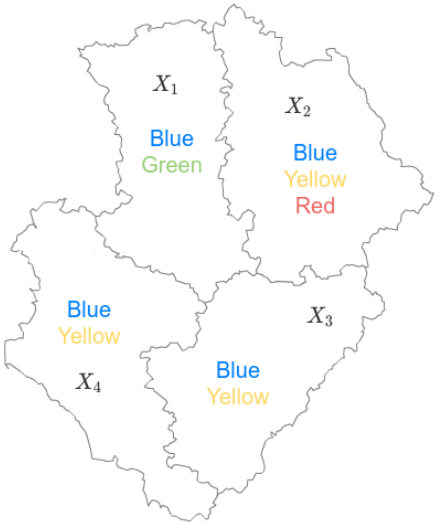


Figure 1.2: Map coloring problem. Each region has a limited set of allowed colors.

Some boxes may contain a digit at the beginning. The domain of the corresponding variables is reduced to that digit.

Example 2. *The map coloring problem [Cormen et al. 2009] shown in Figure 1.2 can be represented as a constraint network with one variable per region, with domain the available colors for that area:*

$$\mathbf{X} = \{X_1, X_2, X_3, X_4\}$$

$$\mathbf{D}_1 = \{Blue, Green\}$$

$$\mathbf{D}_2 = \{Blue, Yellow, Red\}$$

$$\mathbf{D}_3 = \{Blue, Yellow\}$$

$$\mathbf{D}_4 = \{Blue, Yellow\}$$

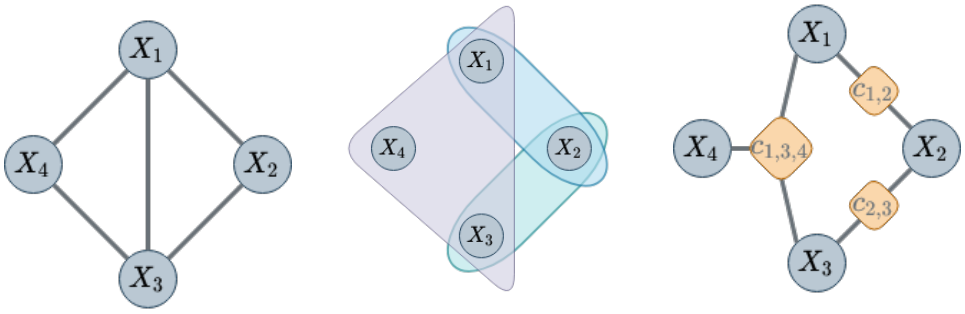
There are 5 *constraints that ensure that two adjacent areas do not have the same color:*

$$\mathbf{c}_{1,2} = \{(Blue, Yellow), (Blue, Red), (Green, Blue), (Green, Yellow), (Green, Red)\}$$

$$\mathbf{c}_{1,3} = \mathbf{c}_{1,4} = \{(Blue, Yellow), (Green, Blue), (Green, Yellow)\}$$

$$\mathbf{c}_{2,3} = \{(Blue, Yellow), (Yellow, Blue), (Red, Blue), (Red, Yellow))\}$$

$$\mathbf{c}_{3,4} = \{(Blue, Yellow), (Yellow, Blue)\}$$



(a) Graph representation

(b) Hypergraph representation

(c) Factor graph representation

Figure 1.3: Graphs representing the constraint network $\mathcal{X} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$, with 4 variables $\mathbf{X} = \{X_1, X_2, X_3, X_4\}$ and 3 constraints $\mathbf{C} = \{\mathbf{c}_{1,2}, \mathbf{c}_{2,3}, \mathbf{c}_{1,3,4}\}$

Graphical representations A constraint network $\mathcal{X} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ can be represented as a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with one vertex per variable. There is an edge e between two vertices X_i and X_j if there exists $\mathbf{c_S} \in \mathbf{C}$ whose scope $\mathbf{S}$ contains both X_i and X_j (see Figure 1.3a):

$$\mathbf{V} = \mathbf{X} \qquad \mathbf{E} = \{\{X_i, X_j\} \mid \exists \mathbf{c_S} \in \mathbf{C}, \{X_i, X_j\} \subset \mathbf{S}\}$$

This representation is sometimes called *constraint graph* or *primal graph*. Although it displays variable interactions, it does not give all the information about the constraint scopes.

A more precise representation of the network can be given by a hypergraph $\mathbf{H} = (\mathbf{V}, \mathbf{E})$, with one vertex per variable, and a subset of variables $\mathbf{S}$ is a hyperedge if it is the scope of a constraint $\mathbf{c_S} \in \mathbf{C}$ (see Figure 1.3b):

$$\mathbf{V} = \mathbf{X} \qquad \mathbf{E} = \{\mathbf{S} \mid \mathbf{c_S} \in \mathbf{C}\}$$

A constraint network can also be represented as a factor graph. This factor graph is the incidence graph of the hypergraph described before. It is a bipartite graph $\mathbf{F} = (\mathbf{V}, \mathbf{V}', \mathbf{E})$ where $\mathbf{V}$ and $\mathbf{V}'$ are sets of vertices and $\mathbf{E}$ is the set of edges, that connect a vertex of $\mathbf{V}$ to a vertex of $\mathbf{V}'$. Here, the vertices of $\mathbf{V}$ are the variables of the network, the vertices of $\mathbf{V}'$ are the constraints. Edges link a constraint vertex to its scope variables (see Figure 1.3c):

$$\mathbf{V} = \mathbf{X} \qquad \mathbf{V}' = \mathbf{C} \qquad \mathbf{E} = \{\{X_i, \mathbf{c_S}\} | X_i \in \mathbf{V}, \mathbf{c_S} \in \mathbf{V}', X_i \in \mathbf{S}\}$$

These graph representations inspired the "graphical model" terminology, used in the next sections.

1.1.2 Resolution strategies

1.1.2.1 Backtracking tree search

A solver of the CSP is required to either prove that no solution exists, or find a solution by assigning a value to each variable. To do so, it can explore a search tree.

The root of a search tree is the initial constraint network $\mathcal{X} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$. An internal node is a constraint network obtained from $\mathcal{X}$ after assigning some of its variables, or reducing some of its domains. Its children nodes are generated by selecting an unassigned variable X_i, and splitting its domain into k parts $\mathbf{D}_i^1, \ldots, \mathbf{D}_i^k$. Each child corresponds to the reduction of $\mathbf{D}_i$ to $\mathbf{D}_i^j, 1 \leqslant j \leqslant k$, *i.e.*, to the constraint network $(\mathbf{X}, \mathbf{D}', \mathbf{C})$, where $\mathbf{D}'$ was deduced from $\mathbf{D}$ by replacing $\mathbf{D}_i$ with $\mathbf{D}_i^j$.

For instance, if $\mathbf{D}_i$ is partitionned as $\{v_1\} \cup \cdots \cup \{v_{d_i}\}$, the children nodes correspond to the assignment of X_i to each value from its domain.

If domain $\mathbf{D}_i$ is split as $\mathbf{D}_i = \{v_i\} \cup \{v_j \mid v_j \in D_i, v_j \neq v_i\}$, the branches to be explored are:

- v_i is assigned to X_i
- v_i is removed from the domain of X_i

This branching strategy generates a binary search tree.

Search tree leaves correspond to complete assignments. If all constraints are satisfied, the assignment is a solution.

An exhaustive, brute-force search would be to explore every tree branch. An improved solving strategy would be to prune the tree as soon as a node is reached, where a constraint is no longer satisfied. In that case, the subtree originating from the node can not lead to any valid solution. The search failed. Previous decisions need to be undone, to climb back in higher tree nodes: this is called backtracking. An example of a possible search tree for the map coloring problem described in Figure 1.2 is given in Figure 1.4.

1.1.2.2 Constraint propagation

The efficiency of the backtracking search relies on the size of the search tree. Constraint propagation can significantly reduce this size, by forbidding values and combinations of values that can not be part of any valid solution.

Example 3. *Let us consider a constraint network with two variables* X_1 *and* X_2*, with domains* $\mathbf{D}_1 = \{1, 2, 3, 4\}$ *and* $\mathbf{D}_2 = \{2, 3, 5\}$ *; and the constraint* $X_1 > X_2$*.*

Values 1 *and* 2 *are forbidden for* X_1*, because the constraint* $X_1 > X_2$ *must be satisfied and* $X_2 \geqslant 2$ *(because of its domain values). Therefore,* 1

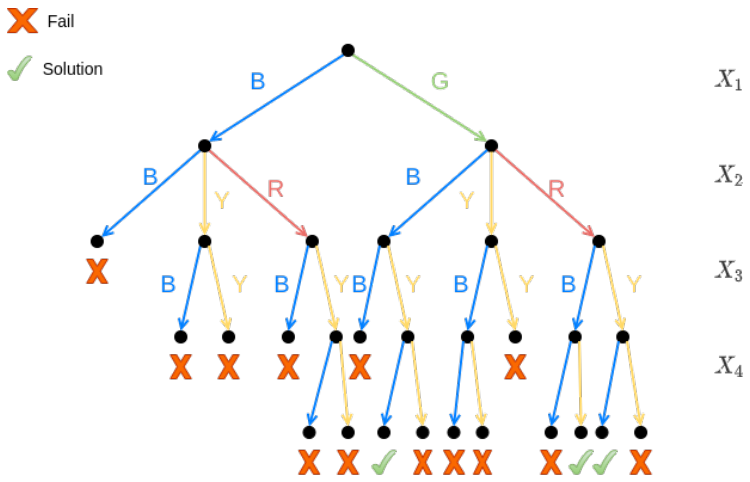


Figure 1.4: Example of a search tree for the map coloring problem of Figure 1.2. After assigning Blue to X_1 and X_2, the constraint $\mathbf{c}_{1,2}$ is violated. The subassignment can not lead to a valid solution. Backtrack takes us to the previous node and X_2 can then be assigned *Yellow*. If the search tree is explored with a left to right Depth first search (DFS), 11 backtracks occur before the solution (*Green*, *Blue*, *Yellow*, *Blue*) is reached.

and 2 *can be removed from* $\mathbf{D}_1$. *Moreover, value* 5 *can be removed from* $\mathbf{D}_2$, *because there is no higher value in the domain* $\mathbf{D}_1$.

Constraint propagation can be viewed as the enforcement of local consistency (LC) properties, that must be satisfied after the propagation phase. The most well-known local consistency property is arc consistency [Barták 1999, Dechter 2003].

Definition 3 (Generalized Arc Consistency (AC))**.**
Let us consider a constraint network $\mathcal{X} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$.

- For any variable X_i, a value $v_i \in \mathbf{D}_i$ is *arc consistent* with respect to a constraint $\mathbf{c_S} \in \mathbf{C}$ when there exists a tuple $\mathbf{t} \in \mathbf{D_S}$ that satisfies the constraint $\mathbf{c_S}$ and such that $\mathbf{t}[X_i] = v_i$.

  Such a tuple $\mathbf{t}$ is called the support of the value $v_i \in \mathbf{D}_i$ on the constraint $\mathbf{c_S}$.

- A constraint $\mathbf{c_S} \in \mathbf{C}$ is *arc consistent* when all the values of all the variables in $\mathbf{S}$ have a support on $\mathbf{c_S}$ (*i.e.* are consistent).

- The network $\mathcal{X}$ is *arc consistent* when all its constraints are arc consistent.

This property is often called generalized arc consistency because it is defined for constraints of arbitrary arity. Here, we will refer to it as arc consistency.

Definition 4. Two constraint networks $\mathcal{X}_1 = (\mathbf{X}, \mathbf{D}_1, \mathbf{C}_1)$, $\mathcal{X}_2 = (\mathbf{X}, \mathbf{D}_2, \mathbf{C}_2)$ are *equivalent* when they have the same set of valid solutions.

Definition 5. An *arc consistency closure* of a constraint network $\mathcal{X}$ is an arc consistent constraint network $\text{AC}(\mathcal{X})$, that is equivalent to $\mathcal{X}$.

The arc consistency closure of a constraint network is unique. In other words, there exists a unique constraint network equivalent to $\mathcal{X}$, that satisfies arc consistency. Computing this equivalent AC network is usually called *enforcing arc consistency* on $\mathcal{X}$. The set of solutions is unaltered, and searching in the AC-closure is easier (faster), because inconsistencies have been propagated, and enable faster pruning.

Arc consistency can be enforced by iteratively removing inconsistent values until reaching a fixpoint. It can be performed at the root node, on the original constraint network, to transform it into an equivalent easier problem, or incrementally at each node in the search tree, to simplify the subproblem and detect inconsistencies (when a domain wipe-out occurs).

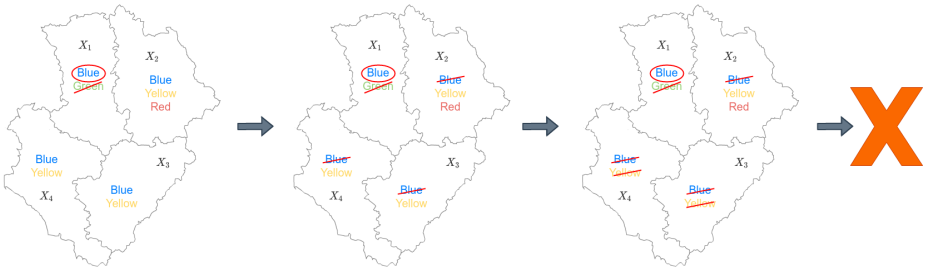
There exist many algorithms for AC enforcement [Barták 1999, Dechter 2003]. The simplest, and best known is AC-3, introduced in [Mackworth 1977]. AC-4 [Mohr & Henderson 1986] was the first time-optimal algorithm. AC-6 [Bessiere 1994] was the first time and space optimal, with respective complexities $O(ed^2)$ and $O(ed)$, with d the domain size and e the number of constraints. However, its complicatedness prevents it from being implemented in solvers. AC-2001 and AC-3.1 [Bessière & Régin 2001, Zhang & Yap 2001] are favored, because they have the same optimal time and space complexity, and they are easier to program.

There also exist stronger consistencies, such as path consistency or the generalizing K-consistency, and corresponding enforcing algorithms.

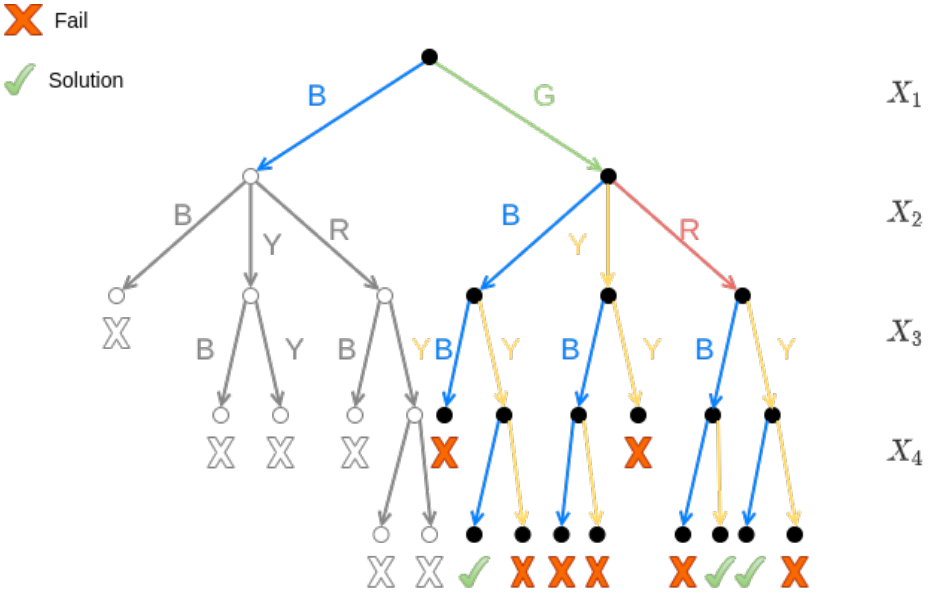
The stronger the consistency, the more uncovered inconsistencies (*i.e.*, constraint violations), and the more pruning. However, strong propagations come at computational and spatial costs. The gain from pruning might be overcome by the propagation time spent in search tree nodes. Therefore, a good trade-off is sought, between pruning efficiency, and propagation complexity.

1.1.2.3 Branching heuristics

Branching heuristics guide the search. They determine which variable, and which value to explore next. These ordering heuristics can either be static



(a) If X_1 is assigned Blue, adjacent variables X_2, X_3 and X_4 can not be Blue. Then, value Yellow in $\mathbf{D}_3$ does not have any support on $c_{3,4}$



(b) As assigning X_1 to Blue leads to a contradiction, the corresponding node will not lead to any solution and does not need to be expanded. The corresponding subtree is pruned.

Figure 1.5: Arc consistency enforcement on the map coloring network, after assigning $X_2 = \mathit{Blue}$, leading to an inconsistency ; resulting pruning in the search tree

(computed before the search starts), dynamic (depend on the network at the current node), or adaptive (the parameters evolve during the execution [Moskewicz *et al.* 2001, Lecoutre *et al.* 2006]). There are generic heuristics, but they can also be specific to the problem at hand. Finally, heuristics might depend on whether we are looking for a solution or whether we think that the problem is infeasible.

If a solution is sought, we want to branch on variables and values that are most likely to yield to a solution. The main drawback of this method is that if the decision we made turns out to be a mistake, we will explore the whole subtree before finding out and backtrack. In that case, we want to explore the infeasible sub-tree as quickly as possible.

For fast infeasibility proof, the first-fail principle can be applied: we want to try first where we are more likely to fail, to prune and backtrack.

A trade-off between those strategies consists of selecting variables that are most likely to cause failure and promising values.

1.1.3 Complexity

1.1.3.1 The polynomial hierarchy

Computational complexity theory organizes computational problems in classes, depending on the resource (like time or space) usage required to solve them. Intuitively, some problems are inherently difficult, because their resolution requires a significant amount of resources. Complexity classes are defined on decidable decision problems. Here, we will focus on time complexity only. A more formal and complete description of complexity classes and polynomial hierarchy can be found in [Papadimitriou 1994] for example. The reader can also refer to the French book [Carton 2008].

The class P gathers problems that are decided by a deterministic Turing machine in polynomial time (polynomial in the size of the input). A problem can be shown to belong to P by exhibiting a polynomial algorithm that solves it.

The class NP consists of problems decided in polynomial time by a *non-deterministic* Turing machine. Such problems can be characterized by the fact that there exists a polynomial time verifier for its certificates (certificates prove whether the answer to the instance is yes).

Of course, $P \subset NP$. Whether P is strictly included in NP, or $P = NP$ remains an open problem.

Example 4. *SAT is the problem of deciding whether a logic boolean formula in conjunctive normal form (CNF) is satisfiable. A formula is in CNF if it is a conjunction of clauses, where clauses are disjunctions of literals, and literals variables or their negation. It can be shown that SAT belongs to NP.*

2SAT is a subclass of SAT. The boolean formula is a CNF, where all clauses have at most two literals. 2SAT is in P. We say that it is tractable.

Definition 6. A *polynomial reduction* of a decision problem A to a decision problem B is a polynomial-time function that transforms instances of A into instances of B, such that, an instance of A is positive if and only if its corresponding instance of B is positive.

If the problem A can be polynomially reduced to B, it means that A is easier to solve than B, because the resolution of B enables the resolution of A through the polynomial reduction. Polynomial reductions enable the identification of a subclass of NP:

Definition 7. A problem is *NP-hard* if every NP problem polynomially reduces to it. A problem is *NP-complete* if it belongs to NP and is NP-hard.

NP-hard problems are therefore problems that are at least as hard as NP because their resolution gives a solution to all NP-problems (in polynomial time). NP-complete problems are the sub-class of the hardest problems of NP.

Example 5. *SAT is NP-complete [Cook 1971, Levin 1973].*

Now that we know that SAT is NP-complete, we can use it for other NP-completeness proofs: if SAT reduces to a problem A, then any problem B from NP can reduce to SAT (because it is NP-hard), then to A.

Definition 8. For a decision problem A, co-A is the complementary (dual) problem of A. The positive instances of A are negative for co-A and vice-versa.

The complementary co-C of a complexity class C is the set of complementary problems of C.

The class co-NP consists of complementary problems of NP. Problems in NP can polynomially check the positivity of a given instance. Conversely, co-NP problems have polynomial checking procedures for negative instances.

Beyond NP The complexity classes P, NP, and co-NP can be generalized, using deterministic and non-deterministic polynomial Turing machines and oracles. A C-oracle answers in constant time a problem of the complexity class C.

The class P^{NP} (also known as Δ_2^{P}) gathers problems that can be solved in polynomial time with an oracle for an NP-complete problem.

The class $\mathrm{NP}^{\mathrm{NP}}$ (also known as Σ_2^{P}) defines problems solved in non-deterministic polynomial time with an oracle for NP-complete problems.

Similarly, the following classes can be defined, that form the polynomial hierarchy.

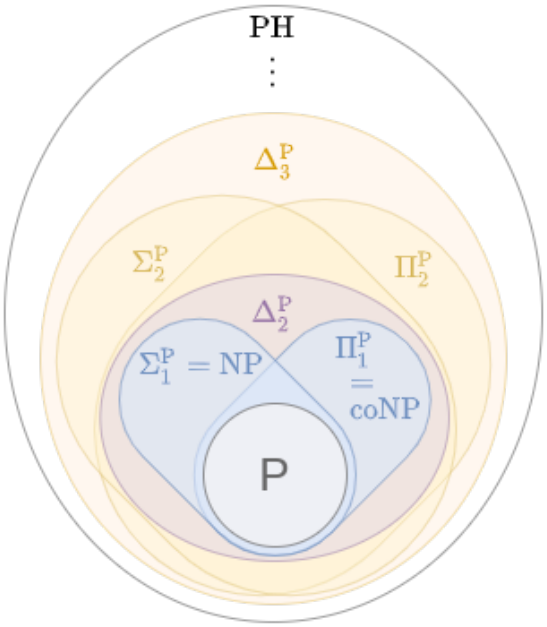


Figure 1.6: The polynomial hierarchy

Definition 9 (The polynomial hierarchy)**.**

$$\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$$

$\forall i \geqslant 0$

$$\Delta_{i+1}^P = P^{\Sigma_i^P} \qquad \Sigma_{i+1}^P = NP^{\Sigma_i^P} \qquad \Pi_{i+1}^P = coNP^{\Pi_i^P}$$

Inclusion relations between complexity classes of the polynomial hierarchy are displayed in figure 1.6. The union of all the complexity classes of the polynomial hierarchy is called the PH and is included in PSPACE. If P=NP, all classes and the PH collapse to P.

Example 6. *The quantified satisfiability problem QSAT, is a generalization of SAT, where universal and existential quantifiers are applied to the boolean variables. An instance of QSAT can be put in prenex normal form:*

$$Q_1 \mathbf{x}_1 Q_2 \mathbf{x}_2 \dots Q_i \mathbf{x}_i \Phi(\mathbf{x}_1, \dots, \mathbf{x}_i)$$

where Φ is a boolean formula, $\mathbf{x}_1, \dots, \mathbf{x}_i$ are sets of boolean variables and $Q_1, \dots, Q_i$ a sequence of alternating quantifiers from $\{\exists, \forall\}$.

QSAT is PSPACE-complete.

Let us consider subclasses of QSAT, with one or two quantification levels:

- *Satisfiability of exitential formulae* $\exists \mathbf{x}, \Phi(\mathbf{x})$ *where* $\mathbf{x}$ *is a set of variables, is the SAT problem. It belongs to NP and as stated before is NP-complete.*
- *Satisfiability of universal formulae* $\forall \mathbf{x}, \Phi(\mathbf{x})$ *is in coNP-complete: it is the dual of SAT.*
- *Satisfiability with two quantification levels* $\exists \mathbf{x} \forall \mathbf{y} \Phi(\mathbf{x}, \mathbf{y})$ *is* Σ_2^P*-complete and* $\forall \mathbf{x} \exists \mathbf{y} \Phi(\mathbf{x}, \mathbf{y})$ *is* Π_2^P*-complete.*

Definition 10. The complexity class #P corresponds to counting problems associated with NP decision problems.

Example 7. *The problem of counting the number of satisfying valuations for a boolean formula #SAT is #P-complete [Valiant 1979].*

1.1.3.2 Complexity of the Constraint Satisfaction Problem

The CSP belongs to NP because checking if an assignment is valid is $O(n \log(d))$ in time, where n is the number of variables and d the domain size.

Moreover, let us note that SAT is a particular case of the CSP, where variables are Boolean (domains $= \{0, 1\}$) and constraints are clauses. Cook's theorem states that SAT is NP-complete. Hence the following result.

Theorem 1. The constraint satisfaction problem is NP-complete.

The NP-hardness can also be proved through the reduction of the k-graph coloring problem, that is known to be NP-complete [Carton 2008, Cormen *et al.* 2009]. As shown in Example 2, the k-graph coloring problem can be expressed as a binary CSP (with constraints of arity less than or equal to 2). This shows that satisfaction in binary constraint networks is also NP-complete.

Therefore, the CSP is inherently hard to solve. That means that no matter the search strategy, the worst case is always exponential (unless P=NP). However, there exist subclasses of the CSP that can be solved in polynomial time. They are called *tractable classes* [Bertele & Brioschi 1973, Freuder 1982, Cooper *et al.* 1994, Dechter 2003].

1.2 Weighted Constraint Satisfaction Problem

Many real-life decision problems can be modeled as constraint networks. The user seeks a solution that satisfies all the requirements expressed by the constraints. In many cases, however, constraints should not express

infeasibilities, but preferences, whose satisfaction is not mandatory, but leads to more appropriate solutions.

Soft constraint networks [Meseguer *et al.* 2006, Marquis *et al.* 2020, Cooper *et al.* 2020] are a generalization of constraint networks that enables the expression of such desires. It was first introduced using fuzzy set theory in 1976 [Rosenfeld *et al.* 1976]. This was followed by various propositions of expression of preferences, dedicated resolution algorithms, and consistency properties [Schiex 1992, Schiex *et al.* 1995, Shapiro & Haralick 1981, Freuder & Wallace 1992].

Simultaneously, a large community developed stochastic graphical models, like Markov Random Fields and Bayesian Networks, that are graph-based representations of probability distributions over multiple variables [Kindermann 1980, Dechter 2013, Dechter 2019].

All these models have a common structure, and the algorithms developed for one framework or another were very much alike. Therefore, graphical models based on a valuation structure were introduced [Schiex *et al.* 1995]. The resulting framework is called the *Valued Constraint Satisfaction Problem* (VCSP). It brings together all the soft constraint and probabilistic variations.

1.2.1 Graphical models and Cost function networks

Definition 11 (Valuation structure)**.** The pair co-domain, combination operator $(\mathbf{V}, \oplus)$ is a *valuation structure* when:

- the co-domain $\mathbf{V}$ is totally ordered, by $\preccurlyeq$; and contains a minimum $\bot$ and maximum element $\top$.
- the operator $\oplus$ is commutative: $\forall a, b \in \mathbf{V}, a \oplus b = b \oplus a$
- the operator $\oplus$ is associative: $\forall a, b, c \in \mathbf{V}, (a \oplus b) \oplus c = a \oplus (b \oplus c)$
- the operator $\oplus$ is monotone: $\forall a, b, c \in \mathbf{V}, (a \preccurlyeq b \Longrightarrow (a \oplus c) \preccurlyeq (b \oplus c))$
- the minimum element $\bot$ is neutral for $\oplus$: $\forall a \in \mathbf{V}, a \oplus \bot = a$
- the maximum element $\top$ is absorbing for $\oplus$: $\forall a \in \mathbf{V}, a \oplus \top = \top$

Definition 12 (Graphical model)**.** A *Graphical Model* (GM) on the valuation structure $(\mathbf{V}, \oplus)$ is defined as a tuple $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{\Phi})$ with:

- a set $\mathbf{X} = (X_1, \ldots, X_n)$ of variables
- a set $\mathbf{D} = (\mathbf{D}_1, \ldots, \mathbf{D}_n)$ of domains. Each variable X_i takes its values in its associated domain $\mathbf{D}_i$, of size less than d.

- a set of functions $\mathbf{\Phi}$. A function $\varphi_{\mathbf{S}} \in \mathbf{\Phi}$, with scope $\mathbf{S} \subset \mathbf{X}$ is defined over $\mathbf{D_S} = \prod_{X_i \in \mathbf{S}} \mathbf{D}_i$ in $\mathbf{V}$:

$$\varphi : \mathbf{D_S} \longrightarrow \mathbf{V}$$

The graphical model $\mathcal{M}$ defines a joint function:

$$\begin{array}{rccc} \Phi_{\mathcal{M}} : & \mathbf{D_X} & \longrightarrow & \mathbf{V} \\ & \mathbf{t} & \longmapsto & \bigoplus\limits_{\varphi_{\mathbf{S}} \in \mathbf{\Phi}} \varphi_{\mathbf{S}}(\mathbf{t}[\mathbf{S}]) \end{array}$$

The monotonicity of the valuation structure ensures that the addition of information can only strengthen it. The minimum and maximum elements enable the expression of functions representing constraints, that take their values in $\{\bot, \top\}$. Tuples that map to $\bot$ are valid, while those mapping to $\top$ are forbidden.

Representation of functions Graphical models are defined as a combination of simple functions. A function is simple because:

- it involves a small number of variables. Then, as domains are finite, the function can be represented as a multi-dimensional table, called a tensor, that maps tuples of values to an element of the valuation structure $\mathbf{V}$.
- it belongs to a class of functions that can be described concisely, in a restricted language. Such functions may (or may not) involve a large number of variables. Examples of functions with such representations are given in Subsection 1.5.3.

Graphical representations A graphical model can be represented graphically. The representations are the same as for constraint networks, with function scopes instead of constraint scopes. There are the graph, hypergraph, and factor graph representations (see subsection 1.1.1).

Definition 13. A valuation structure is *fair* when for all $b \preccurlyeq a \in \mathbf{V}$, there exists a maximum element $c \in \mathbf{V}$ such that $b \oplus c = a$. This element is denoted $a \ominus b$.

Therefore, fairness in a valuation structure provides a pseudo-inverse operator, denoted $\ominus$. Let us note that for all $a \preccurlyeq b \in \mathbf{V}$, $a \oplus (b \ominus a) = b$.

The task at hand in the CSP was to find a solution to the constrained problem. In the soft framework, various quantities can be computed from the joint function. The task at hand is called the *query*, as defined hereafter.

Definition 14. A query on a graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{\Phi})$ corresponds to the computation of:

$$\bigotimes_{\mathbf{t} \in \mathbf{D_X}} \Phi_{\mathcal{M}}(\mathbf{t}) = \bigotimes_{\mathbf{t} \in \mathbf{D_X}} \bigoplus_{\varphi_\mathbf{S} \in \mathbf{\Phi}} \varphi_\mathbf{S}(\mathbf{t}[\mathbf{S}])$$

where $\otimes$ is an elimination operator satisfying the following properties:

- the operator $\otimes$ is commutative: $\forall a, b \in \mathbf{V}, a \otimes b = b \otimes a$
- the operator $\otimes$ is associative: $\forall a, b, c \in \mathbf{V}, (a \otimes b) \otimes c = a \otimes (b \otimes c)$
- the operator $\oplus$ distributes over the operator $\otimes$: $\forall a, b, c \in \mathbf{V}, a \oplus (b \otimes c) = (a \oplus b) \otimes (a \oplus c)$

Examples of queries on graphical models are given in the following Subsection.

1.2.2 Examples of valuation structures

1.2.2.1 Classical constraints

Constraint networks, described in the previous section, correspond to a class of graphical models, on the fair valuation structure:

$$\mathbf{V} = \{t, f\} \qquad t \preccurlyeq f \qquad \oplus = \max = \wedge$$

The constraint satisfaction problem corresponds to the query with the elimination operator $\otimes = \min = \vee$: the aim is to find an assignment that satisfies all the constraints.

1.2.2.2 Fuzzy constraints

Instead of defining constraints as sets of allowed tuples, fuzzy constraint networks describe constraints as fuzzy sets of tuples using membership functions [Zadeh 1999, Dubois & Prade 1980]. They are defined with the following fair valuation structure:

$$\mathbf{V} = [0, 1] \qquad \preccurlyeq = > \qquad \oplus = \min$$

The membership of a tuple is the worst membership amongst all constraints.

A query is defined with elimination operator $\otimes = \max$. The aim is to find an assignment that belongs the most to the constraints, *i.e.*, that has the best, worst membership.

Fuzzy constraint networks can be generalized to triangular norms and co-norms for combination and elimination. The max min query is only a simple version of the fuzzy set and fuzzy logic framework [Klir & Yuan 1995].

1.2.2.3 Possibilistic constraints

Possibilistic networks are dual to fuzzy networks. Instead of describing membership, constraints represent priorities on assignments. The aim is to find an assignment, such that the most important violated constraint is of smallest priority:

$$\mathbf{V} = [0,1] \qquad \preccurlyeq = < \qquad \oplus = \max \qquad \otimes = \min$$

This valuation structure is also fair.

1.2.2.4 Lexicographic constraints

One drawback of the fuzzy framework is that two assignments with same worst membership are equally considered, no matter what the other membership values. Lexicographic constraints overcome this, by considering all membership constraints, in increasing order (see [Meseguer *et al.* 2006, Marquis *et al.* 2020] for more details):

$$\mathbf{V} = [0,1]^* \qquad \preccurlyeq = \succ_{lex} \qquad \oplus = \cup \qquad \otimes = \max$$

A reduction of the lexicographic framework to the weighted framework is given in [Schiex *et al.* 1995].

1.2.2.5 Stochastic models

In the probabilistic framework, functions describe probabilities.

Definition 15 (Markov Random Field)**.** A *Markov Random Field* [Kindermann 1980, Li 2009] is a graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{\Phi})$ with:

- a set $\mathbf{X} = (X_1, \ldots, X_n)$ of random variables
- a set $\mathbf{D} = (\mathbf{D}_1, \ldots, \mathbf{D}_n)$ of domains. Each variable X_i takes its values in domain $\mathbf{D}_i$
- a set $\mathbf{\Phi}$ of potential functions. A potential function $\varphi_{\mathbf{S}}$ maps $\mathbf{D}_{\mathbf{S}}$ to $[0, +\infty]$.

The *joint potential function* is defined as:

$$\begin{array}{rccc} P = \Phi_{\mathcal{M}} : & \mathbf{D_S} & \longrightarrow & [0,+\infty] \\ & \mathbf{t} & \longmapsto & \prod\limits_{\varphi_{\mathbf{S}} \in \mathbf{\Phi}} (\mathbf{t}[\mathbf{S}]) \end{array}$$

A query that arises from this definition is potential maximization. It is defined hereafter and called maximum a posteriory probability (MAP) because the potential can be normalized to a probability distribution.

Definition 16 (Partition function)**.** The *partition function* Z of a Markov random field $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{\Phi})$ is defined as the sum of the potentials over all assignments $\mathbf{t} \in \mathbf{D_X}$:

$$Z = \sum_{\mathbf{t} \in \mathbf{D_X}} P(\mathbf{t}) = \sum_{\mathbf{t} \in \mathbf{D_X}} \prod_{\varphi_\mathbf{S} \in \mathbf{\Phi}} (\mathbf{t}[\mathbf{S}])$$

Note that the partition function computation is a query on the MRF, associated with the elimination operator $\otimes = \sum$. The decision problem associated with this query is #P-complete [Valiant 1979].

The normalization of the potential function P by the partition function Z defines a probability function p associated with the MRF:

$$\begin{array}{rccl} p: & \mathbf{D_X} & \longrightarrow & [0,1] \\ & \mathbf{t} & \longmapsto & \frac{1}{Z} P(\mathbf{t}) = \frac{1}{Z} \prod_{\varphi_\mathbf{S} \in \mathbf{\Phi}} \varphi(\mathbf{t}[\mathbf{S}]) \end{array}$$

Definition 17 (Maximum a posteriori)**.** The maximum a posteriori probability (MAP) on a Markov random field $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{\Phi})$ is the query on $\mathcal{M}$ defined with the elimination operator $\otimes = \max$.

It corresponds to finding the assignment with maximum probability (and maximum potential):

$$\max_{\mathbf{t} \in \mathbf{D_X}} p(\mathbf{t})$$

Bayesian Networks A Bayesian network (BN) is a Markov Random Field with a joint function that directly defines a probability distribution. Its functions are conditional probabilities exclusively, organized in a directed acyclic graph [Pearl 1988]. The Maximum Probability Explanation (MPE) on a Bayesian network corresponds to the MAP query on MRF: the task at hand is to find the assignment with maximum probability.

Additive Markov Random Fields An MRF $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{\Phi})$ with strictly positive potential functions can be expressed with energy functions $e_\mathbf{S} \in \mathbf{E}$ thanks to a logarithmic transformation:

$$\mathbf{E} = \{e_\mathbf{S} = -\log \varphi_\mathbf{S} \mid \varphi_\mathbf{S} \in \mathbf{\Phi}\}$$

The potential function is then an exponential of the energy:

$$P(\mathbf{t}) = \prod_{\varphi_\mathbf{S} \in \mathbf{\Phi}} \varphi_\mathbf{S} = \prod_{e_\mathbf{S} \in \mathbf{E}} exp(-e_\mathbf{S}) = \exp\left(- \sum_{e_\mathbf{S} \in \mathbf{E}} e_\mathbf{S} \right)$$

Potential maximization is equivalent to energy minimization.

1.2.2.6 Weighted constraints

Weighted constraints describe *costs* or *penalties* on tuples. The valuation structure is the integers between 0 and $\top$, $\top$ being finite or infinite. The combination operator is the (bounded) addition.

Weighted constraints are very expressive and capture most of the frameworks described before. This thesis will focus on this class of graphical models, called *Cost function networks* (CFN) [Rosenfeld *et al.* 1976, Freuder & Wallace 1992, Dechter 1999, Dechter 2019, Cooper *et al.* 2020]. Hence, they are more fully described in the next subsection.

1.2.3 Cost function networks

Definition 18 (Cost function network)**.** A *cost function network* (CFN) is a graphical model $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ with:

- a set $\mathbf{X} = (X_1, \ldots, X_n)$ of variables
- a set $\mathbf{D} = (\mathbf{D}_1, \ldots, \mathbf{D}_n)$ of domains. Each variable X_i takes its values in domain $\mathbf{D}_i$.
- a set $\mathbf{C}$ of cost functions. A cost function $c_{\mathbf{S}}$ maps tuples of $\mathbf{D_S}$ to $\{0, \cdots, \top\}$.

The joint function is defined as the bounded sum of the cost functions:

$$\begin{array}{rccc} C_{\mathcal{C}} : & \mathbf{D_X} & \longrightarrow & \{0, \cdots, \top\} \\ & \mathbf{t} & \longmapsto & \sum\limits_{c_{\mathbf{S}} \in \mathbf{C}}^{\top} c_{\mathbf{S}}(\mathbf{t}[\mathbf{S}]) \end{array}$$

where the bounded sum $+^{\top}$ is defined with $a +^{\top} b = \min(a + b, \top)$.

In the valuation structure $(\{0, \cdots, \top\}, +^{\top})$, the value 0 corresponds to $\bot$. Partial assignments with cost $\top$ are forbidden. Cost functions that take their values in $\{0, \top\}$ are hard constraints.

Definition 19 (Weighted constraint satisfaction problem)**.** The weighted constraint satisfaction problem (WCSP) takes a cost function network $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ and answers the query associated with the elimination operator $\otimes = \min$.

It corresponds to finding an assignment with minimum cost:

$$\min_{\mathbf{t} \in \mathbf{D_X}} C_{\mathcal{C}}(\mathbf{t}) = \min_{\mathbf{t} \in \mathbf{D_X}} \sum_{c_{\mathbf{S}} \in \mathbf{C}} c_{\mathbf{S}}(\mathbf{t}[\mathbf{S}])$$

Example 8. *Let us consider the cost function network* $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$*:*

- $\mathbf{X} = (X_1, X_2, X_3)$
- $\mathbf{D}_1 = \{a, b, c\}$ $\mathbf{D}_2 = \{a, b\}$ $\mathbf{D}_3 = \{a, b\}$
- $\mathbf{C} = \{c_\varnothing, c_1, c_{12}, c_{13}, c_{23}\}$

  The function $c_\varnothing$ *has an empty scope. It is a constant in the joint function. As cost functions take positive values,* $c_\varnothing$ *is a lower bound on the joint function.*

  Functions can be described in many ways. If few variables are involved, costs can be put in a tensor (multi-dimensional table). Other representations can be used, such as arithmetical expressions ($f_{ij} = 2x_i + 4x_i x_j^2$*), logical expressions, or restricted dedicated languages (see Subsection 1.5.3). The only requirement is that function descriptions should remain simple.*

  In this example, we will describe cost functions with tensors (multi-dimensional cost tables):

$c_\varnothing = 2$

c_1	
a	1
b	2
c	0

c_{12}	a	b
a	2	0
b	5	0
c	0	0

c_{13}	a	b
a	0	0
b	0	0
c	1	0

c_{23}	a	b
a	$+\infty$	0
b	0	$+\infty$

The WCSP consists of finding the minimum cost:

$$\begin{aligned} C^* &= \min_{\mathbf{t} \in \mathbf{D_X}} C_{\mathcal{C}}(\mathbf{t}) \\ &= \min_{\mathbf{t} \in \mathbf{D_X}} c_\varnothing + c_1(t_1) + c_{12}(t_1, t_2) + c_{13}(t_1, t_3) + c_{23}(t_2, t_3) \end{aligned}$$

It can be shown (for example through an exhaustive enumeration of the possible assignments) that:

$$C^* = C_{\mathcal{C}}(\mathbf{t}^*) = 2 \qquad \mathbf{t}^* = (c, a, b)$$

where $\mathbf{t}^*$ *is an assignment with minimum cost.*

As a CFN is a graphical model, it can be represented as a graph, a hypergraph, or a factor graph. Although these graphs represent the structure of the network, they provide no information on the cost functions.

The *microstructure graph* is a more detailed representation of pairwise CFNs. It has one vertex per value of each variable in the network, and weighted edges represent the costs between pairs of values. Edges with

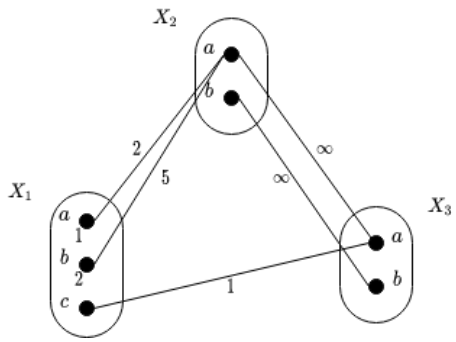


Figure 1.7: Micro-structure graph of the CFN of example 8

weight 0 are not represented. The microstructure of the CFN described in example 8 is shown in Figure 1.7.

1.2.3.1 Cost function networks and Markov random fields

Cost function networks are strongly similar to the energetic expression of Markov random fields. There are however two differences:

- Cost function networks deal with positive integer costs only, whereas MRF energies are real numbers.
- Cost function networks can deal with functions that express hard constraints (with $\top < \infty$), whereas algorithms in the MRF framework usually do not deal well with zero probabilities.

Therefore, if $\top = +\infty$, the CFN $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ can be transformed into an MRF $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{\Phi})$ through an exponential transformation, that converts the sum combination operator into a product:

$$\mathbf{\Phi} = \left\{ \varphi_{\mathbf{S}} \;\middle|\; \begin{array}{c} c_{\mathbf{S}} \in \mathbf{C} \\ \varphi_S : \mathbf{t}_{\mathbf{S}} \mapsto \exp(-c_{\mathbf{S}}(\mathbf{t}_{\mathbf{S}})) \end{array} \right\}$$

Even though CFNs only handle integer costs, an MRF $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{\Phi})$ can be expressed as a CFN $\mathcal{C}^\ell = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ through a log transform, given a precision factor ℓ:

$$\mathbf{C} = \left\{ c_{\mathbf{S}} \mid \begin{array}{c} \varphi_{\mathbf{S}} \in \mathbf{\Phi} \\ c_{\mathbf{S}}(\mathbf{t}_{\mathbf{S}}) \mapsto \lfloor -\log \frac{\varphi_{\mathbf{S}}(\mathbf{t}_{\mathbf{S}})}{\max \varphi_{\mathbf{S}}} \times 10^\ell \rfloor \end{array} \right\}$$

Zero potentials are mapped to cost $\top$, and vice-versa.

1.2.3.2 Complexity

The weighted constraint satisfaction problem can be associated with the following decision problem:

Instance A CFN $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$, an integer $m \in \mathbb{N}$

Question Does there exist an assignment $\mathbf{t} \in \mathbf{D_X}$ of cost less than m ? *i.e.* such that

$$C(\mathbf{t}) = \sum_{c_\mathbf{S} \in \mathbf{C}} c_\mathbf{S}(\mathbf{t}[\mathbf{S}]) \leqslant m$$

Computing the cost of an assignment in the CFN is polynomial. It is linear in the size of the cost tables that describe the cost functions of the network. Hence, the decision problem associated with the WCSP belongs to NP.

Moreover, the CSP is a particular case of this decision problem. Let us consider $m = 0$ and define cost functions that express the constraints, by assigning 0 to valid tuples, and $\top$ to others. This leads to the following result, as the CSP is NP-complete:

Theorem 2. The weighted constraint satisfaction problem is NP-complete.

Optimization is at least as hard as satisfaction. The CSP with boolean variables and constraints of arity at most 2 (2SAT) belongs to P. Optimization on such networks is however NP-complete, by reduction of MAX2SAT. MAXSAT is the optimization problem associated with SAT. It consists of finding an assignment that satisfies the maximum number of clauses. MAXSAT and MAX2SAT (at most 2 literals in every clause) are NP-complete.

The following sections will describe resolution strategies used for solving the WCSP. These technologies are mostly generalizations of those developed for the CSP.

1.3 Tree search

As the CSP, the WCSP can be answered by the exploration of a search tree in which nodes are networks induced by domain reductions on the initial network. This domain reduction step is called *conditionning*. For instance, a binary search tree can be obtained as follows:

The root of the tree is the initial network. At any node in the search tree, there are two possibilities:

1. All the variables are instantiated:

   The node is a leaf that corresponds to a complete assignment.

2. At least one variable is not instantiated. Its domain is of size at least 2:

   A non-assigned variable X_i and a value a from its domain are selected. The left child corresponds to the network induced by assigning a to X_i; the right child to the network induced by removing a from domain $\mathbf{D}_i$.

The aim being cost minimization, a *heuristic function* can be used to estimate the minimum assignment cost in the current subtree. Nodes with a high heuristic cost are not likely to lead to a good solution and are not explored: the corresponding subtree is *pruned*. This pruning strategy is used in the A^* algorithm [Pohl 1970, Hart *et al.* 1968]. Note that the heuristic function is an estimation of the minimum cost, and the optimum branch might be pruned in favor of a low cost though not optimum subtree.

For the assignment returned after the exploration to be optimum, we need the heuristic function to be *admissible*. That means that the optimum branch will not be pruned. In our case, if the heuristic function is a *lower bound* on the current subtree optimum cost, and this lower bound is higher than or equal to the best cost found so far, the subtree can be safely pruned. It will not lead to an assignment with smaller cost. This strategy is called *Branch and Bound* (B&B).

In cost function networks, as cost functions are non-negative, the empty-scoped function $c_\varnothing$ provides a lower bound on the solution costs (in the currently explored subtree). If this lower bound is higher than or equal to the best cost seen so far, the subtree is pruned.

The tighter the lower bound, the more pruning. We will see in Section 1.5 that cost propagations can be performed on the network to increase $c_\varnothing$. Other strategies for lower bound computation have been introduced [Dechter 1997, Verfaillie *et al.* 1996, Dechter & Rish 2003]. They will not be described here.

1.3.1 Depth-first search

Depth-first search is a tree exploration strategy that explores the children of a node before its brothers.

This strategy has worst case linear space complexity $O(dn)$ where d is the maximum domain size and n the number of variables but exponential time $O(\exp(n))$.

1.3.2 Best-first search

The heuristic function that estimates the best solution cost in the subtree defined by the current node can be used for pruning, as stated before. It can

also be exploited to guide the search tree exploration. Here, the heuristic function is the cost lower bound $c_\varnothing$.

Best-first search (BFS) always explores the most promising opened node: at each step, the open node with lowest $c_\varnothing$ in its induced network is selected. Its children are created via conditioning and their heuristics computed. They become *open* nodes. Incrementally, the most promising node among all open nodes is selected, etc.

This strategy has both worst case time and space exponential complexities.

1.3.3 Other exploration strategies

Depth-first search may reach solutions quickly, hence providing an upper bound on the optimum. The maximum cost $\top$ can be updated to this upper bound and used for improved pruning (as we are only interested in finding smaller cost solutions). The main drawback of this search strategy is that a bad decision early in the search tree might stick the search in the wrong branch until it is fully explored.

This drawback is overcome by best-first search, that can open nodes wherever in the search tree. Moreover, BFS provides a global lower bound on the optimum, by taking the minimum lower bound of the open nodes. However, it has exponential space complexity.

Therefore, an anytime hybrid strategy has been developed, that combines both DFS and BFS, called *hybrid best first search* [Allouche *et al.* 2015]. This approach enables the early exploration of complete assignments and can come back to un-explored nodes higher in the search tree. Meanwhile, it provides both lower and upper bounds on the optimum, that give information on the progress of the search and the quality of the best solution so far.

1.4 Variable Elimination

Variable elimination [Bertele & Brioschi 1973, Cooper *et al.* 2020, Dechter 1999, Koller & Friedman 2009, Dechter 2013] is a dynamic programming approach aiming at answering queries on graphical models. It relies on the distributivity of the elimination operator $\otimes = \min$ over the combination operator $\oplus = +^\top$ [Aji & McEliece 2000, Kschischang *et al.* 2001, Bertele & Brioschi 1973, Shafer & Shenoy 1988].

Let $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ be a cost function network. Elimination of variable $X \in \mathbf{X}$ aggregates all the cost functions that involve X in a new cost function called a *message*. This cost function has for scope all the neighbors of X, *i.e.*, all the variables that belong to a scope of a function involving X.

More precisely, let us denote $\mathbf{C}_X$ the set of functions that involve X, and

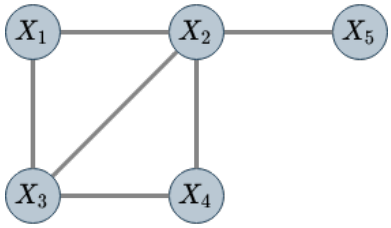


Figure 1.8: Graph representation of the cost function network $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ with 5 variables and 6 cost functions $\mathbf{C} = \{c_{12}, c_{13}, c_{23}, c_{24}, c_{25}, c_{34}\}$

$\mathbf{C}_{-X}$ the set of all the other functions:

$$\mathbf{C}_X = \{c_\mathbf{S} \in \mathbf{C} \mid X \in \mathbf{S}\} \qquad \mathbf{C}_{-X} = \mathbf{C} \setminus \mathbf{C}_X$$

Let $\mathbf{S}_X$ be the set of neighbors of X:

$$\mathbf{S}_X = \{Y \in \mathbf{X} \mid \exists c_\mathbf{S} \in \mathbf{C}, X, Y \in \mathbf{S}\}$$

The query can be decomposed as follows:

$$\begin{aligned}\min_{\mathbf{t}\in\mathbf{D_X}} \sum_{c_\mathbf{S}\in\mathbf{C}} c_\mathbf{S}(\mathbf{t}) &= \min_{\mathbf{t}\in\mathbf{D_X}} \sum_{c_\mathbf{S}\in\mathbf{C}_{-X}} c_\mathbf{S}(\mathbf{t}) + \sum_{c_\mathbf{S}\in\mathbf{C}_X} c_\mathbf{S}(\mathbf{t}) \\ &= \min_{\mathbf{t}_{-X}\in\mathbf{D}_{\mathbf{X}\setminus\{X\}}} \Bigg(\sum_{c_\mathbf{S}\in\mathbf{C}_{-X}} c_\mathbf{S}(\mathbf{t}_{-X}) + \underbrace{\min_{x\in\mathbf{D}_X} \sum_{c_\mathbf{S}\in\mathbf{C}_X} c_\mathbf{S}(\mathbf{t}_{-X}, x)}_{m^{\mathbf{c_X}}_{\mathbf{S_X}}(\mathbf{t}_{-X})} \Bigg)\end{aligned}$$

In stochastic graphical models, the new cost function $m^{\mathbf{c_X}}_{\mathbf{S_X}}$ resulting from the elimination of X is called the *message* from $\mathbf{C_X}$ to $\mathbf{S_X}$ [Pearl 1988]. It combines all the functions that involved X in their scope. This elimination of variable X does not alter the answer to the query [Bertele & Brioschi 1973].

Variables can be eliminated iteratively until only a constant message remains. This message is the empty scoped function $c_\varnothing$ of the resulting network. It corresponds to the answer to the WCSP.

The message $m^{\mathbf{c_X}}_{\mathbf{S_X}}$ can be described in a table of dimension $|\mathbf{S}_X|$. Its computation complexity is $O(d^{|\mathbf{S}_X|+1})$ time and $O(d^{|\mathbf{S}_X|})$ space, where d denotes the maximum domain size [Dechter 1999, Dechter 2013, Cooper *et al.* 2020].

The order in which the variables are eliminated does not alter the result. However, this order influences the neighborhood size, hence the complexity, as shown in example 9.

Example 9. *Let us consider the cost function network* $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ *with*

5 *variables and* 6 *cost functions* $\mathbf{C} = \{c_{12}, c_{13}, c_{23}, c_{24}, c_{25}, c_{34}\}$ *represented in Figure 1.8. Let* c^* *denote the answer to the corresponding WCSP.*

The message $m^{\mathbf{C}}_{X_{i_1},\dots,X_{i_k}}$ *of a set of functions* $\mathbf{C}$ *on a set of indexed variables* $X_{i_1}, \dots, X_{i_k}$ *will also be denoted with variables indices* $m^{\mathbf{C}}_{i_1 \dots i_k}$

Variable elimination with order $(X_5, X_4, X_3, X_2, X_1)$ *transforms the network as follows:*

$$
\begin{aligned}
c^* &= \min_{X_1,X_2,X_3,X_4,X_5} c_{12} + c_{13} + c_{23} + c_{24} + c_{25} + c_{34} \\
&= \min_{X_1,X_2,X_3,X_4} \Big(c_{12} + c_{13} + c_{23} + c_{24} + c_{34} + \underbrace{\min_{X_5} c_{25}}_{m_2^{c_{25}}} \Big) \\
&= \min_{X_1,X_2,X_3} \Big(c_{12} + c_{13} + c_{23} + m_2^{c_{25}} + \underbrace{\min_{X_4} c_{24} + c_{34}}_{m_{23}^{c_{24},c_{34}}} \Big) \\
&= \min_{X_1,X_2,X_3} c_{12} + c_{13} + c_{23} + m_{23}^{c_{24},c_{34}} + m_2^{c_{25}} \\
&= \min_{X_1,X_2} \Big(c_{12} + c_2 + \underbrace{\min_{X_3} c_{13} + c_{23} + m_{23}^{c_{24},c_{34}}}_{m_{12}^{c_{13}c_{23}c_{24}c_{34}}} \Big) \\
&= \min_{X_1} \underbrace{\min_{X_2} c_{12} + m_{12}^{c_{13}c_{23}c_{24}c_{34}} + c_2}_{m_1^{\mathbf{C}}} \\
&= \min_{X_1} m_1^{\mathbf{C}} = m_{\varnothing}^{\mathbf{C}}
\end{aligned}
$$

Successive eliminations only create cost functions of arity 2*, which can each be described in* $d \times d$ *cost tables.*

On the other hand, if X_2 *is firstly eliminated, as it has* 4 *neighbors, a cost function of arity* 4 *must be added to the network. Its description requires a* d^4*-sized cost table.*

$$
\begin{aligned}
c^* &= \min_{X_1,X_2,X_3,X_4,X_5} c_{12} + c_{13} + c_{23} + c_{24} + c_{25} + c_{34} \\
&= \min_{X_1,X_3,X_4,X_5} \Big(c_{13} + c_{34} + \underbrace{\min_{X_2} c_{12} + c_{23} + c_{24} + c_{25}}_{m_{1345}^{c_{12}c_{23}c_{24}c_{25}}} \Big)
\end{aligned}
$$

Solving a WCSP using only variable elimination is often impossible due to its exponential spatial and temporal complexities, especially when eliminating variables with large sets of neighbors. However, there exist

compromise methods that keep the spatial complexity under control [Freuder & Quinn 1985, Dechter 1997, Dechter & Mateescu 2007].

For example, tree search can be combined with on-the-fly variable elimination, that eliminates small degree variables at each node in the search tree.

Elimination exhibits a tractable class of "well-structured" networks (with bounded tree-width), that can be solved in polynomial time [Freuder 1982, Dechter & Pearl 1988, Dechter & Pearl 1989, De Givry *et al.* 2006, Dechter & Mateescu 2007, Bodlaender & Koster 2010, Flerova *et al.* 2017, Jégou *et al.* 2017].

Message passing As stated before, variable elimination is particularly efficient on acyclic networks (with tree-width 1). In this case, the query can be solved in polynomial time by iteratively eliminating leaves until only the root remains. Eliminating leaf-variable X_i with parent X_j consists of computing the message from X_i to X_j

$$m_j^i = m_j^{c_i c_{ij}}$$

and sending it to X_j.

In this framework, variables are not eliminated. Messages are computed and stored. Once X_j has received messages from all its neighbor's nodes but 1, it has become a leaf and its message can be computed and sent to its parent node.

Loopy Belief Propagation The WCSP is NP-complete, and message passing cannot always solve the problem in a reasonable time, the loopy belief propagation heuristic can be used [Pearl 1988]. All binary messages are then initialized to 0, and every message is updated as new information arrives from neighbor variables. These synchronous or asynchronous updates are performed until convergence or until a certain number of iterations has been reached.

1.5 Local consistency and constraint propagation

Branch and bound search assigns variables until reaching a solution or until an inconsistency is reached. In the CFN framework, an inconsistency is detected when the lower bound on solution costs becomes higher than the current upper bound (the cost of the best solution found so far). Therefore, we need strong lower bounds to detect inconsistencies and prune early in the search tree.

As cost functions are all non-negative, the empty-scoped cost function $c_\varnothing$ provides a lower bound on solution costs. Local consistencies aim at condensing as much cost information as possible in $c_\varnothing$ for better pruning while preserving the joint cost function. Indeed, all solution costs must remain unchanged after consistency enforcement.

Constraint propagation as message passing in the CSP For a constraint $c_{\mathbf{S}} \in \mathbf{C}$ and a variable $X_i \in \mathbf{S}$, the message $m_{X_i}^{c_{\mathbf{S}}}$ of constraint $c_{\mathbf{S}}$ on variable X_i can be defined as follows:

$$m_{X_i}^{c_{\mathbf{S}}} = \{v \in \mathbf{D}_i \mid \exists \mathbf{t} \in c_{\mathbf{S}}, \mathbf{t}[X_i] = v\}$$

This message is a constraint on variable X_i, which allows values on X_i that have a support on $c_{\mathbf{S}}$. Note that the domain of X_i can be reduced to the intersection of the unary constraint c_{X_i} and the newly computed constraint $m_{X_i}^{c_{\mathbf{S}}}$, as values that do not belong to the constraint are not valid. Enforcing arc consistency on a constraint network $\mathcal{X}$ corresponds to computing all messages $m_{X_i}^{c_{\mathbf{S}}}$ and adding them to the network until convergence, *i.e.*, until no message brings new information.

In the weighted framework, messages can be computed as unary cost functions on X_i:

$$\begin{array}{rccl} m_i^{c_{\mathbf{S}}} : & \mathbf{D}_i & \longrightarrow & \{0 \dots k\} \\ & v & \longmapsto & \min_{\mathbf{t} \mid \mathbf{t}[X_i]=v} c_{\mathbf{S}}(\mathbf{t}) \end{array}$$

Such messages can not be added to the network without altering the joint cost function. However, thanks to the fairness of the valuation structure, the subtraction $\ominus$ is defined.

Cost propagation in cost function networks consists of computing a message $m_{\mathbf{X}}^{\mathbf{C}}$ of a set of cost functions $\mathbf{C}$ on a set of variables $\mathbf{X}$, adding the message to the network (using $\oplus$), and subtracting it from the cost functions $\mathbf{C}$. That way, the total joint cost function remains unchanged [Koval' & Schlesinger 1976, Schiex 2000].

These movements of costs across the network are called reparameterization in MRFs when the scopes are unchanged [Wainwright *et al.* 2002].

To compute cost messages that will improve the lower bound $c_\varnothing$, local consistency properties have been defined. They guide cost propagations. They rely on equivalence preserving transformations, that ensure that the joint cost function and the optimum are unaltered. Equivalence transformations and the main consistencies are described hereafter.

1.5.1 Equivalence preserving transformations

Definition 20. Two CFNs $\mathcal{C}_1 = (\mathbf{X}, \mathbf{D}, \mathbf{C}_1)$, $\mathcal{C}_2 = (\mathbf{X}, \mathbf{D}, \mathbf{C}_2)$ are *equivalent* [Cooper & Schiex 2004] when they define the same joint cost function:

$$\forall \mathbf{t} \in \mathbf{D}_\mathbf{X}, C_{\mathcal{C}_1}(\mathbf{t}) = C_{\mathcal{C}_2}(\mathbf{t})$$

Definition 21. An *equivalence preserving transformation* (EPT) on a CFN $\mathcal{C}$ is an operation that transforms $\mathcal{C}$ into an equivalent CFN.

Algorithm 1: Projection of cost α from function c_{ij} to unary function c_i on X_i, on value $v \in D_i$

```
1  Procedure Project (X_i, v, X_j, α)
       ▷ Condition min c_ij(v, ·) ⩾ α
2      c_i(v) ← c_i(v) ⊕ α;
3      for w ∈ D_j do
4          c_ij(v, w) ← c_ij(v, w) ⊖ α;
```

Algorithm 2: Extension of cost α from the unary function c_i to the binary function c_{ij} on value $v \in \mathbf{D}_i$

```
5  Procedure Extend (X_i, v, X_j, α)
       ▷ Condition c_i(v) ⩾ α
6      c_i(v) ← c_i(v) ⊖ α;
7      for w ∈ D_j do
8          c_ij(v, w) ← c_ij(v, w) ⊕ α;
```

Algorithm 3: Projection of cost α from the unary function c_i to the constant $c_\varnothing$

```
9  Procedure UnaryProject (X_i, α)
       ▷ Condition c_i(·) ⩾ α
10     c_∅ ← c_∅ ⊕ α;
11     for v ∈ D_i do
12         c_i(v) ← c_i(v) ⊖ α;
```

The different local consistencies that will be introduced can be established using three basic EPTs [Schiex 2000]. These operations shift costs between functions. For the sake of simplicity, we focus here on binary networks, with functions of arity at most two.

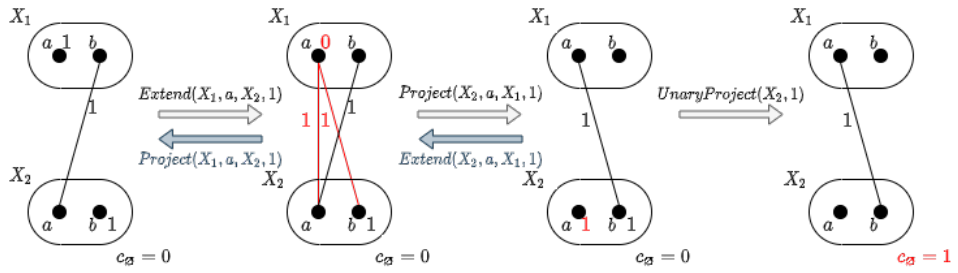


Figure 1.9: Sequence of equivalence preserving transformations applied to a CFN $\mathcal{C}$ with two variables.

- Project moves costs from binary to unary cost functions:

  Let us assume that value $v \in \mathbf{D}_i$ is such that the binary cost $c_{ij}(v, \cdot)$ is always greater than $\alpha > 0$. Then, whatever the assignment of X_j, cost α will always be paid if X_i is assigned v. Therefore, cost α can be moved from the binary cost function c_{ij} with value v, to the unary cost $c_i(v)$. This operation is described in algorithm 1.

  The condition $c_{ij}(v, \cdot) \geqslant \alpha$ ensures that Project will not create any negative cost.

- Extend is the reciprocal from Project. It moves costs from unary cost $c_i(v)$ to the binary cost function $c_{ij}(v, \cdot)$, as described in algorithm 2. It can be viewed as the projection of the negative cost $-\alpha$.

- UnaryProject projects unary costs on $c_\varnothing$. It relies on the fact that if $c_i(v) \geqslant \alpha, \forall v \in D_i$, then the joint function is always greater than α. Therefore, α can be shifted from $c_i(\cdot)$ to $c_\varnothing$.

  UnaryProject increases $c_\varnothing$, enabling more efficient pruning in the search tree. It is described in algorithm 3

Example 10. *Let us consider the CFN* $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ *with variables* $\mathbf{X} = (X_1, x_2)$*, domains* $\mathbf{D}_1 = \mathbf{D}_2 = \{a, b\}$*, cost functions* $c_\varnothing = 0$*,* $c_1 = \begin{pmatrix} 1 & 0 \end{pmatrix}$*,* $c_2 = \begin{pmatrix} 0 & 1 \end{pmatrix}$ *and* $c_{12} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$*, and top cost* $\top = 4$*.*

The CFN $\mathcal{C}$ *is represented in Figure 1.9. The lower bound on its joint cost function is* $c_\varnothing = 0$*. Figure 1.9 displays a sequence of operations applied on the network. First, cost* 1 *is extended from value* a *of* X_1 *to the binary cost function* c_{12}*. Then, as* $\min c_{12}(\cdot, a) = 1$*, cost* 1 *can be projected from* c_{12} *to* $c_2(a)$*. Finally, unary costs* $c_2(a)$ *and* $c_2(b)$ *are both greater than* 1*, so cost* 1 *can be projected from the unary function* c_2 *to the constant* $c_\varnothing$*.*

Therefore, the network $\mathcal{C}' = (\mathbf{X}, \mathbf{D}, \mathbf{C}')$ *with cost functions* $c'_\varnothing = 1$, $c'_{12} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$ *is equivalent to the initial network* $\mathcal{C}$*, and provides a tighter lower bound on its costs* $c'_\varnothing = 1$*. In that case,* $c_\varnothing$ *is also equal to the optimum as* $C'(a, a) = 1$.

Note that EPTs can be applied one way or the other, and propagate back costs.

Sequences of cost-shifting operations applied to a CFN to improve its lower bound $c_\varnothing$ can be designed through properties that are enforced on the CFN. These properties, called *local consistencies*, aim at creating zero unary/binary costs, while improving $c_\varnothing$. These sequences of operations should converge to a fix-point (closure).

1.5.2 Local consistencies

This subsection describes different local consistency properties, and their enforcing algorithms.

Let $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ be a cost function network, with joint cost function C. Let us assume that for all $X_i \in \mathbf{X}$, there is a unary cost function $c_i \in \mathbf{C}$. Cost functions take their values in the valuation structure $V = (\{0..\top\}, +)$, where $\top = k \in \mathbb{N}^*$ or ∞. Note that $\top$ can decrease during the search, as new solutions are found, that provide better upper bounds on the optimum [Larrosa & Schiex 2004].

1.5.2.1 Node consistency

Node consistency [Larrosa & Schiex 2003] is the simplest form of consistency. It focuses on unary costs.

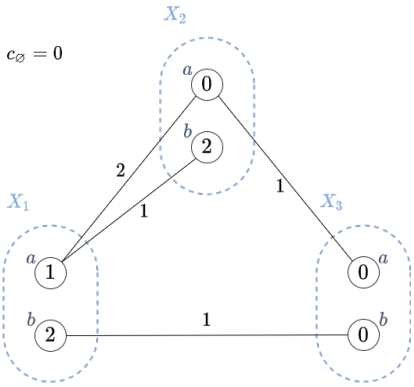
Definition 22. A variable X_i is *node consistent* if:

- For all value $v \in \mathbf{D}_i, c_i(v) + c_\varnothing < \top$
- There exists a value $v \in \mathbf{D}_i$ such that $c_i(v) = 0$

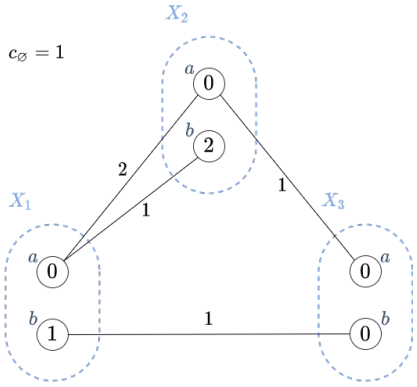
  Value v is called a *support* for X_i.

Definition 23. A CFN is *node consistent* when all its variables are node consistent.

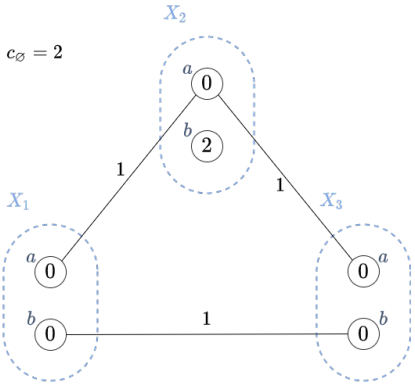
Node consistency can easily be enforced on a CFN by creating unary supports and pruning infeasible values for each variable, with UnarySupport and PruneVar respectively, described in algorithm 4.



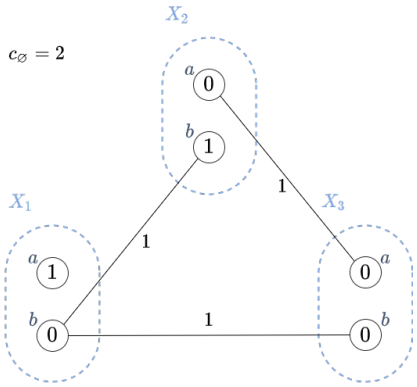
(a) Original cost function network $\mathcal{C}$,



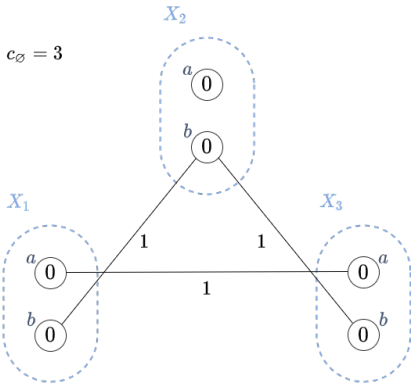
(b) A node consistent closure



(c) An arc consistent closure



(d) A full directional arc consistent closure



(e) An existential directional arc consistent closure

Figure 1.10: Five equivalent CFNs satisfying different levels of local consistency. This example is taken from [De Givry *et al.* 2005]

Algorithm 4: Algorithms for node consistency enforcement

```
13 Procedure UnarySupport (X_i)
14     α ← min_{v∈D_i} c_i(v);
15     if α ≠ 0 then
16         UnaryProject (X_i, α);
17 Procedure PruneVar (X_i)
18     for v ∈ D_i do
19         if c_∅ + c_i(v) ⩾ ⊤ then
20             D_i ← D_i \ {v} ;
```

Example 11. *The network in Figure 1.10a is not node consistent because variable X_1 has no support. A call to* UnaryProject *$(X_1, 1)$ will lead to the CFN in Figure 1.10b, where value a is a support for X_1. Note that the projection increases the constant cost $c_\varnothing$ from 0 to 1.*

1.5.2.2 Arc consistency

Arc consistency [Larrosa & Schiex 2003] tackles cost functions of arity 2 or more. It projects costs from binary to unary functions.

Definition 24. A variable X_i is *arc consistent* with respect to a cost function c_{ij} when for all value $v \in \mathbf{D}_i$:

$$\exists w \in \mathbf{D}_j \text{ such that } c_{ij}(v, w) = 0$$

Value w is called a *simple support* to value v of X_i w.r.t c_{ij}

Definition 25. A CFN $\mathcal{C}$ is *arc consistent* when all its variables are node and arc consistent.

Algorithm 5: Enforcing simple supports on function c_{ij}

```
21 Procedure SimpleSupports (c_ij)
22     for v ∈ D_i do
23         α = min c_ij(v, ·);
24         if α > 0 then
25             Project (X_i, v, X_j, α)
```

Arc consistency is established by enforcing simple supports on every variable w.r.t. every binary function, as done by SimpleSupports in algorithm

5. After projecting costs to unary functions, a call to UnarySupport might be necessary to re-establish node consistency. [Larrosa & Schiex 2003] showed that AC can be enforced in polynomial time $O(ed^3)$.

Example 12. *The NC CFN of Figure 1.10b is not AC because* $\min c_{12}(a, \cdot) = 1$. *Hence, variable* X_1 *has no simple support on* a. *Enforcing arc consistency requires the creation of a simple support on* (X_1, a). *This can be done by calling* Project $(X_1, a, X_2, 1)$.

Note that this projection breaks node consistency on X_1. *A call to* UnaryProject $(X_1, 1)$ *creates the equivalent AC network of Figure 1.10c, with improved lower bound* $c_\varnothing = 2$.

∅-inverse consistency Time complexity for arc consistency enforcement is at least quadratic in the domain size d. Therefore, AC is not suited for networks with large domains. For such problems, a weaker property called ∅-inverse consistency was introduced in [Zytnicki *et al.* 2006, Gaspin *et al.* 2005].

Definition 26. A cost function $c_{\mathbf{S}} \in \mathbf{C}$ is *∅-inverse consistent* when:

$$\exists \mathbf{t} \in \mathbf{D_S} \text{ such that } c_{\mathbf{S}}(\mathbf{t}) = 0$$

The tuple $\mathbf{t}$ is called the *∅-inverse support* of $c_{\mathbf{S}}$.

Definition 27. A CFN $\mathcal{C}$ is *∅-inverse consistent* when all its functions $c_{\mathbf{S}}, |\mathbf{S}| > 1$ are ∅-inverse consistent.

Algorithm 6: Creating a ∅-inverse support for $c_{\mathbf{S}}$ for ∅-inverse consistency enforcement

```
26 Procedure ∅ICSupport (c_S)
27     α = min_{t∈D_S} c_S(t);
28     c_∅ ← c_∅ + α;
29     for t ∈ D_S do
30         c_S(t) ← c_S(t) − α;
```

∅-inverse consistency can be enforced on $\mathcal{C}$ by creating supports for all cost functions, as in algorithm 6: the minimum cost of $c_{\mathbf{S}}$ is computed and projected on $c_\varnothing$.

1.5.2.3 (Full) directional are consistency

Full arc consistency takes both arc and unary costs into account. It relies on the notion of full support [Cooper & Schiex 2004].

Definition 28. A variable X_i is *full arc consistent* (FAC) with respect to a constraint c_{ij} when for all value $v \in \mathbf{D}_i$:

$$\exists w \in \mathbf{D}_j \text{ such that } c_{ij}(v, w) + c_j(w) = 0$$

Value w is called a *full support* to v on X_i w.r.t. c_{ij}.

Definition 29. A CFN $\mathcal{C}$ is *full arc consistent* (FAC) when all its variables are node and full arc consistent.

Algorithm 7: Enforcing full supports on function c_{ij}

```
31 Procedure FullSupports (c_ij)
32     for v ∈ D_i do
33         α[v] = min c_ij(v, ·) + c_j(·);
34     for w ∈ D_j do
35         β[w] = max α[·] − c_ij(·, w) ;
36     for w ∈ D_j do
37         Extend (X_j, w, X_i, β[w]);
38     for v ∈ D_i do
39         Project (X_i, v, X_j, α[v]);
```

Enforcing a full support on value v w.r.t. c_{ij} requires the projection of binary costs from c_{ij} and unary costs from c_j. Negative costs ought not be produced, so that $c_\varnothing$ remains a lower bound on solution costs. So, unary costs need to be extended from c_j to c_{ij} before being projected on $c_i(v)$. Procedure FullSupports described in algorithm 7 extends enough costs from c_j, so that no negative costs are created, while maintaining as much information as possible in unary functions.

Unlike SimpleSupports, iterative calls to FullSupports do not always enforce FAC on a network. Costs can be moved between functions without leading to a fixpoint. Besides, for a CFN $\mathcal{C}$, there does not always exist an equivalent FAC network. Therefore, a weaker property has been introduced, assuming the set of variables $\mathbf{X}$ is totally ordered by $>$:

Definition 30. A variable X_i is *directional arc consistent* (DAC) if every value $v \in \mathbf{D}_i$ has a full support w.r.t. c_{ij}, with $X_j > X_i$.

Definition 31.

- A CFN is *directional arc consistent* when all its values are directional arc consistent.
- A CFN is *full directional arc consistent* (FDAC) when every variable is node, arc and directional arc consistent.

FDAC can be enforced, based on calls to FullSupports, on a CFN in polynomial time [Larrosa & Schiex 2003].

Example 13. *Let us assume that the variables are ordered by* $X_1 < X_2 < X_3$. *Then, the AC network of Figure 1.10c is not FDAC because* (X_1, a) *has no full support w.r.t* c_{12}*:*

$$\min c_{12}(a, \cdot) + c_2(\cdot) = 1$$

FDAC can be enforced by the application of Extend $(X_2, b, X_1, 1)$*, followed with* Project $(X_1, a, X_2, 1)$*. These EPTs lead to the FDAC network of Figure 1.10d.*

Mini-bucket It has been shown [Larrosa & Schiex 2003] that direction arc consistency provides the same lower bound as mini-buckets [Dechter 1997, Dechter & Rish 2003] involving at most two variables. However, unlike DAC that provides a lower bound and an equivalent network, mini-bucket is not incremental.

1.5.2.4 Existential arc consistency

DAC was introduced to enforce full supports on variables. It is weaker than FAC (a FAC network is also DAC), but it guarantees the existence of an equivalent DAC network. Existential arc consistency (EAC) [De Givry *et al.* 2005] aims at improving DAC, by getting closer to FAC, while maintaining the equivalent network existence guarantee.

Definition 32. A variable X_i is *existential arc consistent* (EAC) when there exists a value $v \in \mathbf{D}_i$ such that:

- v is a unary support for X_i: $c_i(v) = 0$
- v has a full support in every constraint c_{ij}

EAC was designed as to enforce full supports only when it leads to an increase of $c_\varnothing$ [De Givry *et al.* 2005]. Existential and directional consistencies can now be combined, to tighten the lower bound:

Definition 33. A CFN is *existential directional arc consistent* when it is FDAC and EAC.

EDAC can be enforced in polynomial time [De Givry *et al.* 2005].

Example 14. *The network of Figure 1.10d is FDAC* $(X_1 < X_2 < X_3)$*. The network is indeed node and arc consistent, and values of* D_1 *and* D_2 *have full supports w.r.t.* c_{12}, c_{13} *and* c_{23} *respectively.*

However, existential arc consistency also requires that at least one value from each domain has a full support in all directions. (X_3, a) *has no full support w.r.t.* c_{23} *and* (X_3, b) *has no full support w.r.t.* c_{13}.

Let us apply the following sequence of EPTs to the CFN:

Extend $(X_2, b, X_3, 1)$

Project $(X_3, a, X_2, 1)$

Extend $(X_1, a, X_3, 1)$

Project $(X_3, b, X_1, 1)$

UnaryProject $(X_3, 1)$

The resulting network, shown in Figure 1.10e is EDAC, and the lower bound was increased to $c_\varnothing = 3$.

1.5.2.5 Optimal soft arc consistency

We have described before different levels of arc consistency (AC, DAC, FDAC, EDAC). Algorithms have been introduced to enforce these consistencies in polynomial time. They rely on the application of equivalence preserving transformations, or EPTs, iteratively, until a fixpoint is reached. Such a fixpoint is called AC-closure (or DAC-, FDAC-, EDAC-closure). Unlike the AC-closure in the CSP framework, weighted AC-closures are not unique, and might even provide different lower bounds.

Furthermore, finding a closure with the tightest lower bound $c_\varnothing$ is NP-hard on integer costs [Cooper *et al.* 2020]. Optimal soft arc consistency aims at overcoming this complexity barrier. To that end, it relaxes the definition of CFN by allowing rational costs, and transitory negative costs. For now, let us assume that $\top = +\infty$ and that all costs are finite, *i.e.*, there are no cost functions that encode hard constraints.

Definition 34. A *Soft arc consistent* (SAC) transformation is a set of EPTs Project, Extend, and UnaryProject that transform a CFN $\mathcal{C}$ into an equivalent valid CFN.

When applied individually, EPTs of a SAC transformation may introduce negative costs. The only requirement is that after all the EPTs, the CFN has only positive costs, and $c_\varnothing$ remains a valid lower bound.

A SAC transformation shifts costs between unary and binary functions and between unary and empty scoped functions. Let us denote $p^j_{i,v}$ the total cost transfered from $c_{ij}(v, \cdot)$ to $c_i(v)$. Note that this cost can be negative if extensions happened from $c_i(v)$ to c_{ij}. Let u_i be the total cost projected from unary function c_i to $c_\varnothing$.

The problem at hand here is finding the tightest (highest) lower bound:

$$\max \sum_{i=1}^{n} u_i$$

The resulting CFN must have non-negative unary and binary costs.

Each unary cost $c_i(v)$ received $p^j_{i,v}$ from each c_{ij} and sent u_i to $c_\varnothing$. So, we need to have:

$$\forall i \in [\![1,n]\!], \forall v \in \mathbf{D}_i, c_i(v) + \sum_{j|c_{ij} \in \mathbf{C}} p^j_{i,v} - u_i \geqslant 0$$

Similarly, each binary cost $c_{ij}(v,w)$ projected $p^j_{i,v}$ and $p^i_{j,w}$ on $c_i(v)$ and $c_j(w)$ respectively:

$$\forall c_{ij} \in \mathbf{C}, \forall v \in \mathbf{D}_i, \forall w \in \mathbf{D}_j, c_{ij}(v,w) - p^j_{i,v} - p^i_{j,w} \geqslant 0$$

So, finding the best lower bound corresponds to solving the following linear programming problem:

$$\max \sum_{i=1}^{n} u_i \qquad \text{subject to}$$

$$\begin{cases} \forall i \in [\![1,n]\!], \forall v \in \mathbf{D}_i, & c_i(v) + \sum_{j|c_{ij} \in \mathbf{C}} p^j_{i,v} - u_i \geqslant 0 \\ \forall c_{ij} \in \mathbf{C}, \forall v \in \mathbf{D}_i, \forall w \in \mathbf{D}_j, & c_{ij}(v,w) - p^j_{i,v} - p^i_{j,w} \geqslant 0 \end{cases}$$

If the CFN framework is relaxed to rational costs, this program can be solved in polynomial time [Karmarkar 1984].

Solving this problem, and applying cost projections of the resulting $u_i, p^j_{i,v}$ creates an equivalent CFN with lower bound $c_\varnothing$. It was shown that no equivalent CFN with the same set of function scopes can present a better lower bound [Cooper *et al.* 2007].

OSAC was introduced for networks with soft constraints only. If $\top < +\infty$, or if infinite costs exist in the network, the hard constraints are not encoded in the linear program. In that case, soft and hard propagations are performed iteratively, until convergence: OSAC is established, then hard constraints are propagated, then OSAC is established on the resulting problem, etc. This iterative procedure remains polynomial.

Although Optimal soft arc consistency provides very strong lower bounds, its computational cost is often too high, and it was never built in the solver ToulBar2. It is however available in the former version ToulBar.

1.5.2.6 VAC

As stated in [Cooper *et al.* 2007], the tightness of OSAC lower bounds suggests that better consistencies can be defined, by seeking sets of EPTs that, when applied simultaneously, will improve the lower bound.

Besides, ($\varnothing$-E-FD) arc consistencies on CFNs all reduce to arc consistency in the hard constraint framework.

As described hereafter, virtual arc consistency (VAC [Cooper *et al.* 2008, Cooper *et al.* 2010]) infers a SAC transformation from the AC-closure of a constraint network derived from the CFN.

Definition 35. Given a CFN $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$, the constraint network $Bool(\mathcal{C})$ is defined as $Bool(\mathcal{C}) = (\mathbf{X}, \mathbf{D}, \bar{\mathbf{C}})$ with constraints:

$$\bar{\mathbf{C}} = \{\bar{c_{\mathbf{S}}} \mid c_{\mathbf{S}} \in \mathbf{C}\} \text{ where } \forall \mathbf{t} \in \mathbf{D_S}, (\mathbf{t} \in \bar{c_{\mathbf{S}}} \Leftrightarrow c_{\mathbf{S}}(\mathbf{t}) = 0)$$

So, if there exists a solution to $Bool(\mathcal{C})$, it has cost $c_\varnothing$ in $\mathcal{C}$.

Definition 36. A CFN $\mathcal{C}$ is *virtual arc consistent* (VAC) when the arc consistent closure of $Bool(\mathcal{C})$ is not empty (all its domains are non-empty).

VAC is applied on $\mathcal{C}$ as follows: the constraint network $Bool(\mathcal{C})$ is computed, and arc consistency is enforced on it. If the closure is non-empty, then $\mathcal{C}$ is VAC. Otherwise, the sequence of AC transformations on $Bool(\mathcal{C})$ is used to build a set of EPTs to be applied on $\mathcal{C}$ that will improve its lower bound. These transformations may introduce rational costs. The process is described on a small CFN in example 15.

Example 15. *An example of virtual arc consistency enforcement is illustrated in Figure 1.11. Network A represents the original CFN. It is EDAC for the variable ordering $X_1 < X_2 < X_3$. Its lower bound is $c_\varnothing = 0$.*

All costs are either 0 or 1, so $Bool(\mathcal{C}) = \mathcal{C}$. We can propagate arc consistency on $Bool(\mathcal{C})$. These propagations lead to an inconsistency in the equivalent constraint network D, where the domain of X_3 is empty.

This means that network A is not VAC. We can use the operations performed to enforce AC on $Bool(\mathcal{C})$ to deduce a sequence of SAC operations to apply on $\mathcal{C}$ to improve its lower bound. To that end, let us consider the network in E. The empty domain in X_3 tells us that costs will be unary-projected from c_3 to $c_\varnothing$. Let λ be the amount of cost projected.

Let us trace back the operations that lead to the inconsistency, and apply their reverse to E, with cost λ. A projection's reverse is an extension and an extension's reverse is a projection. For example, the last operation that emptied the domain $\mathbf{D}_3$ was Project $(X_3, a, X_2, 1)$. *Reversely, we apply* Extend (X_3, a, X_2, λ) *to network E. This results in the network F. All the reverse operations are performed, bottom to top, to network I.*

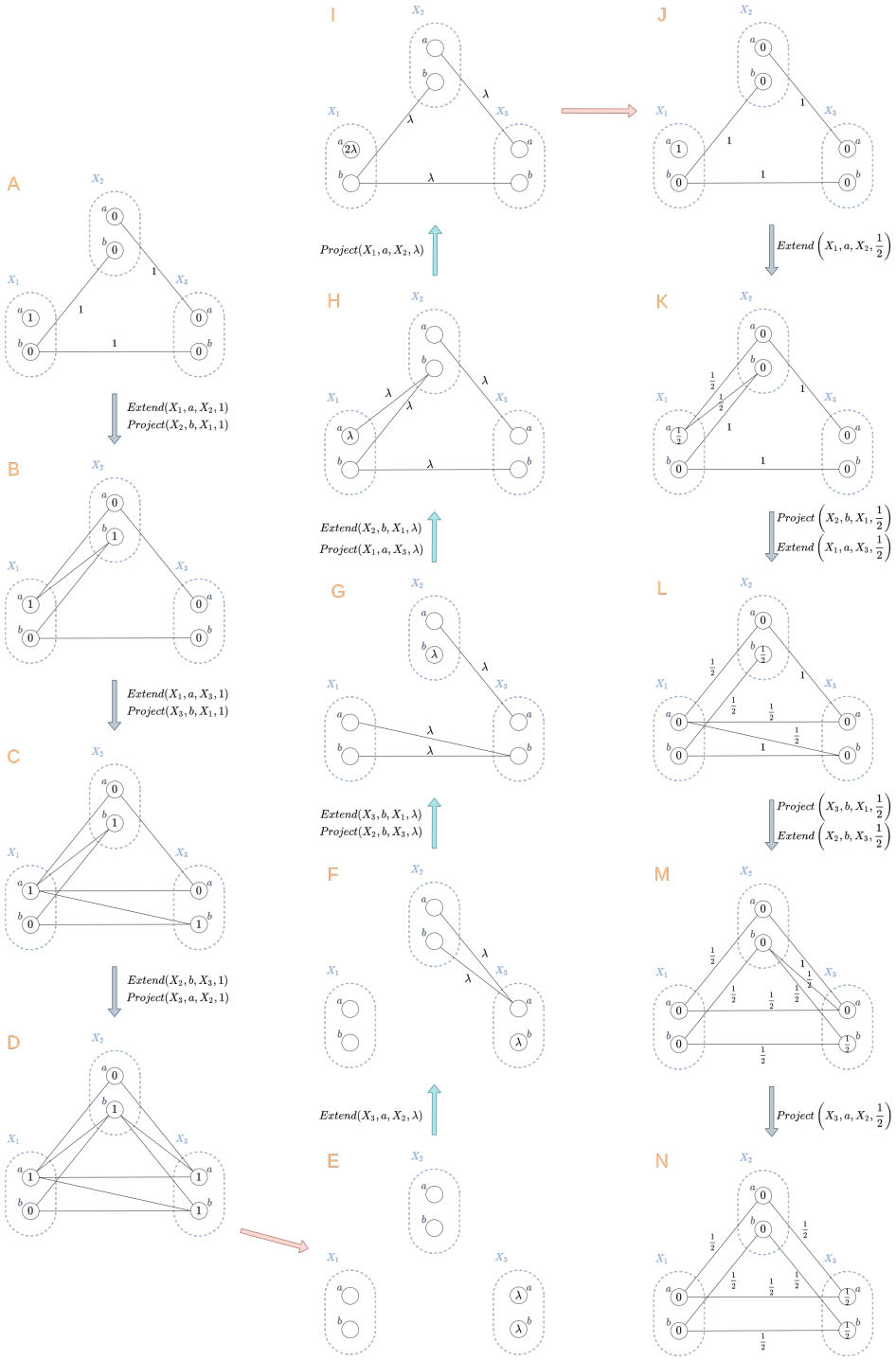


Figure 1.11: Example of CFN where enforcing virtual arc consistency increases the lower bound to $c_\varnothing = \frac{1}{2}$. This example was taken from [Cooper *et al.* 2008].

Network I displays cost quantities that will be necessary from the original network, to project λ *from* c_3 *to* $c_\varnothing$*. In particular, cost* 2λ *will be taken from* $c_1(a)$*. As* $c_1(a) = 1$ *in the original network, we deduce that* $\lambda = \frac{1}{2}$*.*

Now, from network J (equal to the original network A), we can perform the sequence of operations that were applied to $Bool(\mathcal{C})$*, with cost* $\lambda = \frac{1}{2}$*.*

These operations take us to network N. Then, we can apply UnaryProject $(X_3, \frac{1}{2})$*. This increases the lower bound* $c_\varnothing$ *to* $\frac{1}{2}$*.*

As the joint cost function takes integer values, we can deduce that the minimum cost will be at least 1*.*

The reader may refer to [Cooper *et al.* 2008, Cooper *et al.* 2010] for detailed polynomial VAC enforcing algorithms.

Submodularity Given an ordering on the domains, a cost function $c_{\mathbf{S}} \in \mathbf{C}$ is *submodular* when $\forall \mathbf{t}, \mathbf{t}' \in \mathbf{D_S}$,

$$c_{\mathbf{S}}([\min(\mathbf{t}[X], \mathbf{t}'[X]]_{X \in \mathbf{S}})]) + c_{\mathbf{S}}([\max(\mathbf{t}[X], \mathbf{t}'[X])]_{X \in \mathbf{S}}) \leqslant c_{\mathbf{S}}(t) + c_{\mathbf{S}}(t')$$

A CFN is *permuted submodular* [Schlesinger 2007] if there exists an ordering of the domains such that its functions are submodular.

[Cooper *et al.* 2008, Cooper *et al.* 2010] showed that (permuted) submodular problems could be solved in polynomial time by enforcing VAC.

An alternative converging reparameterization method TRW-S has been introduced [Kolmogorov 2006], that provides strong lower bonds. Although the strategy is different, TRW-S and VAC converge to similar fixpoints.

1.5.3 Propagation & Global cost functions

The local consistency properties introduced before apply to small arity cost functions. Here, they were presented for binary networks but they can easily be generalized to higher arity, as long as the tensors (multi-dimensional cost tables) that represent the functions remain of reasonable size. Yet, sometimes, we need specific functions that involve a lot of variables. They are called *global cost functions* and are represented with dedicated concise descriptions.

As stated before, local consistencies are essential for search efficiency, and they must be enforced on these global functions. Therefore, dedicated algorithms are developed for global functions, that can rely on flow theory, directed acyclic graphs or decompositions, to enable (minimum) support computation and cost extensions/projections [Lee & Leung 2009, Lee & Leung 2012, Allouche *et al.* 2012, Allouche *et al.* 2016].

Examples of characterizations of well-behaved global cost functions and corresponding algorithms are given in this subsection.

Definition 37. A *global cost function*[Allouche *et al.* 2016], denoted $\mathrm{GCF}(\mathbf{S}, \mathcal{A})$ is a family of cost functions, with scope $\mathbf{S}$ and possible parameters $\mathcal{A}$.

Example 16. *The global cost function* Sum *can be defined on a scope* $\mathbf{S}$ *of variables with domains of positive integer values:*

$$\begin{array}{rccl} \textsc{Sum}(\mathbf{S}): & \mathbf{D_S} & \longrightarrow & \{0..\top\} \\ & \mathbf{t} & \longmapsto & \min(\top, \sum_{X \in \mathbf{S}} \mathbf{t}[X]) \end{array}$$

Example 17. *Given a regular language* $\mathcal{R}$ *[Linz 2006], the cost function* $\textsc{Regular}(\mathbf{S}, \mathcal{R})$ *is equal to the Hamming distance between an assignment of* $\mathbf{S}$ *and the regular language* $\mathcal{R}$ *(the minimum Hamming distance to the words in* $\mathcal{R}$*).*

A similar function can be defined with the edit distance [Crochemore & Rytter 1994].

Definition 38. A global cost function is

- *tractable* when its minimum can be computed in polynomial time.
- *tractable projection-safe* when it is tractable and any global cost function remains tractable, being derived from it by a series of equivalence preserving transformations Project, Extend, or UnaryProject.

There exist several characterizations of functions that are sufficient conditions for tractable projection safety. These characterizations provide the programmer with strategies for weighted local consistency enforcements. Two of them are described below.

1.5.3.1 Flow-based projection safety

Definition 39. A *flow network* is a directed graph $G = (V, E)$, with capacity function $d : E \to \mathbb{R}$, and cost function $c : E \to \{0, \ldots, \top\}$.

A (s,t)*-flow* in G from a source $s \in V$ to a sink $t \in V$ is a function $f : E \to \mathbb{R}$ such that:

- For all $(u,v) \in E$, $0 \leqslant f(u,v) \leqslant d(u,v)$ *Capacity*
- For all $u \in V \setminus \{s,t\}$: *Conservation*

$$\sum_{(u,v)\in E} f(u,v) = \sum_{(v,u)\in E} f(v,u)$$

- $\sum_{(s,u)\in E} f(s,u) = \sum_{(u,t)\in E} f(u,t)$ *Value*
  This quantity is the *value* of the flow.

The *cost* of the flow f is defined as $cost(f) = \sum_{(u,v)\in E} c(u,v) \times f(u,v)$

Definition 40. A global cost function $\mathrm{GCF}(\mathbf{S}, \mathcal{A})$ is *flow-based* when it can be represented as a flow network G such that the minimum cost among maximum flows [Cormen *et al.* 2009] is equal to $\min \mathrm{GCF}(\mathbf{S}, \mathcal{A})$.

Definition 41. A global cost function is *flow-based projection safe* when it is flow-based and remains flow-based after any sequence of projections/extensions.

Example 18. *The global cost function* AllDifferent *returns the number of variables in its scope that share the same value. As stated in [Lee & Leung 2009, Lee & Leung 2012],* AllDifferent *is flow-based projection-safe.*

[Lee & Leung 2009, Lee & Leung 2012] showed that a flow-based projection safe global cost function is tractable projection safe. Supports of such a function are efficiently computed using min-cost maximum flows.

1.5.3.2 Function decomposition

Definition 42. A global cost function $\mathrm{GCF}(\mathbf{S}, \mathcal{A})$ is p-decomposable if it can be replaced in the network by a set of additional variables $\mathbf{X}'$ and a set of functions $\mathbf{C}'$ of arity at most p

$$\forall c_{\mathbf{S}} \in \mathbf{C}', |S| \leqslant p$$

while preserving network equivalence (after projection)

$$\forall \mathbf{t} \in \mathbf{D_S}, \mathrm{GCF}(\mathbf{S}, \mathcal{A})(\mathbf{t}) = \min_{\mathbf{t}' \in \mathbf{D}_{\mathbf{S}\cup\ \mathbf{X}'} | \mathbf{t}'[\mathbf{S}] = \mathbf{t}} \sum_{c_{\mathbf{S}'} \in \mathbf{C}'} c_{\mathbf{S}'}(\mathbf{t}'[\mathbf{S}'])$$

Example 19. *Let us introduce the* WRegular$(\mathbf{S}, \mathcal{A})$ *cost function, where* $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ *is a finite weighted automaton with alphabet* Σ*, set of states* Q*, transition function* $\delta : Q \times \Sigma \times Q \to \{0, \top\}$ *that associates transitions with costs, initial state* q_0 *and final states* F*. Forbidden transitions are associated with cost* $\top$*.*

The cost of $\mathbf{t} \in \mathbf{S}$ *is a minimum cost path associated with* $\mathbf{t}$*, if it exists. If* $\mathbf{t}$ *does not belong to the regular language defined by* $\mathcal{A}$*, it has infinite cost* $\top$*.*

Let $\mathbf{S} = (X_1, \ldots, X_n)$ *be the scope of the cost function. As stated in [Allouche et al. 2016],* WRegular *can be decomposed as:*

- *A set of variables* $Q_0, \ldots, Q_n$ *with domain* $\mathbf{D}_{Q_i} = Q$*. The value of variable* Q_i *represents the state of the automaton reached after reading values of* $X_1, \ldots, X_i$*. Domains for* Q_0 *and* Q_n *are restricted to the initial and final states:* $\mathbf{D}_{Q_0} = \{q_0\}, \mathbf{D}_{Q_n} = F$*.*

- *A set of ternary cost functions* $c^i, 1 \leqslant i \leqslant n$ *with scope* (Q_{i-1}, X_i, Q_i) *that maps a triple* (q, v, q') *to the weight of the corresponding transition if it exists, or to* $\top$ *otherwise.*

Definition 43. A CFN $\mathcal{X}$ is *Berge-acyclic* [Allouche *et al.* 2016] iff its hypergraph is *Berge-acyclic* iff its incidence graph (*i.e.*, the factor graph) is acyclic.

In the CSP, it is known that if the decomposition of a global constraint is *Berge-acyclic*, then AC can be established on the constraint by enforcing it on the decomposition.

Virtual arc consistency links a CFN $\mathcal{C}$ with a CSP through $Bool(\mathcal{C})$. The networks have the same structures. Hence the following result [Allouche *et al.* 2016].

Let $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ be a cost function network. Let us assume that $\mathcal{C}$ with the global cost function $\mathrm{GCF}(\mathbf{S}, \mathcal{A}) \in \mathbf{C}$ decomposes into a Berge-acyclic network $\mathcal{C}' = (\mathbf{X} \cup \mathbf{X}', \mathbf{D} \cup \mathbf{D}', \mathbf{C} \cup \ \mathbf{C}' \setminus \{\ \mathrm{GCF}(\mathbf{S}, \mathcal{A})\})$. Then, enforcing VAC on $\mathcal{C}$ or on the decomposition $\mathcal{C}'$ yields the same lower bound $c_\varnothing$.

So, if the global cost function is decomposed into well-structured small arity functions, VAC can infer the same lower bound from the combination of small and simple functions than from a single global function that by itself gathers a lot of information.

For instance, the decomposition of the WRegular global cost function of example 19 satisfies the Berge acyclicity property. Therefore, enforcing VAC on the set of n ternary functions yields the same lower bound as dedicated algorithms for the global function would.

1.6 Other tools to improve search

As in the CSP case, the search can be quickened using additional tools and heuristics.

1.6.1 Branching heuristics

Regardless of the exploration strategy, the order in which variables and values are selected in the tree can strongly influence the search algorithm. On the one hand, branching on the easiest and most promising variables and values should lead faster to good solutions. However, it prevents the early

discovery of inconsistencies. On the other hand, branching on the hardest and worst variables and values leads to faster backtracking.

A trade-off [Marquis *et al.* 2020] consists of branching on the hardest variable, *i.e.*, the variable whose assignment has the strongest chance to lead to an inconsistency; and the most promising value for this variable, *i.e.*, the most likely to be part of the optimal solution. For instance, the variable involved in the most cost functions (with the highest number of neighbors) can be assigned the value with the smallest unary cost.

More advanced heuristics rely on the information from previous fails [Lecoutre *et al.* 2009, Boussemart *et al.* 2004], or from consistency enforcements, such as VAC [Trösser *et al.* 2020].

1.6.2 Soft neighborhood substitutability

Soft neighborhood substitutability (SNS) [De Givry *et al.* 2013] is a domain pruning strategy based on dominance analysis. It is a generalization of substitutability (or interchangeability) from the constraint framework [Freuder 1991, Lecoutre *et al.* 2012]. A value v is dominated by another value w when all solutions using w instead of v have a smaller cost. Therefore, value v can be removed from the domain (or assigned cost $\top$), as it will never be part of an optimal assignment.

Unlike local consistency enforcement, SNS does not preserve the equivalence, as value removal yields loss of solutions. However, the optimum is preserved.

Identifying dominated values is NP-hard. Sufficient conditions are defined, aiming at a trade-off between computation time and pruning efficiency.

1.6.3 Other strategies

The WCSP is NP-complete, so branch-and-bound combined with local consistencies might not always answer the query, especially as instances grow larger and larger. Other methods can be used, that will be not described here:

- Local search can be combined to global branch-and-bound search to improve sub-optimal solutions [Shaw 1998, Pisinger & Ropke 2010, Mladenović & Hansen 1997, Charpentier *et al.* 2018, Ouali *et al.* 2020].

- Look back techniques aim at backjumping as high as possible in the search tree, up to where the first decision was made that created a conflict [Dechter & Frost 2002].

Conclusion

In this chapter, we have described the necessary background on weighted constraint programming and optimization for this thesis work.

Cost function networks were introduced, with optimization algorithms. As we are solving an NP-complete problem, different strategies and heuristics are combined to speed the search, as branch-and-bound search, variable elimination, and local consistencies.

All these strategies have been implemented in the WCSP solver ToulBar2 (http://miat.inrae.fr/toulbar2/), an open-source C++ solver for cost function networks. It won several competitions, such as CPAI08 (Max-CSP) and UAI 2008, 2010, 2014 MAP task (probabilistic graphical models).

Chapter 2

Computational protein design

Contents

2.1 Proteins: from sequence to 3D-structure 54
2.1.1 The different structure levels of a protein 54
2.1.2 The protein data bank 59
2.1.3 Protein folding 59
2.1.4 Protein structure prediction: the folding problem . . . 60
2.2 Computational protein design 61
2.2.1 The ultimate goal of CPD 61
2.2.2 Input: the target structure 62
2.2.3 Search space: sequences and conformations 62
2.2.4 Objective: the energy function 64
2.2.5 Optimization methods for the CPD 65
2.2.6 In silico quality estimation of predicted sequences . . . 68
2.3 CPD as a WCSP 69
2.4 Success stories 71
2.5 Limits and challenges 73

Introduction

This chapter sets the framework for computational protein design (CPD). CPD seeks to answer the need for automatic methods for the conception of new proteins, with improved or new properties and functions. As we will see, it can be addressed with the CFN-based optimization strategies introduced in the previous chapter.

Section 2.1 provides basic notions on proteins, protein structure, and folding mechanisms. Then, Section 2.2 defines the computation protein design problem: the input instance, the search space, the optimization criterion, and existing solving methods. Section 2.3 shows how the CPD problem can

be expressed as an optimization problem on a cost function network. Finally, sections 2.4 and 2.5 present some examples of successful designs, and their limits, and the challenges that remain to be tackled.

2.1 Proteins: from sequence to 3D-structure

Proteins are complex molecules that perform a large variety of functions that are fundamental to all living organisms [Gu & Bourne 2009]. All these functions result from billions of years of evolution.

Several functional protein groups can be identified [Kaiser *et al.* 2007]. Some proteins have a *structural role*. They provide rigidity, by forming tissue, tendons, or muscle fiber: keratin makes up our hair and nails, and collagen our cartilage, bones, and skin. *Enzymes* act as catalysts. They speed up chemical reactions. For example, amylases and proteases are digestive enzymes. They break large molecules into smaller ones that can be absorbed. Some proteins act as hormones: they have a *regulatory function*. For instance, insulin regulates glycemia; the human growth hormone is also a protein. *Signaling proteins*, that pass messages from the exterior to the interior of the cell; *transport proteins*, such as hemoglobins; and *motor proteins*, that create motion, can also be mentioned.

Proteins are made of a chain of organic compounds, called amino acids. Most of these chains fold into a three dimensional (3D) structure. The protein function is directly linked to this structure.

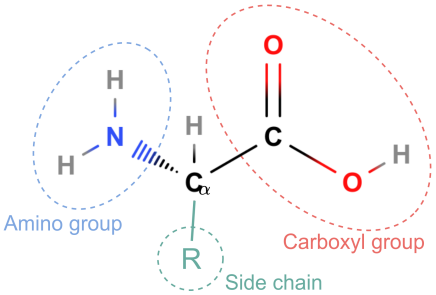
2.1.1 The different structure levels of a protein

2.1.1.1 Primary structure: amino acid sequence

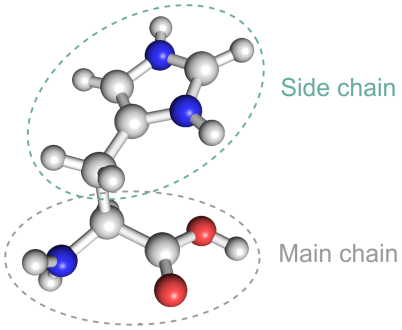
An amino acid is composed of an amino group (NH_2), a carboxyl group (COOH), and a central alpha carbon (C_α) to which are attached a hydrogen atom (H) and a side-chain (R). The amino and carboxyl groups, the alpha carbon, and the hydrogen atom constitute the *main chain* (see Figure 2.1).

The *side chain* R distinguishes the 20 different natural amino acid types (see Figure 2.2). It determines their specific physicochemical properties, such as hydrophobicity, polarity, or charge. These properties are given as an indication but they might depend on the environmental conditions. For example, histidine can be neutral polar and/or positively charged under different physiological conditions. Moreover, tryptophan and tyrosine both have a hydrophobic component but also a polar one.

Sidechains are not fixed, they can adopt different conformations. The degrees of freedom are described with dihedral angles, denoted χ. Depending



(a) Formula representation of the structure of an amino acid.



(b) Amino acid residue with its main chain and side chain

Figure 2.1: Two representations of an amino acid residue. The side chain R represents any of the 20 possible side chains.

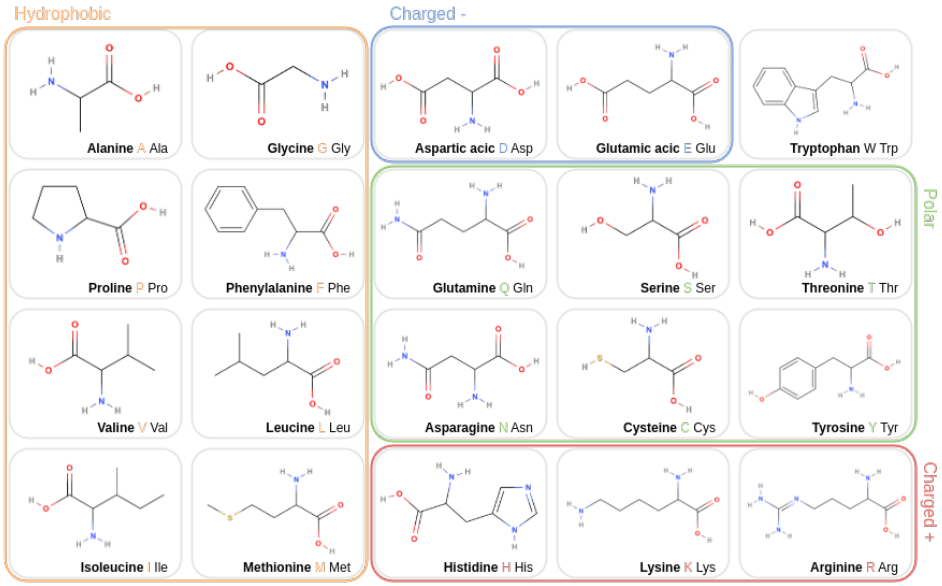
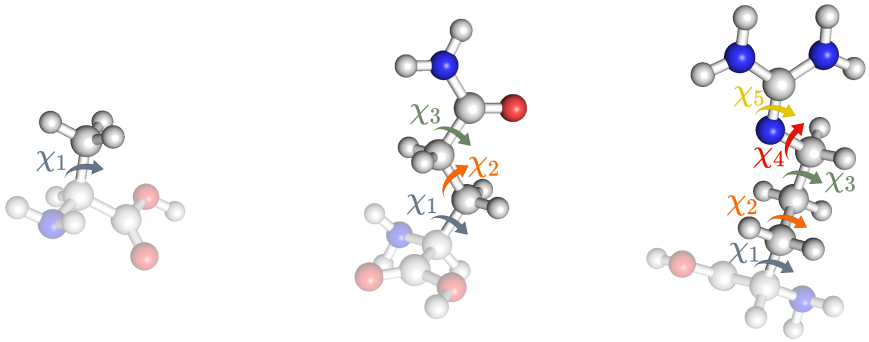


Figure 2.2: The 20 amino acids, with their full name, one letter, and three letter abreviations.



(a) (b) (c)

Figure 2.3: Dihedral angles χ_i of alanine (a), glutamine (b), and arginine (c)

on the amino acid type, the side chain can have from one to five χ-angles (see Figure 2.3).

2.1.1.2 The peptide bond

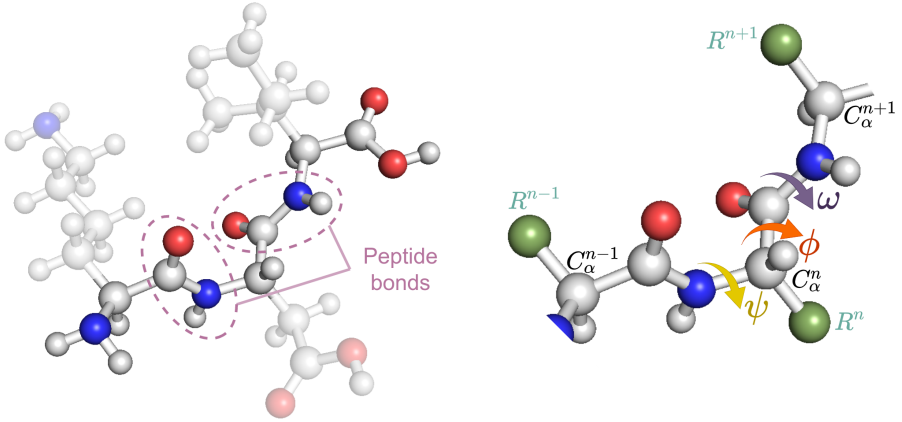
Two amino acids can form a *peptide bond* by the reaction of their respective carboxyl and amino group. Two or more amino acids linked this way are called a *peptide*, or a polypeptide (see figure 2.4a). A protein is formed by the linear succession of many amino acids, which constitutes its primary structure. The *backbone* of the protein corresponds to the sequence of amino acid main chains.

The peptide bond is planar and quite rigid, but it allows rotations of the backbone, measured by three dihedral angles for each residue (see Figure 2.4b). The ω-angle corresponds to the peptide bond between two residues. It usually takes the value π (*trans* conformation) but may be equal to 0 (*cis* conformation). The ϕ- and ψ-angles are located before and after the α-carbon, and truly contribute to the backbone flexibility.

For structural proteins, interaction forces between atoms of the amino acids constrain the peptide chain to fold into a 3D-structure.

2.1.1.3 The secondary structure

The secondary structure corresponds to the local conformations of the protein, *i.e.*, the structure of small parts of the chain. The peptide bond is rather rigid and the backbone movements are restricted to the rotations of the ϕ- and ψ-dihedral angles. The dominant secondary structural elements are the highly regular α-*helices* and the β-*sheets* (see Figure 2.5b), that are kept in shape by regular patterns of hydrogen bonds between backbone atoms.



(a) Chain of 3 amino acid residues, or tripeptide.

(b) Peptide bond dihedral angles.

Figure 2.4: The peptide bond

Other structures are observed, such as $3_{10}-$helices or β-bulges. Loops are the most unstructured elements. They often act as transitions between the regular regions. As they are highly flexible, some loops may be essential to the protein function.

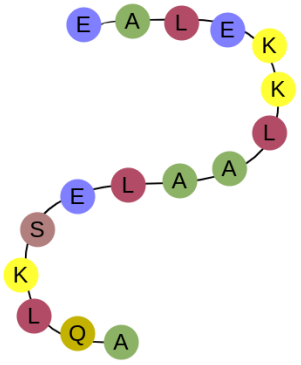
2.1.1.4 The tertiary and quaternary structures

The tertiary level corresponds to the global 3D-structure of the peptide chain. It is stabilized by atomic interactions, van der Walls potentials, electrostatic interactions, or hydrogen bonds. This final tertiary structure can be referred to as the *fold*, or the *domain* of the protein (see Figure 2.5c).

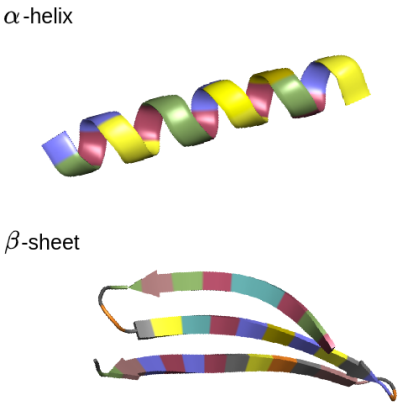
Some proteins, called *monomers*, consist of a single chain. There also exist proteins, that result from the combination of two or more chains. The quaternary structure describes the relative positions of these different domains (see Figure 2.5d).

2.1.1.5 Flexibility

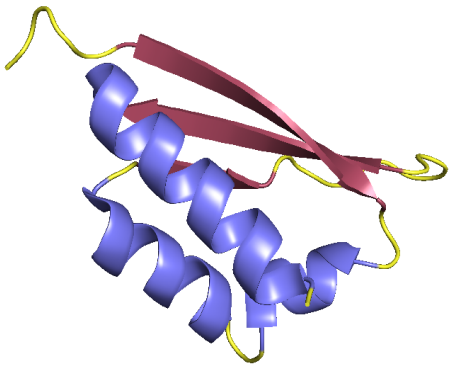
Structured proteins are by definition relatively stable in their adopted 3D-structure. However, atom bond lengths and valence and dihedral angles of the main and side chains provide the molecule with flexibility. These conformational changes range from side-chain small reorientation to wide movement of local regions in the protein structure. Unstructured secondary elements, such as loops, are very likely to move.



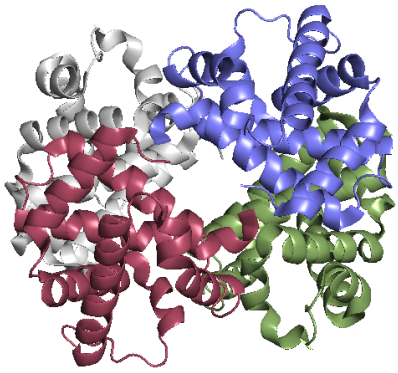
(a) Primary strcture



(b) Secondary structure



(c) Tertiary structure



(d) Quaternary structure

Figure 2.5: Primary, secondary, tertiary and quaternary levels that describe a protein 3D-structure.

This flexibility contributes largely to the protein functionality [Henzler-Wildman *et al.* 2007]. Although the static structure of a protein is often known (or sought), it is a dynamic molecule and the *set* of conformations it is prone to adopt might better capture its flexibility and functional properties.

2.1.2 The protein data bank

Experimentally determined protein structures are gathered in the *Protein Data Bank* [Bergman *et al.* 2016]. Most of these structures were identified through X-ray crystallography, nuclear magnetic resonance (NMR) spectroscopy [Gu & Bourne 2009], or cryogenic electron microscopy (cryo-EM) [Bhella 2019]. X-ray crystallography enables the high structure resolution of proteins of all sizes. However, the molecule needs to be crystallized, and the approach returns a static state structure. NMR spectroscopy identifies the structure of proteins directly in solution, thus having access to several conformational states. But it is limited to small molecules and has a low resolution. More and more structures are being identified through cryo-EM, as it requires small-sized, easily prepared samples and keeps structures in their native state. The main drawbacks are the relatively low resolution and the fact that this method is not fit for small molecules.

The *Protein Data Bank* (PDB) format [Berman *et al.* 2000] describes the Cartesian coordinates of all the atoms that are involved in a protein. At its creation in 1971, the PDB contained 7 structures. It has since then been growing exponentially. On November 6th, 2020, 149 550 protein structures were available.

The number of distinct *folds* among these determined structures is much less: 1487 on October 13th, 2020, according to the SCOP2 database [Murzin *et al.* 1995]. It means that many proteins with different functionalities have the same fold. This property may be of use for protein structure prediction (see Subsection 2.1.4) and protein design (see Section 2.2).

2.1.3 Protein folding

Given a protein sequence of amino acids, the space of possible conformations is gigantic. The *Levinthal paradox* observes that if a protein were to sample each conformation, folding would require a huge amount of time [Levinthal 1968]. Yet folding is a spontaneous, almost instant process, guided by interactions between amino acids, such as hydrophobic interactions, formations of hydrogen bonds, van der Wals forces, etc.

Proteins fold into low energy conformations [Anfinsen 1973], that ensure their thermodynamic stability. Evolution lead to funneled-shape conformation energy landscapes that enable proteins to fold rapidly into their native

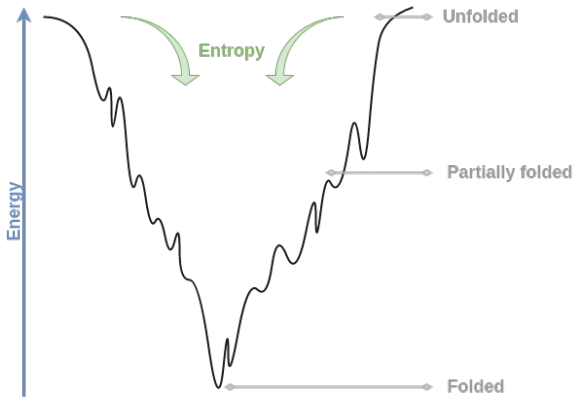


Figure 2.6: Funnel-shaped conformation energy landscape of a protein.

state through a series of intermediate states (see Figure 2.6).

2.1.4 Protein structure prediction: the folding problem

On the one hand, protein structure resolution methods are costly. On the other hand, DNA sequencing and protein-coding gene annotation techniques are easier and the number of available protein sequences, for example in the UniProt database [Consortium 2007], is still growing exponentially. On October 7th, 2020, the UniProt release indexed 195 104 019 protein sequences. This is more than one thousand times the number of resolved structures in the PDB.

Therefore, there has been a need for in silico protein structure prediction methods, to fill this *sequence-structure gap*. The task at hand is the inference of a protein 3D-structure from its amino acid sequence [Dill *et al.* 2008, Dill & MacCallum 2012].

Critical Assessment of protein Structure Prediction or CASP is a world-wide competition that takes place every two years [Kryshtafovych *et al.* 2019]. Participating research groups are provided with target sequences, whose structure is known but kept secret. The community can test and enhance their structure prediction methods on these sequences. Afterward, the actual structures are revealed.

Most successful strategies rely on the assumption that similar sequences fold into similar structures. The PDB is therefore searched for sequences with identified structures, that are similar to the target sequence. Prediction methods have therefore improved with the PDB expansion and the refinement of search and alignment tools. When no similar sequences are to be found, prediction algorithms rely on *fragment assembly*, *i.e.*, the occurrence of conserved amino acid patterns.

In 2018, CASP13 was dominated by AlphaFold [Senior *et al.* 2020], a neural network technology designed by DeepMind. In November 2020, the improved AlphaFold2 won CASP14, with nearly 90% prediction accuracy [Callaway 2020].

2.2 Computational protein design

Protein design aims at answering the need for the conception of new proteins, with enhanced or new characteristics. New amino acid sequences are sought, that will lead to molecules with the desired target properties.

In vitro protein design methods have been developed, as directed evolution [Joyce 2007, Romero & Arnold 2009], which mimics the natural evolution process, that optimized proteins to meet biological needs.

However, these non-computational experimental strategies are very hard and expensive and soon meet the search space size challenge. As there are 20 amino acid possibilities for each position, designing a 10-residue peptide requires the exploration of $20^{10} \approx 10^{13}$ sequences. Automatic methods are therefore needed, to fully explore and prune the sequence space, and identify sequences of interest for the target structure and properties.

2.2.1 The ultimate goal of CPD

Computational protein design aims at finding a *sequence* $\mathbf{s}$ that will adopt a *target structure*, that corresponds to *desired function* and *properties*. The sought sequence should therefore be stable in the target fold and suitable for the intended function. Stability and function suitability are expressed with an energy function E and a fitness function F:

$E(\tau, \mathbf{s}, \chi)$ describes the energy of the sequence $\mathbf{s}$ when it adopts the three dimensional conformation defined by the backbone dihedral angles $\tau = (\omega, \phi, \psi)$ and amino acid side-chain dihedral angles χ.

$F(\tau, \mathbf{s}, \chi)$ describes the suitability for purpose of the protein (affinity, catalysis, solubility, etc).

The task at hand is the prediction of a sequence-conformation $(\tau, \mathbf{s}, \chi)$ that:

- is able to perform the desired function, *i.e.*, has a high enough fitness

$$F(\tau, \mathbf{s}, \chi) > F_{\min}$$

- adopts the expected fold, *i.e.*, the dihedral angles τ, χ correspond to the minimum in the conformation energy landscape of the predicted protein sequence $\mathbf{s}$:

$$E(\tau, \mathbf{s}, \chi) = \min_{\tau', \chi'} E(\tau', \mathbf{s}, \chi')$$

Stated this way, the CPD problem is highly challenging due to the large discrete sequence space of $\mathbf{s}$, and the huge continuous conformation space of τ and χ. Several simplifications are made, relying on "native-like" structure knowledge, that aim at making the intractable conformational space reasonable. The following subsections describe precisely these simplifying assumptions, as well as the problem input, parameters, and objective function.

2.2.2 Input: the target structure

Natural proteins share a rather small number of distinct folds (see Subsection 2.1.2). So, existing folds are used to design new proteins. The target protein structure can be described with an existing protein backbone that was determined experimentally. If such a structure does not exist, it can be inferred from existing 3D-structures of similar proteins.

If *de novo* design of a completely new protein is addressed, the structure might be created by putting together 3D-fragments of existing proteins.

2.2.3 Search space: sequences and conformations

The design of a new sequence might either affect all the positions in the protein, or only one or several regions of the protein, that are key for the molecule function or properties. For each position in the protein sequence, the user states whether this position is:

Fixed The residue is fixed and rigid. The amino acid type and orientation are determined by the input target structure.

Flexible The residue type is fixed to the amino acid type of the input structure, but the side-chain might rotate. As described in Section 2.1, amino acid side-chains have from one to five degrees of freedom (the χ-dihedral angles).

Mutable Several amino acid types are allowed at the position, along with their different side-chain orientations.

As stated before, the discrete sequence-continuous conformation space has many hundreds of dimensions and is intractable. In practice however, observed configurations occupy a subspace of small volume, enabling simplifying assumptions on the conformational angles ϕ, ψ and χ.

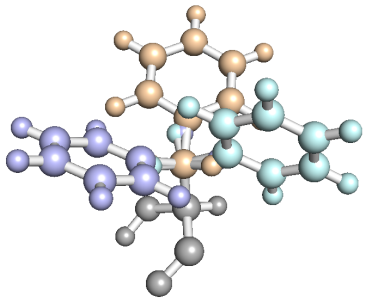


Figure 2.7: Three rotamers of the phenylalanine amino acid.

Fixation of the backbone Backbone angles τ are constrained by the inclination of the protein chain to adopt locally optimal secondary structures. Therefore, the target backbone can be supposed to be fixed.

The backbone angles τ remaining unchanged, the sequence-conformation $(\tau, \mathbf{s}, \chi)$ search space is characterized by two decision levels: the sequence space, which corresponds to all the possible sequences $\mathbf{s}$ enabled by the mutable positions, and the conformation space, which must be searched for the residue best side-chain positions χ.

The main drawback of this assumption is that we are no longer sure that the predicted sequence-conformation $(\tau, \mathbf{s}, \chi)$ will fold to the target structure, *i.e.*, there might exist a lower energy backbone configuration for the sequence $\mathbf{s}$:

$$E(\tau, \mathbf{s}, \chi) \stackrel{?}{=} \min_{\tau', \chi'} E(\tau', \mathbf{s}, \chi')$$

Moreover, the fixed backbone formulation prevents the representation of protein inherent flexibility requirements in the model.

Despite this, the fixed backbone assumption is very useful and is made by a lot of CPD methods [Samish 2017]. The resulting CPD formulation could be seen as the *inverse folding problem* [Yue & Dill 1992]: the prediction of the sequence $\mathbf{s}$ that will best fit the input structure τ.

Discretization of the side-chain conformation space An analysis of the residue side-chain positions χ in the PDB shows that most conformations are never explored in protein structures. The observed positions are frequent and gathered in clusters. The side-chain conformation space is therefore simplified to a discrete set of conformations that represent these clusters: the *rotamers* (see Figure 2.7).

Rotamers are indexed in *rotamer libraries*, such as the Dunbrack or the Penultimate library [Shapovalov & Dunbrack Jr 2011, Lovell *et al.* 2000].

In the Rosetta design software [Leaver-Fay *et al.* 2011b], the Dunbrack library creates domains of around 400 rotamers for mutable positions where all amino acid types are allowed. Designing a 10-residue peptide actually requires the exploration of $400^{10} \approx 10^{26}$ conformations.

2.2.4 Objective: the energy function

As stated before, given a backbone structure τ, the CPD problem seeks a stable and functional sequence-conformation $(\tau, \mathbf{s}, \chi)$, with good enough fitness F and minimum energy E.

The protein functionality is assumed to result from its conformation. The fitness function F is therefore neglected or included as additional terms in the energy function E. Sequences and conformations are therefore discriminated, depending on their energy when folded in the desired structure τ. The task at hand is the minimization of an objective function, that models the sequence-conformation energies on the target backbone. The optimal sequence is then likely (but not assured) to adopt the desired fold.

To model the energy, score functions are used. They can model physics-based energy terms as the energetic forcefields AMBER [Case *et al.* 2005] and CHARMM [Brooks *et al.* 2009]. They include atomic interactions, van der Walls potentials, electrostatic interactions, or hydrogen bonds. Score functions may also embed knowledge-based energy terms, that result from the statistical analysis of known protein structures. For instance, Rosetta ref2015 and beta_nov_16 energies [Leaver-Fay *et al.* 2011b] integrate rotamer probabilities, provided by the Dunbrack library. As peptides fold in aqueous environments, due to the hydrophobicity of their core amino acids, these energy functions need to represent the solvent. Although there exist computationally expensive *explicit* representations, they are scarcely used in CPD methods and energy functions mostly rely on an *implicit* representation of the solvent, in the form of an additional solvation energy term.

The energy function should provide computational efficiency, *i.e.*, the computation of conformation energy should be quick, while remaining accurate so as to predict relevant sequences. An *additive pairwise decomposable* approximation of the energy meets these needs [Samish 2017, Allouche *et al.* 2014]. The decomposable energy E of a sequence-conformation $\mathbf{r} = (r_1, \ldots, r_n)$ where r_i is the rotamer at the position i in the protein sequence can be written as follows:

$$E(\mathbf{r}) = E_\varnothing + \sum_{1 \leqslant i \leqslant n} E_i(r_i) + \sum_{1 \leqslant i < j \leqslant n} E_{ij}(r_i, r_j)$$

The term $E_\varnothing$ is a constant that captures interactions between the fixed parts of the backbones. For $1 \leqslant i \leqslant n$, the unary term E_i captures the

interactions between the rotamer r_i in position i and the backbone. For $1 \leqslant i < j \leqslant n$, the binary term E_{ij} captures the interactions between rotamers r_i and r_j in positions i and j respectively. These energy terms only vary with the rotamers because the backbone is assumed to be rigid.

Design dedicated software, such as OSPREY [Gainza *et al.* 2013] or Rosetta [Leaver-Fay *et al.* 2011b], compute all the constant, unary, and binary energy terms, for each rotamer and combination of rotamers for each position and pair of positions. Once computed, these values are stored in energy matrices. During the exploration of the sequence-conformation space, conformation energies are simply computed by summing the relevant energy terms.

Definition 44. The optimum conformation $(\tau, \mathbf{s}, \chi)$ is called the *global minimum energy conformation* (GMEC).

CPD methods aim at finding the global minimum energy conformation.

2.2.5 Optimization methods for the CPD

The previous Subsections detailed the elements of the computational protein design problem. The allowed amino acids at each position and the rotamer library define the sequence-conformation search space. This search space is equipped with an energy function E, that depends on the target protein structure. This energy is defined by pre-computed constant, unary, and binary energy terms: $E_\varnothing$, E_i for each position i, and E_{ij} for each pair of position $\{i, j\}$.

Let us denote $\mathcal{S}$ the set of allowed sequences, and for a sequence $\mathbf{s} \in \mathcal{S}$, $\mathcal{R}(\mathbf{s})$ the set of conformations (rotamers) that correspond to this sequence. The GMEC $\mathbf{s}^*$ is defined as follows:

$$\begin{aligned}\mathbf{s}^* &= \arg\min_{\mathbf{s}\in\mathcal{S}} \min_{\mathbf{r}\in\mathcal{R}(\mathbf{s})} E(\mathbf{r}) \\ &= \arg\min_{\mathbf{s}\in\mathcal{S}} \min_{\mathbf{r}\in\mathcal{R}(\mathbf{s})} \Big(E_\varnothing + \sum_i E_i(r_i) + \sum_{i,j} E_{ij}(r_i, r_j)\Big)\end{aligned}$$

Complexity Despite the simplifications enabled by the rigid backbone and discrete rotamers assumptions, the problem has been proven to be NP-hard [Pierce & Winfree 2002].

2.2.5.1 Stochastic methods

Simulated annealing The *Rosetta Molecular Modeling Suite* relies on Monte-Carlo Simulated Annealing [Van Laarhoven & Aarts 1987] to produce

sequences of low energy [Leaver-Fay *et al.* 2011b].

Starting from a given sequence-conformation, the algorithm generates small modifications of the conformation at each iteration. These alterations are accepted if they satisfy a Metropolis criterion: if they lower the energy or if the energy increase satisfies a given criterion that depends on the system temperature. High energy increases might be accepted in early iterations, to overcome the energetic barriers and avoid local minima. The temperature decreases progressively so that energy degradations are tolerated with lower and lower probability.

Genetic algorithms Genetic Algorithms [Jones 1994] are developed for example in the *EGAD* library [Chowdry *et al.* 2007]. They simulate the natural evolution process. The set of parent conformations are mutated. Some of these mutated sequences are selected, and recombined, forming the next set of parents. The algorithm iterates until equilibrium.

These stochastic methods have been improved to enable the relaxation of the discretization assumptions, thus the integration of flexibility information. Their main drawback is that finite-time convergence and quality are not guaranteed.

2.2.5.2 Exact search method

Dead-end elimination As stated in Chapter 1, dead-end elimination (DEE) [Desmet *et al.* 1992] reduces the search space, based on a dominance analysis. A rotamer r is dominated by a rotamer r' when any conformation with rotamer r does not have lower energy than the conformation with r' substituted for r. If so, rotamer r can be eliminated from the search space.

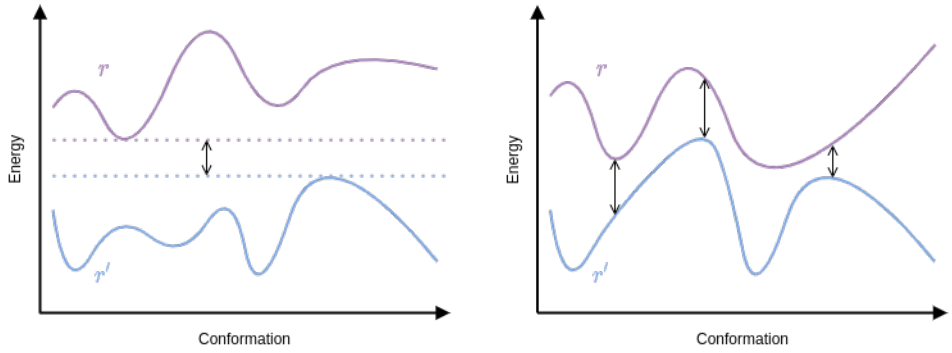
The Desmet criterion [Desmet *et al.* 1992] (see Figure 2.8a) is a sufficient condition for the dominance of rotamer r' over rotamer r in position i:

$$E_i(r) + \sum_{j \neq i} \min_s E_{ij}(r, s) > E_i(r') + \sum_{j \neq i} \max_s E_{ij}(r', s)$$

It was later improved by the Goldstein criterion [Goldstein 1994] (see Figure 2.8b):

$$E_i(r) - E_i(r') + \sum_{j \neq i} \min_s \left(E_{ij}(r, s) - E_{ij}(r', s)\right) > 0$$

These criteria have been further improved [Pierce *et al.* 2000, Looger & Hellinga 2001, Georgiev *et al.* 2006] and provide polynomial-time algorithms that prune dominated rotamers. As the design problem is NP-complete,



(a) Desmet criterion (b) Goldstein criterion

Figure 2.8: Dead-end elimination criteria.

these dominance analyses will not reach a solution, in general.

Dead-end elimination is used as a pre-processing step before an A* tree-search [Leach & Lemon 1998].

A* search A* is a best-first search algorithm, that expands nodes of the sequence-conformation search tree with the lowest energy. The first full sequence-conformation reached by the search is the GMEC.

The open-source design software OSPREY [Gainza *et al.* 2013] implements an exact search method based on dead-end elimination and A* tree-search.

Improvements in OSPREY 3.0 The latest OSPREY release offers powerful new features for computational protein design [Hallen *et al.* 2018, Hallen & Donald 2019]. Besides the engineering improvements that enable up to 10 times computation speed-ups, it embeds powerful tools that relax the strong discretization assumptions of the CPD problem formulation.

Small movements of the backbone dihedral angles are possible, and the structure can adjust to each mutant sequence. Rotamers are no longer discrete, but both discrete *and* continuous as small side-chain adjustments are enabled around the "ideal" rotamers. Several algorithms are available, that provide different complexity levels of flexibility, as iMinDEE [Georgiev *et al.* 2008], LUTE [Hallen *et al.* 2017], DEEPer [Hallen *et al.* 2013] and CATS [Hallen & Donald 2017]. Some of these procedures also provide a more precise, non-pairwise energy function.

2.2.6 In silico quality estimation of predicted sequences

Computational design methods provide the biologist with candidate sequences for a given target structure. These sequences can be produced experimentally and tested.

Even though experimental validations provide the best evaluation of the prediction methods, they are costly: they require equipment, skills, working force, and time.

In silico validation methods, that estimate the quality of predicted sequences are useful. They provide a cheap way of assessing prediction improvements. Moreover, given a target structure, the biologist might not be able to experimentally test all the predicted sequences. Computational validation methods discriminate candidate sequences for the most promising ones, worth synthesizing.

2.2.6.1 Similarity with natural sequences

When designing sequences for existing protein backbones, identified by X-ray crystallography or NMR spectroscopy, information about the target is provided by the existing native sequence: it fits the backbone perfectly and results from millions of years of selection pressure and evolution. It has therefore all the desired properties, for folding, solubility, functionality, etc.

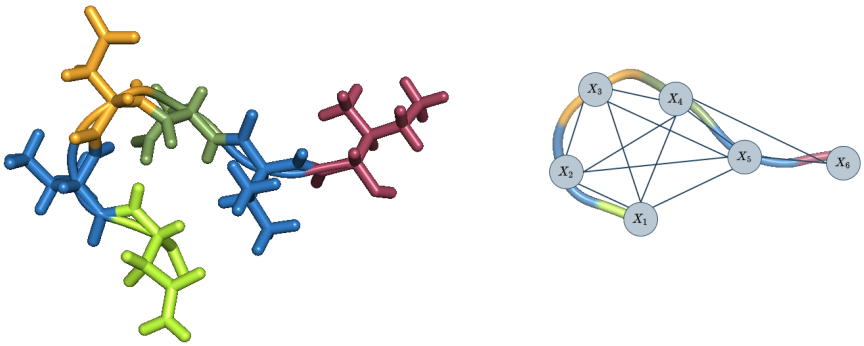
Assessing how much the designed sequence resembles the native provides an estimator of the prediction quality, that has been used for many protein design method evaluations.

Native sequence recovery (NSR) The simplest comparison between the design and the native is *native sequence recovery* (NSR). It corresponds to the proportion of positions where the design is identical to the native, *i.e.*, where sequences share the residue type.

Native sequence similarity recovery (NSSR) The NSR can be enhanced by taking into account *similarity scores* between the amino acid types. Such scores are provided by similarity matrices, like BLOSUM62 [Henikoff & Henikoff 1992].

The *native sequence similarity recovery* (NSSR) corresponds to the fraction of positions where the design and native have a positive similarity score.

Amino acid covariations More sophisticated methods exist to assess the similarity between sets of designed sequences and sets of natural sequences, that exploit statistical information inferred from alignments of



(a) Input existing backbone

(b) Resulting cost function network with 6 variables.

Figure 2.9: Cost function network representation of a CPD instance with 6 mutable residues.

natural sequences. For instance, [Ollikainen & Kortemme 2013] assesses the conservation of amino acid covariation motifs.

2.2.6.2 Folding tools

Computational folding methods predict the protein structure from a given amino acid sequence (see Subsection 2.1.4).

Candidate design sequences can be fed to a structure prediction algorithm like the Rosetta biased forward folding method [Marcos *et al.* 2017]. Matching predicted and target structures are encouraging regarding the design quality.

2.3 CPD as a WCSP

The computational protein design problem as introduced before is a combinatorial optimization problem, with discrete domain variables and an objective function expressed as a sum of small arity functions, stored in energy matrices. This corresponds exactly to the cost function network framework.

Therefore, the CFN formulation of CPD is straightforward [Traoré *et al.* 2013, Allouche *et al.* 2014, Traoré *et al.* 2016, Traoré *et al.* 2017], as shown hereafter (see Figure 2.9).

Given a CPD instance, let $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ be a cost function network with:

- Variables $\mathbf{X} = (X_1, \ldots, X_n)$ with one variable X_i for each flexible or mutable position i in the protein sequence
- Domains $\mathbf{D} = (\mathbf{D}_1, \ldots, \mathbf{D}_n)$ where the domain $\mathbf{D}_i$ of the variable X_i consists of the available rotamers at position i, *i.e.*, the amino acid types and their side-chain conformations.
- The cost functions are the empty-scoped, unary and binary energy terms:

$$\mathbf{C} = \{\ E_\varnothing\} \ \cup \{\ E_i, 1 \leqslant i \leqslant n\} \cup \{E_{ij}, 1 \leqslant i < j \leqslant n\}$$

Normalization CFN search algorithms are based on the assumption that non-empty-scoped cost functions are non-negative so that the empty-scoped cost function provides a lower bound on solution costs.

Energy terms can equally be positive or negative. A first resolution processing step is therefore the normalization of the problem: all cost functions $E_\mathbf{S}$ such that $|\mathbf{S}| > 0$ are substracted their minimum $\min_{\mathbf{t}\in\mathbf{S}} E_\mathbf{S}(\mathbf{t})$; this minimum is added to $E_\varnothing$. This reparameterization is an equivalence preserving transformation, *i.e.*, solution costs are unchanged. Resulting cost functions are non-negative., and $E_\varnothing$ is a lower bound on solution costs.

Search options The resolution of the WCSP corresponding to the CPD instance enforces virtual arc consistency as a preprocessing option. Dead-end elimination is often abandoned, as it empirically results in few rotamer eliminations, beyond those achieved by VAC.

The depth-first branch-and-bound search is performed, that exploits a tree-decomposition of the network. Existential directed arc consistency is maintained during search, as it is the consistency level that provides the best compromise between computational complexity and pruning efficiency.

SCP branching Variable and value ordering heuristics follow the "first-fail" method. A specific heuristic for CPD problems has been defined, called *SCP branching* [Traoré *et al.* 2016].

SCP branching split domain values so as to select an amino acid type. It does not branch on rotamers until all variable domains are reduced to a single amino acid. This method was shown to provide better results than other n-ary or binary branching methods.

Moreover, SCP branching first assigns a sequence, before branching on conformations. This allows for efficient enumeration of sequences, within an energy threshold of the GMEC: once a good enough conformation is reached for a given sequence, the conformation optimization stops, and the sequence

is added to the enumeration. This reduces drastically the computation time, as an exponential number of conformations are pruned.

Comparison with other search methods The CFN-based WCSP solving strategy implemented in Toulbar2 was compared on CPD instances to the Rosetta sampling protocol in [Simoncini *et al.* 2015]. This paper shows that although sampling methods have the ability to always provide a sequence, they do not offer any optimality proof or any reliable indication of the sequence quality. Moreover, to enhance the probability of finding a good quality sequence, a lot of sampling runs are needed, often requiring more computation time than the exact CFN solver. Finally, although sampling is sometimes able to reach the optimum, the energy gap between the best-sampled sequence and the actual optimum increases with the instance hardness.

Other exact algorithms can solve the WCSP or its reformulations, such as linear and quadratic programming solvers like Cplex [Cplex 2009], MaxSAT solvers like MiniMaxSat [Heras *et al.* 2008], CP solvers like MiniZinc [Marriott *et al.* 2008], or MAP/MRF solvers like Mplp [Sontag *et al.* 2012]. The previously introduced CPD-dedicated solver Osprey [Gainza *et al.* 2013], in its simple rigid-backbone discrete rotamers configuration, can also be mentioned, as well as the award-winning MAP/MRF solver Daoopt [Otten *et al.* 2012].

Daoopt has proven very efficient on genetic applications [Marinescu & Dechter 2006, Silberstein *et al.* 2013]. It relies on a depth-first AND/OR search [Dechter & Mateescu 2007] that exploits the network structure, and mini-bucket lower bounds [Dechter 1997, Dechter & Rish 2003]. The private version also benefits from a dual decomposition strengthening component [Otten *et al.* 2012]. CPD instances are particularly challenging for the AND/OR, mini-bucket bound strategy because the network is almost complete and domains are very large.

The previously mentioned solvers, as well as others, were rigorously compared to Toulbar2 on a benchmark of CPD instances [Traoré *et al.* 2013, Allouche *et al.* 2014]. Overall, the CFN-based approach, with tree-search and incremental local consistency enforcement was the most efficient.

2.4 Success stories

Since the 1980s, computational methods have been developed to assist in the design of new proteins. The first computer-aided protein design was performed in 1985 by [DeGrado *et al.* 1985]. A 17-residue sequence was produced, that folds into what was the tightest calmodulin-binding peptide.

More than a decade later, the "computational protein design" terminology was introduced. [Dahiyat & Mayo 1997] presented an automated method for the design of 20 out of 28 residues of a $\beta\beta\alpha$ motif, based on the structure of a zinc finger domain. The predicted sequence was validated experimentally. A year later, based on simple knowledge about stability, in the form of hydrophobic-polar residue patterns, [Harbury *et al.* 1998] presented the *de novo* design of α-helical bundle proteins, with a right-handed superhelical twist.

In 2003, a team of the Baker lab used iterations of sequence and structure predictions and achieved the *de novo* design of a new 93-residue protein, called Top7 [Kuhlman *et al.* 2003]. The predicted sequence was experimentally shown to fold to the initial target structure.

A few years later, the same lab presented two computational enzyme designs. [Jiang *et al.* 2008] describes the design of several retro-aldolases. [Röthlisberger *et al.* 2008] combined computational protein design with directed evolution methods to predict 8 new enzymes that catalyze the Kemp elimination, achieving a 10^5 rate enhancement.

[Chen *et al.* 2009] also tackled enzyme design. Their computationally predicted enzyme mutants were shown to provide a significant specificity improvement, thanks to exact structure-based redesign.

Although the bimolecular Diels-Alder reaction is not catalyzed by any naturally occurring enzyme, [Siegel *et al.* 2010] achieved the *de novo* design of enzymes catalyzing this reaction with high stereoselectivity and substrate specificity. The designs were validated experimentally.

The same year, the Mayo lab used computational design to generate experimental libraries of mutants. They were able to shift the wavelength of fluorescent proteins without affecting their stability [Chica *et al.* 2010].

In 2012, [Roberts *et al.* 2012] produced peptides able to inhibit a protein involved in cystic fibrosis, with higher affinity and increased binding efficiency, compared to state-of-the-art molecules.

More recently, aiming at the design of an α-transglucosylase, [Verges *et al.* 2015] used the Rosetta sampling method to predict 1515 candidate sequences, that were then further analyzed and discriminated, down to the successful identification of a particular mutant, with new substrate specificity. At the same time, [Hsia *et al.* 2016] presented a newly computationally designed icosahedral nanocage, that self-assembles from trimeric building blocks.

Two years ago, eight-bladed β-propeller protein designs were presented in [Noguchi *et al.* 2019]. β-propellers are made of pseudo-symmetrical subdomains called blades. Tako8 is a perfectly symmetrical eight-bladed β-propeller, that was designed from expert evolution and homology knowl-

edge. Two or four of the Tako8 blades were expressed separately, but they were not able to self-assemble to form the eight-bladed molecule. Tako8 was redesigned in Ika8, a fourfold-symmetrical protein, using cost function network optimization methods. Ika2 and Ika4, made of two of four blades of Ika8 respectively, happened to assemble spontaneously into a complete eight-bladed ring.

Exact optimization techniques reviewed in [Hallen & Donald 2019] predicted a neutralizing antibody against HIV, in collaboration with the NIH vaccine research center. This antibody shows strong activity against a broad range of HIV strains and is currently in clinical trials.

The examples given here are just a small part of all the computer-assisted successful designs. The interested reader might refer to [Samish 2017] for a more complete overview.

2.5 Limits and challenges

The CPD framework introduced in this chapter makes several simplifications. The target backbone is assumed to be rigid, the conformation space is discretized, and the optimization criterion is a pairwise-decomposable approximation of an energy function.

The first deficiency of the CPD framework is that it predicts sequences with low energy on the target structure. But there is no guarantee that no other better structure will exist for the designed sequences, and their fold might be completely different from the target.

Moreover, CPD works with an approximate representation of an otherwise highly intractable model, that fails at expressing some very desirable properties.

Proteins are dynamic and flexible functional molecules. Optimization of an energy criterion leads to the prediction of highly stable sequences, at the expense of flexibility and adaptability.

Overall, the main CPD challenge lies in the exploration of the exponential size conformation search space and the integration of protein good properties that govern stability, folding, functionality, binding, affinity, specificity, etc.

Conclusion

This chapter provided basic notions of protein structure and folding, that help to grasp the stakes of computational protein design. The precise CPD framework was described, as it is defined and solved in the Toulouse Biotechnology Institute (TBI) and Applied Mathematics and Computer Science (MIAT) teams.

A lot of challenges remain to be addressed and this thesis aims at providing new strategies to that end.

Chapter 3

Guaranteed solution diversity and quality

Contents

3.1 Ensuring diversity and quality of solutions 76
3.1.1 Mesuring diversity 77
3.1.2 Sets of diverse and good quality solutions 79
3.2 Relation with existing work 81
3.3 Representing the diversity constraint 82
3.3.1 Attempt at the Lagrangian relaxation 82
3.3.2 Using automata 83
3.3.3 Multi-valued decision diagram encoding cost function 85
3.3.4 Exploiting automaton function decomposition 88
3.3.5 Compressing the encoding 89
3.4 Combining the dissimilarity constraints 92
3.5 Greedy DiverseSeq 95
3.6 First experiments on Bayesian networks 97
3.7 Experiments on CPD instances 99

Introduction

Constraint programming is a modeling paradigm that is effective to build models of problems where solutions and non-solutions are separated by a set of properties, or constraints, that solutions must satisfy. Weighted constraint satisfaction captures costs and uncertainty when more gradual information is available. Cost function networks directly model a criterion and feasibility constraints as a sum of cost functions, where infinite costs capture infeasibility.

When the optimized criterion is not a perfect representation of the real problem, optimization queries are often insufficient to identify a single universally satisfying assignment. In CFNs, there are often additional criteria

that could not be captured in the model. In these cases, optimal solutions may not be the best "ìn practice". This can happen if the criterion has been learned from data or when it is an approximate representation of an otherwise highly intractable mathematical model. In these cases, it intuitively makes sense to produce a set of diverse low-cost solutions.

In computational protein design, simplifying assumptions are made: the backbone is assumed to be rigid, and side chain conformations discrete. Moreover, the optimized criterion only approximates the real criterion: the actual energy of the protein. Besides, it is centered on stability at the expense of functionality. This makes the protein design process unreliable. A library of highly relevant mutants should be produced, for further experimental testing. Ideally, this library should be a set of diverse and low energy solutions. The hope is that diversity will improve the likelihood that a working protein is found.

Because of their important applications, protein sequences can be subject to patents: in this case, a newly designed sequence must absolutely satisfy a Hamming distance constraint to existing patented sequences. Moreover, CPD input structures often come from existing, experimentally resolved proteins. In this case, a native sequence exists, that has a lot of desirable properties, as it is the result of billions of years of natural evolution. A maximum distance constraint to the native sequence can be used so that the designed sequence hopefully inherits some of the native good features.

In this Chapter, we are interested in producing a set of solutions that have low cost and that satisfy inter-solution distance constraints. Section 3.1 precisely defines the general problem we tackle; and Section 3.2 provides a quick tour of existing approaches. We represent distance constraints using automaton-based functions. Section 3.3 explores several encoding attempts of these functions. Then, in Section 3.4, we consider enforcing local consistencies on all dissimilarity requirements at once and show that it is NP-hard. Our attempts at using combined dissimilarity information remained unsuccessful. The final strategy is fully described in section 3.5, and evaluated in Sections 3.6 and 3.7 on a set of problems that include very large Bayesian network benchmark problems and CPD instances.

This work has been published in [Ruffini *et al.* 2019a]. Code is available at `https://github.com/toulbar2/toulbar2`.

3.1 Ensuring diversity and quality of solutions

Let us consider a cost function network $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ with variables $\mathbf{X} = (X_1, \ldots, X_n)$, domains $\mathbf{D} = (\mathbf{D}_1, \ldots, \mathbf{D}_n)$ and cost functions $\mathbf{C} \subset \{c_{\mathbf{S}} | \ \mathbf{S} \subset$

$\mathbf{X}\}$. The CFN defines the joint cost function:

$$\begin{array}{rccc} C_{\mathcal{C}}: & \mathbf{D_X} & \longrightarrow & \{0,\dots,\top\} \\ & \mathbf{t} & \longmapsto & \sum\limits_{c_{\mathbf{S}}\in\mathbf{C}}^{\top} c_{\mathbf{S}}(\mathbf{t}[\mathbf{S}]) \end{array}$$

3.1.1 Mesuring diversity

The task at hand is the production of a *set* of *diverse* and *low cost* solutions of $\mathcal{C}$. First, we need a measure of diversity between pairs of solutions, and among sets of solutions.

The simplest diversity measure between pairs of solutions is the *Hamming distance*, defined hereafter. It counts the number of variables in the solutions that take different values. In the CPD framework, sequence variables are added to the network, whose values are the amino acid types. The distance is measured on these variables and therefore corresponds to the number of positions that present different amino-acids, *i.e.*, the number of substitutions, or mutations.

Definition 45. Given a set of variables $\mathbf{S} \subset \mathbf{X}$ and two assignments $\mathbf{t}$ and $\mathbf{t}' \in \mathbf{D_S}$, the *Hamming distance* between $\mathbf{t}$ and $\mathbf{t}'$ is defined as follows:

$$H(\mathbf{t},\mathbf{t}') = \sum_{X_i\in\mathbf{S}} \mathbb{1}(\mathbf{t}[X_i] \neq \mathbf{t}'[X_i])$$

The Hamming distance can be generalized to take into account dissimilarity scores between values. The resulting distance is a semi-metric, defined as a sum of variable-wise dissimilarities, as follows:

Definition 46. Given a zero-diagonal symmetric positive matrix D, that defines value dissimilarities, and two assignments $\mathbf{t},\mathbf{t}' \in \mathbf{D_S}$, the *weighted-Hamming distance* between $\mathbf{t}$ and $\mathbf{t}'$ is defined as:

$$\mathtt{d}(\mathbf{t},\mathbf{t}') = \sum_{X_i\in\mathbf{D}} D(\mathbf{t}[X_i],\mathbf{t}'[X_i])$$

In computational biology, protein sequences are often compared using dedicated similarity matrices, such as the BLOSUM62 matrix [Henikoff & Henikoff 1992]. A protein similarity matrix S can be transformed into a dissimilarity matrix by:

$$\begin{array}{rccc} D: & [\![1,20]\!]^2 & \longrightarrow & \mathbb{N} \\ & (i,j) & \longmapsto & \dfrac{S(i,i)+S(j,j)}{2} - S(i,j) \end{array}$$

Example 20. *The tables below describe the BLOSUM62 similarity values for amino acids A, R, N, D, and C; and the corresponding dissimilarities:*

A	*4*				
R	*-1*	*5*			
N	*-2*	*0*	*6*		
D	*-2*	*-2*	*1*	*6*	
C	*0*	*-3*	*-3*	*-3*	*9*
	A	***R***	***N***	***D***	***C***

A	*0*				
R	*10*	*0*			
N	*12*	*11*	*0*		
D	*12*	*13*	*11*	*0*	
C	*13*	*17*	*18*	*18*	*0*
	A	***R***	***N***	***D***	***C***

Let us consider two random amino acid sequences $\mathbf{s}_1 = AARCDN$, $\mathbf{s}_2 = ARRCDA$. *There are 2 mutations between* $\mathbf{s}_1$ *and* $\mathbf{s}_2$, *i.e.,* $H(\mathbf{s}_1, \mathbf{s}_2) = 2$. *The BLOSUM62-based dissimilarity measure is:*

$$\mathtt{d}(\mathbf{s}_1, \mathbf{s}_2) = 0 + 10 + 0 + 0 + 0 + 12 = 22$$

Definition 47. Given a set $\mathbf{Z}$ of solutions, we define:

- its *average dissimilarity*: $\bar{\mathtt{d}}(\mathbf{Z}) = \dfrac{2}{|\mathbf{Z}|(|\mathbf{Z}|-1)} \displaystyle\sum_{\mathbf{t}\neq\mathbf{t}'\in\mathbf{Z}} \mathtt{d}(\mathbf{t}, \mathbf{t}')$
- its *minimum dissimilarity*: $\check{\mathtt{d}}(\mathbf{Z}) = \displaystyle\min_{\mathbf{t}\neq\mathbf{t}'\in\mathbf{Z}} \mathtt{d}(\mathbf{t}, \mathbf{t}')$

We are aiming at producing a set of diverse solutions, that provides a complete exploration of the search space, to enhance the probability of having a good solution "in practice". The *average dissimilarity* does not meet this need: a set of solutions might have a satisfying average dissimilarity value, with several occurrences of the same assignment, and one or a few very dissimilar ones. So, the *minimum dissimilarity* will be considered as the diversity measure throughout this chapter.

So, producing a set of *diverse* solutions requires that all solution pairs have their distance above a given threshold. This is guaranteed by constraint-encoding cost functions, that take their values in $\{0, \top\}$:

Definition 48. Given two sets of variables $\mathbf{S}, \mathbf{S}'$ of the same cardinality, a dissimilarity matrix D and a diversity threshold δ, we define the high-order cost function:

$$\begin{array}{rccl} \textsc{Dist}(\mathbf{S}, \mathbf{S}', D, \delta): & \mathbf{D}_\mathbf{S} \times \mathbf{D}_{\mathbf{S}'} & \longrightarrow & \{0, \top\} \\ & (\mathbf{t}, \mathbf{t}') & \longmapsto & \begin{cases} 0 & \text{if } sign(\delta).\mathtt{d}(\mathbf{t}, \mathbf{t}') \geqslant \delta \\ \top & \text{otherwise.} \end{cases} \end{array}$$

Allowing both positive and negative threshold δ enables the expression of minimum and maximum diversity constraints, through the Dist cost function. When $\delta > 0$, the cost function expresses a minimum dissimilarity value between the assignments $\mathbf{t}$ and $\mathbf{t}'$:

$$\textsc{Dist}(\mathbf{t}, \mathbf{t}', D, \delta) = 0 \Leftrightarrow \mathtt{d}(\mathbf{t}, \mathbf{t}') \geqslant \delta$$

If $\delta < 0$, the cost function represents the fact that $\mathbf{t}$ and $\mathbf{t}'$ must be similar, with a dissimilarity lower than the absolute value of δ:

$$\textsc{Dist}(\mathbf{t}, \mathbf{t}', D, \delta) = 0 \Leftrightarrow -\mathtt{d}(\mathbf{t}, \mathbf{t}') \geqslant \delta \Leftrightarrow \mathtt{d}(\mathbf{t}, \mathbf{t}') \leqslant -\delta = |\delta|$$

Example 21. *Let us consider a CFN $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ encoding a CPD instance. Let us assume that the following sequence information are available:*

- *A native sequence $\mathbf{s}_{nat}$ is known for the target backbone. The biologist wants the number of mutations of the design from the native to be less than δ_{nat}.*
- *A patented sequence $\mathbf{s}_{pat}$ exists, and sequences with less than δ_{pat} mutations from the patent are forbidden.*

The distance here is the Hamming measure H. The CFN $\mathcal{C}$ can be transformed into $\mathcal{C}' = (\mathbf{X}', \mathbf{D}', \mathcal{C}')$, to embed the patented and native sequences, as follows:

Variables *We add sequence variables to the network: $\mathbf{X}' = \mathbf{X}^{seq} \cup \mathbf{X}$.*

Domains $\mathbf{D} = \mathbf{D}^{seq} \cup \mathbf{D}$ *where the domain $\mathbf{D}_i^{seq}$ of X_i^{seq} is the set of available amino acid types in position i.*

Constraints *The new set of cost functions $\mathbf{C}'$ is made of the initial functions $\mathbf{C}$; sequence constraints, that ensure that X_i^{seq} is the amino acid type of rotamer X_i; the diversity constraint-encoding cost functions:* $\textsc{Dist}(\mathbf{X}^{seq}, \mathbf{s}_{nat}, H, -\delta_{nat})$ *and* $\textsc{Dist}(\mathbf{X}^{seq}, \mathbf{s}_{pat}, H, \delta_{pat})$

3.1.2 Sets of diverse and good quality solutions

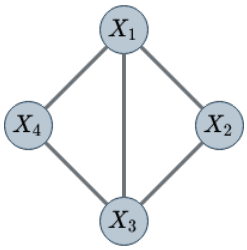
The problem of producing a set of diverse and good quality solutions, *i.e.*, such that all pairs of solutions satisfy the diversity constraint, and the solutions have minimum cost, can be expressed as follows:

Definition 49 (DiverseSet)**.** Given a dissimilarity matrix D, an integer M and a dissimilarity threshold δ, the problem $\textsc{DiverseSet}(\mathcal{C}, D, M, \delta)$ consists in producing a set $\mathbf{Z}$ of M solutions of $\mathcal{C}$ such that:

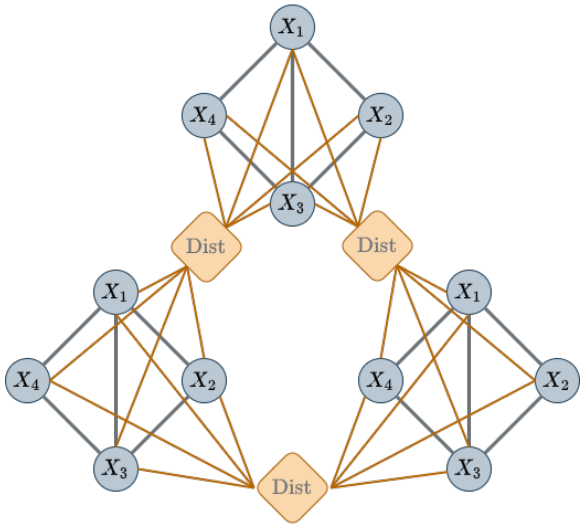
Diversity For all $\mathbf{t} \neq \mathbf{t}' \in \mathbf{Z}$, $\mathtt{d}(\mathbf{t}, \mathbf{t}') \geqslant \delta$, *i.e.*, $\textsc{Dist}(\mathbf{t}, \mathbf{t}', D, \delta) = 0$.

Quality The solutions have minimum cost, *i.e.* $\sum_{\mathbf{t} \in \mathbf{Z}}^{\top} C_{\mathcal{C}}(\mathbf{t})$ is minimum.

For a CFN $\mathcal{C}$ with n variables, solving DiverseSet requires to simultaneously decide the value of nM variables. It can be solved by making M copies of $\mathcal{C}$ with variable sets $\mathbf{X}^1$ to $\mathbf{X}^M$ and adding $\frac{M.(M-1)}{2}$ constraints



(a) A CFN $\mathcal{C}$ with 4 variables



(b) Three copies of $\mathcal{C}$ and pairwise diversity cost functions, represented as factors.

Figure 3.1: A CFN $\mathcal{C}$ and the CFN corresponding to DiverseSet$(\mathcal{C}, D, 3, \delta)$, the problem of computing 3 diverse solutions of $\mathcal{C}$.

Dist$(X^i, X^j, D\delta)$ for all $1 \leqslant i < j \leqslant M$. If the upper bound $\top$ is finite, all its occurrences must be replaced by $M.(\top - 1) + 1$. An example of a DiverseSet network for a simple CFN is represented in Figure 3.1. The number of variables being multiplied by the number of wanted solutions, this problem is very hard and expensive to solve even on tiny problems, as shown in Section 3.6.

Therefore, we decided to solve a relaxed version of DiverseSet . An interative algorithm provides a greedy approximation of the problem:

Definition 50 (DiverseSeq)**.** Given a dissimilarity matrix D, an integer M and a dissimilarity threshold δ, the set of assignments $\mathbf{Z}$ of DiverseSeq$(\mathcal{C}, D, M, \delta)$ is built recursively:

- The first solution $\mathbf{Z}[1]$ is the optimum of $\mathcal{C}$
- When solutions $\mathbf{Z}[1..(i-1)]$ are computed, $\mathbf{Z}[i]$ is such that for all $1 \leqslant j < i$, Dist$(\mathbf{Z}[i], \mathbf{Z}[j], D, \delta) = 0$ and $\mathbf{Z}[i]$ has minimum cost.

  That is, $\mathbf{Z}[i]$ is the minimum cost solution, among assignments that are at distance at least δ from all the previously computed solutions.

The set of solutions DiverseSeq is only an approximation of DiverseSet . However, it requires optimally assigning n variables M times, instead of $n.M$ variables. Given the NP-hardness of the WCSP, computing the

greedy approximation may be exponentially faster than exactly solving the problem while still providing guarantees that distance constraints are satisfied together with a weakened form of optimality, which is conditional on previous solutions.

3.2 Relation with existing work

In the case of Boolean functions, [Hebrard *et al.* 2005] considers the optimization of the solution set cardinality M or the diversity threshold δ using average or minimum dissimilarity. The authors prove that enforcing Arc Consistency on a constraint requiring sufficient average dissimilarity $\bar{d}$ is polynomial but NP-complete for minimum dissimilarity $\check{d}$ and evaluate an algorithm for incremental production of a set maximizing $\bar{d}$. [Hebrard *et al.* 2007] and [Hadžić *et al.* 2009] later addressed the same problems using global constraints and knowledge compilation techniques. More recently, [Petit & Trapp 2015] proposed a COP approach to provide diverse high-quality solutions. Their approach however trades diversity for quality.

The idea of producing diverse solutions has also been explored in the related area of discrete stochastic Graphical Models (such as Markov Random Fields), which are closely related to CFNs. [Batra *et al.* 2012] exploits the fact that the Lagrangian relaxation of minimum dissimilarity constraints adds only unary cost functions. This approach can be adapted to cost function networks, as detailed in Subsection 3.3.1. This was extended in [Prasad *et al.* 2014] using high-order functions (or potentials) to *approximately* optimize a trade-off between diversity and quality. More recently, [Kirillov *et al.* 2015] addressed the DiverseSet problem, but using optimization techniques that provide no guarantee.

In the end, we observe that none of these approaches simultaneously provides guarantees on quality and diversity. Closest to our target, [Chen *et al.* 2013] considered the problem of incrementally producing the set of the best M δ-modes of the joint distribution $J_N(X)$.

Definition 51 ([Chen *et al.* 2018])**.** A solution $\mathbf{t}$ is said to be a δ-mode iff there exists no better solution than $\mathbf{t}$ in the Hamming ball of radius δ centered in $\mathbf{t}$ (implying that $\mathbf{t}$ is a local minimum).

In [Chen *et al.* 2013, Chen *et al.* 2014, Chen *et al.* 2016, Chen *et al.* 2018], an exact dynamic programming algorithm, combined with an A* heuristic search and tree-decomposition was proposed to exactly solve this problem with the Hamming distance. This algorithm relies however on NP-complete lower bounds and is restricted to a fixed variable order. It however provides a diversity guarantee: indeed, a δ-mode will always be *strictly* more than δ away from another one and will be produced by greedily solving DiverseSeq.

Theorem 3. Given a cost function network $\mathcal{C}$, a diversity threshold δ, and $D = H$ the Hamming dissimilarity matrix, for any δ-mode $\mathbf{t}$, there exists a value M' such that the solution of DiverseSeq$(\mathcal{C}, H, M', \delta + 1)$ contains $\mathbf{t}$.

Proof. If a δ-mode $\mathbf{t}$ is not in the solution of DiverseSeq$(\mathcal{C}, H, M', \delta + 1)$, this must be because it gets forbidden by a Dist constraint. Consider the iteration i which forbids $\mathbf{t}$ for the first time: a solution with a cost lower than the cost of $\mathbf{t}$ was produced (else $\mathbf{t}$ would have been produced instead) but this solution is strictly less than $\delta + 1$ away from $\mathbf{t}$ (since $\mathbf{t}$ gets forbidden). But this contradicts the fact that $\mathbf{t}$ is a δ-mode. □

For a sufficiently large M, the sequence $\mathbf{Z}$ of DiverseSeq$(N, H, M, \delta+1)$ solutions will therefore contain all δ-modes and possibly some extra solutions. Interestingly, it is not difficult to separate modes from non-modes.

Theorem 4. 1. Any assignment $\mathbf{t}$ of a CFN $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ is a δ-mode iff it is an optimal solution of the CFN $(\mathbf{X}, \mathbf{D}, \mathbf{C} \cup \{\text{Dist}(\mathbf{X}, \mathbf{t}, H, -\delta)\})$
2. For bounded δ, this problem is in P.

Proof. 1. The function Dist$(\mathbf{X}, \mathbf{t}, H, -\delta)$ restricts $\mathbf{X}$ to be within δ of $\mathbf{t}$. If $\mathbf{t}$ is an optimal solution of $(\mathbf{X}, \mathbf{C} \cup \{\text{Dist}(\mathbf{X}, \mathbf{t}, H, -\delta\}))$ then there is no better assignment than $\mathbf{t}$ in the δ-radius Hamming ball and $\mathbf{t}$ is a δ-mode.
2. For bounded δ, a CFN with n variables and at most d values in each domain, there is $O((nd)^\delta)$ tuples within the Hamming ball, because from $\mathbf{t}$, we can pick any variable (n choices) and change its value (d choices), δ times. Therefore the problem of checking if $\mathbf{t}$ is optimal is in P. □

3.3 Representing the diversity constraint

The key to guaranteeing quality and diversity of solutions in a cost function network is the dissimilarity cost function Dist. Given a complete assignment $\mathbf{t}$, a dissimilarity D and a threshold δ, we need to encode the global diversity constraint Dist$(\mathbf{X}, \mathbf{t}, D, \delta)$.

3.3.1 Attempt at the Lagrangian relaxation

Within the Markov random field framework, [Batra *et al.* 2012] exploits the Lagrangian relaxation of dissimilarity constraints. This idea is tempting because it only adds unary costs to the existing network. The correspondence between MRFs and CFNs makes the adjustment of the method to our problem straightforward.

Given a CFN $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$, a dissimilariy matrix D, a diversity threshold δ, and solutions $\mathbf{Z}[1], \ldots, \mathbf{Z}[M-1]$, the task at hand is the following:

$$\mathbf{Z}[M] = \arg\min_{\mathbf{t} \in \mathbf{D_X}} C_{\mathcal{C}}(\mathbf{t}) \text{ subject to } \begin{cases} \mathtt{d}(\mathbf{t}, \mathbf{Z}[1]) \geqslant \delta \\ \vdots \\ \mathtt{d}(\mathbf{t}, \mathbf{Z}[M-1]) \geqslant \delta \end{cases}$$

The dissimilarity constraints are global cost functions on the network, that threaten to harden its resolution. They can be dualized, leading to the *Lagrangian relaxation* of the problem [Bertsekas *et al.* 1999]:

$$q(\lambda) = \min_{\mathbf{t} \in \mathbf{D_X}} \mathbf{L}(\mathbf{t}, \lambda) = \min_{\mathbf{t} \in \mathbf{D_X}} \mathbf{C}_{\mathcal{C}}(\mathbf{t}) - \sum_{\mathbf{i}=\mathbf{1}}^{\mathbf{M-1}} \lambda_{\mathbf{i}} (\mathtt{d}(\mathbf{t}, \mathbf{Z}[\mathbf{i}]) - \delta)$$

The parameters $\lambda = (\lambda_{\mathbf{i}})_{\mathbf{1} \leqslant \mathbf{i} \leqslant \mathbf{M-1}}$ are the Lagrange multipliers. They correspond to the penalty imposed on diversity constraint violations.

For all λ, the value $q(\lambda)$ is a lower bound on the cost of the optimum solution $\mathbf{Z}[M]$. Maximizing $q(\lambda)$, in order to find the tightest lower bound, corresponds to tuning λ: constraint violations must be penalized enough so that the dissimilarities are satisfied, but not too much, to not miss potential solutions.

The function q is piece-wise linear and concave in λ, so it has a subgradient:

$$\nabla q(\lambda) = -\left[\mathtt{d}(\mathbf{t}^*, \mathbf{Z}[\mathbf{i}]) - \delta\right]_{\mathbf{1} \leqslant \mathbf{i} \leqslant \delta} \text{ where } \mathbf{t}^* = \arg\min_{\mathbf{t}} \mathbf{L}(\mathbf{t}, \lambda)$$

Therefore, the function q can be maximized through a subgradient ascent. At each step, the value of $q(\lambda) = \min_{\mathbf{t}} \mathbf{L}(\mathbf{t}, \lambda)$ is computed by optimizing the CFN $\mathcal{C}$ with modified unary costs, and λ is updated, by the addition of the subgradient value, multiplied by a step size.

Various step sizes can be used [Bertsekas *et al.* 1999]. The main result being that for diminishing step size and length, the subgradient method converges to $q^* = \max_\lambda q(\lambda)$ [Bertsekas *et al.* 1999].

Unfortunately, q^* is only a lower bound on the optimum cost $C_{\mathcal{C}}(\mathbf{Z}[M])$. The *duality gap* is the difference between this lower bound and the actual optimum. With the Hamming distance, the *duality gap* is non zero, and preliminary tests on CPD instances showed that obtained solutions were neither optimal nor satisfied the dissimilarity requirements.

3.3.2 Using automata

Relaxing the constraints does not yield satisfying results. Given a solution $\mathbf{t}$, a dissimilarity matrix D and a diversity threshold $\delta > 0$, the cost function

$\textsc{Dist}(\cdot, \mathbf{t}, D, \delta)$ needs to be added to the cost function network. Note that the function involves all the network variables: it is a *global cost function*.

As explained in Chapter 1, CSP constraints are defined by the finite set of authorized tuples. This set defines a regular language, that can be encoded into a finite state automaton [Pesant 2004]. Inside a CFN, equivalence preserving transformations used to enforce local consistencies can shift costs between various functions. Therefore, any function initially defined as a constraint will quickly be transformed into a function using costs other than just 0 and $\top$. Weighted automata can be used to define the weighted regular language of all tuples with their associated cost. A weighted automaton $\mathcal{A} = (\Sigma, \mathbf{Q}, \Delta, Q_0, \mathbf{F})$ encoding $\textsc{Dist}(\mathbf{X}, \mathbf{t}, D, \delta)$ can be defined as follows:

- The alphabet is the set of possible values, *i.e.*, the union of the variable domains $\Sigma = \bigcup_{i=1}^{n} \mathbf{D}_i$
- The set of states $\mathbf{Q}$ gathers $(\delta + 1) \cdot (n + 1)$ states denoted q_i^d:

  $$\mathbf{Q} = \{q_i^d |\ 0 \leqslant i \leqslant n, 0 \leqslant d \leqslant \delta\}$$

  that represent the fact that the first i values of $\mathbf{X}$ have distance d to the first i values of $\mathbf{t}$. For $d = \delta$, automaton state q_i^δ represents the fact that the first i values of $\mathbf{X}$ have distance $\geqslant \delta$ to the first i values of $\mathbf{t}$.
- In the initial state, no value of $\mathbf{X}$ has been read, and the dissimilarity is 0:

  $$Q_0 = q_0^0$$
- The assignment is accepted if it has dissimilarity from $\mathbf{t}$ higher than the threshold δ, hence the accepting state:

  $$\mathbf{F} = \{q_n^\delta\}$$
- For every value r of X_i, the transition function $\Delta : \mathbf{Q} \times \Sigma \times \mathbf{Q}$ defines a 0-cost transition from q_i^d to $q_{i+1}^{\min(d+D(r,\mathbf{t}[i+1]),\delta)}$. All other transitions have infinite cost $\top$.

This weighted automaton contains $O(n \cdot (\delta + 1) \cdot d)$ finite cost transitions, were d is the maximum domain size. An assignment $\mathbf{t}'$ of $\mathbf{X}$ is accepted if and only if $\mathtt{d}(\mathbf{t}', \mathbf{t}) \geqslant \delta$; and the automaton represents the $\textsc{Dist}$ cost function. An example of a $\textsc{Dist}$ encoding automaton is given in Figure 3.2.

3.3.2.1 Tractable-projection safety

The weighted automaton representation of the $\textsc{Dist}$ global cost function shows that it is compatible with equivalence preserving transformations

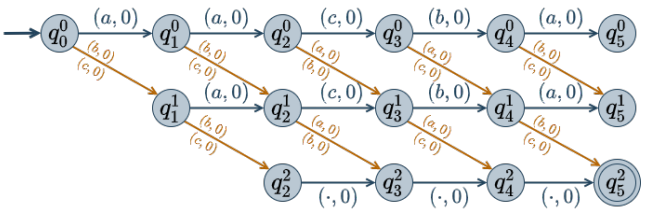


Figure 3.2: Weighted automaton representing $\textsc{Dist}(\mathbf{X}, \mathbf{t}, H, \delta)$ where $\mathbf{X}$ is a set of 5 variables, with domains $\mathbf{D}_i = \{a, b, c\}$, $\mathbf{t} = aacba$, H represents the Hamming distance, and δ is set to 2. State q_i^d means that values $X_1 \dots X_i$ are such that $H(X_1 \dots X_i, t[X_1 \dots X_i]) = d$ (or $\geqslant \delta$ if $d = \delta$). A labeled arrow $q \xrightarrow{(v,w)} q'$ means $\Delta(q, v, q') = w$, *i.e.*, there is a transition from q to q' with value v and weight w.

and local consistency enforcement algorithms. More precisely, we have the following result.

Theorem 5. The global cost function Dist is *tractable projection safe*.

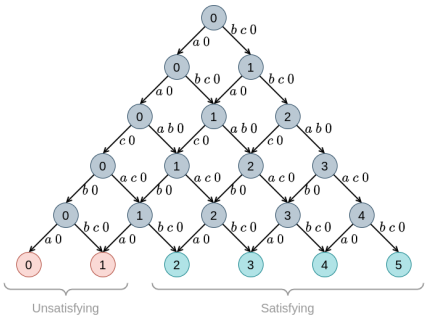
Proof. A sufficient condition for *tractable projection safety* is through function decomposition: if the global function decomposes into a Berge-acyclic network, then it is tractable projection safe.

Dist can be expressed as a weighted automaton, that itself decomposes as a Berge-acyclic network [Allouche *et al.* 2016]. Therefore, Dist is tractable projection safe. □

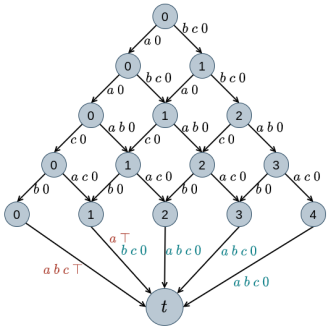
3.3.3 Multi-valued decision diagram encoding cost function

Following the [Pesant 2004] strategy for arc consistency enforcement on the regular membership constraint, a weighted automaton can be propagated through a linear Forward/Backward algorithm. First, the automaton has to be layered, *i.e.*, expressed as an equivalent multi-valued decision diagram (MDD) [Bergman *et al.* 2011, Bergman *et al.* 2016].

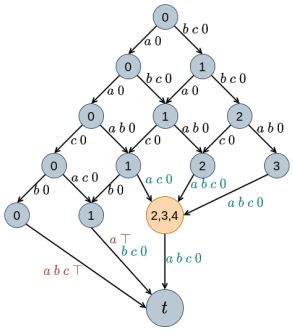
Representing the dissimilarity cost function as a Multivalued decision diagram In this work, a multivalued decision diagram is a *layered automaton*, with $n+1$ layers. The first layer L_0 consists of a single node: the source s. Then, for $1 \leqslant L_i \leqslant n$, the layer L_i is associated with the variable X_i. Edges connect nodes in a layer L_i to nodes in the layer L_{i+1} and are labeled with a value and a cost. An edge from layer L_i to layer L_{i+1}, labeled with value v and cost c means that variable X_i is assigned value v. For a given assignment, the MDD cost is the sum of the assignment edge costs.



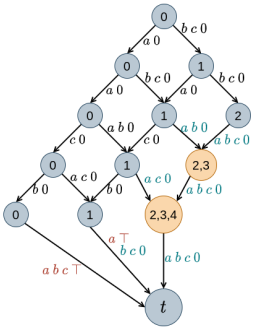
(a) The layers are computed iteratively.



(b) Nodes in the last layer are merged and edges are redirected and relabeled if needed.



(c) Equivalent nodes in layer 4 can be merged.



(d) Equivalent nodes in layer 3 can be merged.

Figure 3.3: Example of weighted MDD construction, to represent the dissimilarity cost function $\textsc{Dist}(\mathbf{X}, \mathbf{t}, H, \delta)$, where $\mathbf{X}$ is a set of 5 variables with domains $\mathbf{D}_i = \{a, b, c\}$, H represents the Hamming distance, and $\delta = 2$. Each node u is labeled with its state I_u.

Each node u in a layer L_i is associated with a state I_u, that corresponds to the state of the automaton reached after reading $X_1 \dots X_i$. For the dissimilarity representation from a given assignment $\mathbf{t}$, the state of the node u is:

$$I_u = \mathtt{d}(X_1 \dots X_i, \mathbf{t}[X_1 \dots X_i])$$

The MDD is built layer after layer. When the last layer L_n is computed, it consists of several nodes, with states that correspond to accepted or forbidden assignments. These nodes are merged into the terminating node t, and edges are redirected to t. If they previously lead to a forbidden node, they are relabeled with the infinite cost $\top$.

Once the last layer is computed, the layers can be compressed in a backward manner, by merging all *equivalent* nodes, *i.e.*, nodes with the same

outgoing edges (same values, weights, and target node). An example of an MDD construction, for a dissimilarity cost function representation, is given in Figure 3.3.

Note that MDDs and automata are equivalent [Hadzic *et al.* 2008], hence the similarity between the dissimilarity encoding MDD and weighted automaton. Moreover, the MDD computation can be extended to all weighted automata. The L_i node states are simply set to the automaton state, or set of states, reached after reading the first i-th values.

Enforcing strong $\varnothing$-inverse consistency The forward/backward algorithm presented in [Pesant 2004] can be adapted to the weighted framework, for the $\varnothing$-inverse consistency enforcement on cost functions expressed as multivalued decision diagrams.

The data structure and mechanisms of [Pesant 2004] are kept, to deal with domain wipeouts and inconsistencies (infinite costs). $\varnothing$-inverse consistency enforcement relies on the computation of the minimum function cost, among all assignments of the scope variables (see Chapter 1 for more details). This minimum cost is then subtracted from the cost function and added to the empty-scoped cost function.

In order to compute this minimum, let us define, for each layer L_i and each node $u \in L_i$, the quantity $\alpha[u]$ as the minimum path cost from the source s to u. The α values can be computed for all the nodes, in linear time in the size of the MDD. The α values in layer L_{i+1} only depend on the α values in layer L_i and edge costs between L_i and L_{i+1}. When a domain is altered, α needs only to be corrected from the corresponding layer. The minimum cost is then $\alpha[t]$.

Therefore, $\varnothing$-inverse consistency can be enforced in linear time in the size of the MDD.

Enforcing arc consistency In order to enforce arc consistency, let us define $\beta[u]$ for each node $u \in L_i$, as the minimum path cost from node u to the terminating node t. The β values can be computed as the α values, but in a backward manner.

Enforcing arc consistency requires the minimum function cost, with a variable X_i assigned to a value v. This minimum can be inferred as follows: all assignments $\mathbf{t}$ such that $\mathbf{t}[X_i] = v$ correspond to a path in the MDD that use an edge from L_i to L_{i+1} labeled with value v. Moreover, the minimum cost path that goes through the edge $u \overset{(v,c)}{\rightarrow} u'$ corresponds to the minimum cost path from s to u, added to the edge cost c, added to the minimum cost

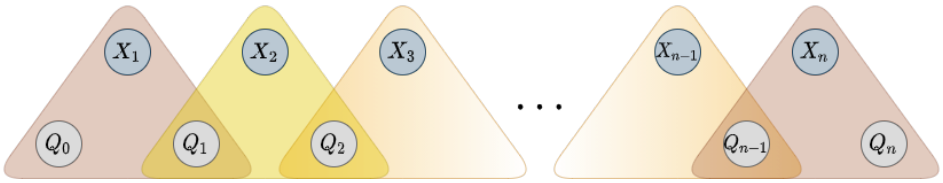


Figure 3.4: Hypergraph representation of the decomposition of a WRegular cost function with additional state variables Q_i and transition-encoding ternary functions.

path from u' to t. Therefore, we have:

$$\min_{\mathbf{t}|\ \mathbf{t}[X_i]=v} MDD(\mathbf{t}) = \min_{u \in L_i \xrightarrow{(v,c)} u'} \alpha[u] + c + \beta[u]$$

Comparison with existing regular encoding functions The ideas presented before have been implemented into a new global cost function MDD in ToulBar2, and compared to WRegular, a decomposition based, weighted automaton encoding global cost function [Allouche *et al.* 2016]; and SRegular and SRegularDP, dynamic-programming based, regular language encoding global cost functions [Lee & Leung 2012].

Preliminary tests showed that although MDD performed better than SRegular and SRegularDP, it failed at improving computation times and numbers of explored nodes compared to WRegular. Therefore, the Forward/Backward strategy was set aside, in favor of the cost function decomposition.

3.3.4 Exploiting automaton function decomposition

As in the CSP case, the WRegular cost function, encoding automaton $\mathcal{A}$, can be decomposed into a sequence of ternary cost functions [Allouche *et al.* 2016] (see Chapter 1). As stated in Chapter 1, the decomposition is achieved by adding $n+1$ state variables $Q_0, \ldots, Q_n$, and n ternary cost functions $w^{\mathcal{A}}_{Q_i,X_{i+1},Q_{i+1}}$, such that $w^{\mathcal{A}}_{Q_i,X_{i+1},Q_{i+1}}(q_i, x_{i+1}, q_{i+1}) = c$ if and only if there exists a transition from q_i to q_{i+1} in $\mathcal{A}$ labeled with value x_{i+1} and cost c. Variable Q_0 is restricted to the starting states and variable Q_n to the accepting states. Additional variables and ternary functions are represented in Figure 3.4.

To preserve the strength of the decomposition regarding soft local consistencies, the order of variables used to build the automaton must be consistent with the order used for directional AC [Allouche *et al.* 2016]. This condition can always be satisfied for Dist , as the set of forbidden solutions does not

depend on the order of the variables (the distance measure is the sum of independent variable-wise dissimilarities).

In the case of the DIST cost function, each state variable has $(\delta+1)\cdot(n+1)$ values (the number of states in the automaton) and each ternary function cost table describes costs of $(\delta+1)^2 \cdot (n+1)^2 \cdot d$ tuples, where d is the domain size. To speed up the resolution through better soft local consistency enforcements, we exploit the properties of DIST and the dissimilarity matrix D.

3.3.5 Compressing the encoding

The encoding of a DIST cost function in a sequence of n ternary functions, described in cost tables of size $(\delta+1)^2 \cdot (n+1)^2 \cdot d$ can be reduced along several lines.

Reducing state variable domains $\mathbf{D}_{Q_i}$ First, for DIST, we know that states s_i^d can only be reached after i transitions, *i.e.*, the reachable states of variable Q_i are the states in the i-th column in the DIST automaton (see Figure 3.2). The domains of the variables Q_i can be reduced to the $\delta+1$ states s_i^d:

$$\mathbf{D}_{Q_i} = \{q_i^d |\ 0 \leqslant d \leqslant \delta\}$$

Furthermore, our semi-metrics are defined by a non-decreasing sum of non-negative elements of D. Therefore, any state q_i^d can reach the accepting state q_n^δ if and only if the maximum dissimilarity md_i that can be achieved from variable i to variable n is larger than the remaining diversity to reach $\delta - d$. All such maximum dissimilarities md_i can be pre-computed in one pass over all variables in $\mathbf{X}$ as follows:

- $md_n = 0$
- For $0 \leqslant i < n$, $md_i = md_{i+1} + max_{v,v' \in \mathbf{D}_{i+1}} D(v, v')$

In the Hamming case, the distance can increase by 1 at most, *i.e.*, $\max_{v,v' \in \mathbf{D}_{i+1}} H(v, v') = 1$, therefore $md_i = n - i$.

A symmetric argument holds for the starting state q_0^0. An example of reduced state variable domains is given in figure 3.5. These simplifications reduce ternary cost tables to $O((\delta+1)^2 \cdot d)$.

Dissimilarity levels For a given dissimilarity matrix D, let $\#D$ denote the *number of distinct values* that appear in D.

If variables have domains of maximum size d and ignoring the useless 0 matrix, we know that $2 \leqslant \#D \leqslant 1 + \frac{d\cdot(d+1)}{2}$. However, distance matrices are

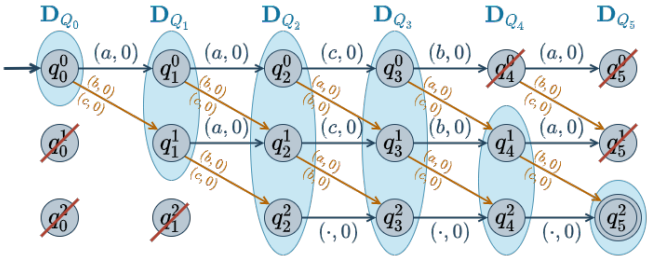
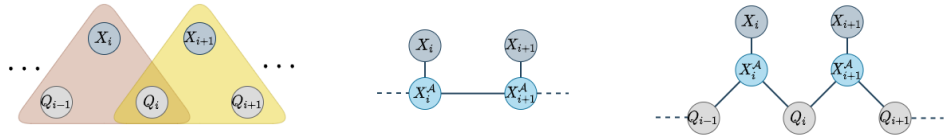


Figure 3.5: Reduced state variable domains $\mathbf{D}_{Q_i}$ in the decomposition of the DIST automaton of Figure 3.2. Initial domains consisted of all automaton states.



(a) (b) (c)

Figure 3.6: Representation of the ternary cost function (a), its dual representation (b), and its hidden representation (c).

usually more structured. For example, the BLOSUM62 similarity matrix leads to $\#D = 12$ levels.

In the Hamming case, there are $\#H = 2$ dissimilarity levels. This means that a state q_i^d can only reach states q_{i+1}^d or q_{i+1}^{d+1}. This sparsity of the transition matrix can be exploited, provided it is made visible. This can be achieved using extended variants of the *dual* and *hidden* encoding of constraint networks [Bacchus & Van Beek 1998, Larrosa & Dechter 2000]. These transformations, detailed hereafter, are known to preserve the set of solutions and their costs.

Dual representation In constraint satisfaction, the *dual* representation of a constraint network $\mathcal{X} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ is a new network $\mathcal{X}' = (\mathbf{X}', \mathbf{D}', \mathbf{C}')$ with:

- One variable $X_{\mathbf{S}}$ per constraint $c_{\mathbf{S}} \in \mathbf{C}$:

$$\mathbf{X}' = \{\, X_{\mathbf{S}} |\, c_{\mathbf{S}} \in \mathbf{C} \,\}$$

- Domain $\mathbf{D}_{X_{\mathbf{S}}}$ of variable $X_{\mathbf{S}}$ is the set of tuples $\mathbf{t} \in \mathbf{D}_{\mathbf{S}}$ that satisfy

the constraint $c_{\mathbf{S}}$:

$$\mathbf{D}' = \{\mathbf{D}_{X_{\mathbf{S}}} |\ c_{\mathbf{S}} \in \mathbf{C}\} \qquad \mathbf{D}_{X_{\mathbf{S}}} = \{\mathbf{t} \in c_{\mathbf{S}}\}$$

- For each pair of constraints $c_{\mathbf{S}}, c_{\mathbf{S}'} \in \mathbf{C}$ with overlapping scopes $\mathbf{S} \cap \mathbf{S}' \neq \varnothing$, there is a constraint $c_{\mathbf{X_S},\mathbf{X_{S'}}}$ that ensures that tuples assigned to $X_{\mathbf{S}}$ and $X_{\mathbf{S}'}$ are compatible, *i.e.*, they have the same values on the overlapping variables:

$$\mathbf{C}' = \{c_{\mathbf{X_S},\mathbf{X_{S'}}} |\ X_{\mathbf{S}}, X_{\mathbf{S}'}\ \in \mathbf{X}', \mathbf{S} \cap \mathbf{S}' \neq \varnothing\}$$

where

$$c_{\mathbf{X_S},\mathbf{X_{S'}}} = \{\ (\mathbf{t}, \mathbf{t}') \in \mathbf{D}_{X_{\mathbf{S}}} \times \mathbf{D}_{X_{\mathbf{S}'}} |\ \mathbf{t}[\mathbf{S} \cap \mathbf{S}']\ = \mathbf{t}'[\mathbf{S} \cap \mathbf{S}']\ \}$$

Instead of applying this transformation on all the functions of a CFN, we apply it to the reduced $w^{\mathcal{A}}_{Q_i,X_{i+1},Q_{i+1}}$ functions only (see Figure 3.6b).

The dual variable of $w^{\mathcal{A}}_{Q_i,X_{i+1},Q_{i+1}}$ is a variable $X^{\mathcal{A}}_i$ that contains all pairs $(q, q') \in Q_i \times Q_{i+1}$ such that there is a transition from q to q' in $\mathcal{A}$. For the Hamming case, the variable $X^{\mathcal{A}}_i$ has at most $2\delta + 1$ values. It is connected to X_i by a pairwise function:

$$\begin{array}{rccl} c_{X^{\mathcal{A}}_i,X_i}: & \mathbf{D}_{X^{\mathcal{A}}_i \times \mathbf{D}_i} & \longrightarrow & \{0, \ldots, \top\} \\ & ((q, q'), v) & \longmapsto & \Delta(q, v, q') \end{array}$$

where Δ is the weighted transition function of the automaton $\mathcal{A}$.

In this new dual representation, for every pair of consecutive dual variables $X^{\mathcal{A}}_{i-1}$ and $X^{\mathcal{A}}_i$, we add a function on these two variables to ensure that the arriving state of $X^{\mathcal{A}}_{i-1}$ is the starting state of $X^{\mathcal{A}}_i$:

$$c_{X^{\mathcal{A}}_{i-1}X^{\mathcal{A}}_i} : ((q_{i-1}, q'_{i-1}), (q_i, q'_i)) \mapsto \begin{cases} 0 & \text{if } q'_{i-1} = q_i \\ \top & \text{otherwise.} \end{cases}$$

In the worst case, this function has size $O(\#D^2 \cdot \delta^2)$ ($O(\delta^2)$ in the Hamming case). Only n extra variables are required.

Hidden representation The *hidden* representation of a constraint network $\mathcal{X} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ is a network $\mathcal{X}'' = (\mathbf{X}'', \mathbf{D}'', \mathbf{C}'')$ with:

- All the variables in $\mathbf{X}$ and the variables $X_{\mathbf{S}}$ from the dual network (and associated domains):

$$\mathbf{X}'' = \mathbf{X} \cup \mathbf{X}'$$

- For any dual variable $X_{\mathbf{S}}$, and each $X_i \in \mathbf{S}$, the set of constraints $\mathbf{C}''$ contains a function involving X_i and $X_{\mathbf{S}}$:

$$c_{X_i X_{\mathbf{S}}} : (v, \mathbf{t}) \in \mathbf{D}_i \times \mathbf{D}_{X_{\mathbf{S}}} \mapsto \begin{cases} 0 & \text{if } \mathbf{t}[X_i] = v \\ \top & \text{otherwise.} \end{cases}$$

As before, this transformation is applied to the reduced $w^{\mathcal{A}}_{Q_i, X_{i+1}, Q_{i+1}}$ functions only (see Figure 3.6c).

In this new hidden representation, we keep variables Q_i and create two pairwise functions involving each Q_i and respectively $X_i^{\mathcal{A}}$ and $X_{i+1}^{\mathcal{A}}$:

$$c_{Q_i X_{i-1}^{\mathcal{A}}} : (q'', (q, q')) \mapsto \begin{cases} 0 & \text{if } q'' = q \\ \top & \text{otherwise.} \end{cases}$$

$$c_{Q_i X_i^{\mathcal{A}}} : (q'', (q, q')) \mapsto \begin{cases} 0 & \text{if } q'' = q' \\ \top & \text{otherwise.} \end{cases}$$

These functions ensure that the state value of Q_i is consistent with the arriving state of the transition represented in $X_{i-1}^{\mathcal{A}}$ and the starting state of $X_i^{\mathcal{A}}$.

In the worst case, these functions have size $O(\#D \cdot \delta^2)$ ($O(\delta^2)$ in the Hamming case).

Comparison of the representations The dedicated dual and hidden representations require the description of $O(\delta \cdot d + \#D^2 \cdot \delta^2)$ and $O(\delta \cdot d + \#D \cdot \delta^2)$ tuples respectively (it is $O(\delta \cdot d + \delta^2)$ in the Hamming case), instead of the $O(d \cdot \delta^2)$ tuples in $w^{\mathcal{A}}_{Q_i, X_{i+1}, Q_{i+1}}$.

3.4 Combining the dissimilarity constraints

Given an assignment $\mathbf{t}$, a dissimilarity matrix D and a threshold δ, the global cost function $\textsc{Dist}(\mathbf{X}, \mathbf{t}, D, M)$ encodes the diversity requirement from a single assignment $\mathbf{t}$.

Definition 52. Given a set of solutions $\mathbf{Z}$, the global cost function $\textsc{Div}_{\min}$ is defined as follows:

$$\textsc{Div}_{\min}(\mathbf{X}, \mathbf{Z}, D, \delta) = \sum_{\mathbf{t} \in \mathbf{Z}} \textsc{Dist}(\mathbf{X}, \mathbf{t}, D, \delta)$$

It has cost 0 if and only if all individual dissimilarity constraints $\textsc{Dist}(\mathbf{X}, \mathbf{t}, D, \delta)$ are satisfied.

In the CSP framework, [Hebrard *et al.* 2005] proved that enforcing AC on $\text{Div}_{\min}$ is NP-hard. We know that soft arc consistency and stronger variants in the weighted framework generalize arc consistency in the CSP. Therefore, we know that enforcing soft arc consistency on $\text{Div}_{\min}$ is NP-hard too. $\varnothing$-inverse consistency is weaker than AC and we could hope that it can be enforced efficiently on $\text{Div}_{\min}$.

Theorem 6. $\varnothing$-inverse consistency is NP-hard to enforce on $\text{Div}_{\min}(\mathbf{X}, \mathbf{Z}, D, \delta)$.

Proof. Checking AC on $\text{Div}_{\min}$ is NP-complete. By definition of AC, $\text{Div}_{\min}$ is AC iff for any $X_i \in \mathbf{X}$, for any $v \in \mathbf{D}_i$, there exists an assignment of $\mathbf{X} \setminus \{X_i\}$ that has cost 0. Equivalently, if we reduce $\mathbf{D}_i$ to $\{v\}$, $\text{Div}_{\min}$ is $\varnothing$-inverse consistent.

AC has been reduced to a linear number of calls to a $\varnothing$-inverse consistency oracle. □

As before, $\text{Div}_{\min}$ defines a regular language and can be represented by a multivalued decision diagram (MDD). Enforcing AC on an MDD-encoding cost function is linear in the size of the MDD. As enforcing AC on $\text{Div}_{\min}$ is NP-hard, an MDD representing $\text{Div}_{\min}$ has exponential size.

3.4.0.1 $\text{Div}_{\min}(\mathbf{X}, \mathbf{Z}, D, \delta)$ encoding exponential-size MDD

A diversity encoding MDD is defined as before as a layered automaton, with a layer L_i per variable X_i, value/cost labeled edges link nodes in layer L_i to nodes in layer L_{i+1}. For a node u in layer L_i, the state of u, denoted I_u corresponds to the dissimilarities of the partial assignment $X_1 \dots X_i$ to each assignment of $\mathbf{Z}$:

$$I_u = \Big(\mathtt{d}(X_1 \dots X_i, \mathbf{t}[X_1 \dots X_i])\Big)_{\mathbf{t} \in \mathbf{Z}}$$

In each layer L_i, there are therefore $(\delta + 1)^M$ possible node states, with $M = |\mathbf{Z}|$, and as many potential nodes.

There are two possibilities, for the encoding of $\text{Div}_{\min}(\mathbf{X}, \mathbf{Z}, D, \delta)$ in the CFN $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$:

- Individual dissimilarity constraints $\text{Dist}(\mathbf{X}, \mathbf{t}, D, \delta)$, for each $\mathbf{t} \in \mathbf{Z}$, can be added to $\mathcal{C}$, with one of the regular encoding decomposition (ternary, hidden or dual).

  This strategy adds reasonable size cost functions to the network, with reasonable time consistency enforcements.

- The $\text{Div}_{\min}(\mathbf{X}, \mathbf{Z}, D, \delta)$ MDD can be built and added to the network with one of the decompositions (as MDD are equivalent to

automata). Enforcing local consistencies on this cost function will be time-consuming compared to the previous strategy. But, this cost function embedding all dissimilarity information, it should lead to enhanced lower bounds, and better pruning in the search tree.

3.4.0.2 Relaxing the $\text{Div}_{\min}$ MDD

As stated in [Bergman *et al.* 2011] in the CSP framework, generating exact exponential size MDDs is not practical. Instead, they propose to create a *relaxation* of the MDD to approximate the set of feasible solutions. We adapt these notions to the weighted framework.

Definition 53. Given an exponential size weighted MDD cost function $\mathcal{M}$, a ω-relaxation of $\mathcal{M}$ is a MDD $\mathcal{M}'$ such that $\mathcal{M}'(\mathbf{t}) \leqslant \mathcal{M}(\mathbf{t})$ for all assignments $\mathbf{t} \in \mathbf{D_X}$, and all layers of $\mathcal{M}'$ have less than ω nodes.

For a constraint-encoding MDD cost function, with cost 0 or $\top$, it means that the relaxation $\mathcal{M}'$ allows more solutions than $\mathcal{M}$.

Our diversity encoding strategy is as follows:

- A relaxed MDD for $\text{Div}_{\min}(\mathbf{X}, \mathbf{Z}, D, \delta)$ is computed, and added to the network $\mathcal{C}$. It aims at combining information from all diversity requirements to enhance propagations and pruning.
- To keep the diversity guarantee, all individual constraints $\text{Dist}(\mathbf{X}, \mathbf{t}, D, \delta)$ are also added to the CFN $\mathcal{C}$.

Given the maximum layer size ω, the relaxed MDD can be computed in a top-down manner [Bergman *et al.* 2011]:

- The first layer L_0 is computed: it has a single node s, with associtated state $I_s = (0, \cdots, 0)$
- For each $1 \leqslant i \leqslant n$, layer L_i is computed from layer L_{i-1} (see Section 3.3). If the number of nodes in L_i is greater than ω, the layer is relaxed, as described below.

Merging equivalent nodes Two nodes are *equivalent* is they lead to the terminal node with the same remaining partial assignments and costs. Therefore, equivalent nodes can be merged without loss of information.

For a given layer L_i, if two assignments $\mathbf{t}, \mathbf{t}' \in \mathbf{Z}$ are the same from position i, *i.e.*, $\mathbf{t}[X_{i+1} \dots X_n] = \mathbf{t}'[X_{i+1} \dots X_n]$, they can be treated as a single assignment for the remaining of the MDD compilation. Node states $I_u = \Big(\mathtt{d}(X_1 \dots X_i, \mathbf{t}[X_1 \dots X_i])\Big)_{\mathbf{t} \in \mathbf{Z}}$ can be reduced accordingly, decreasing the layer size.

Layer relaxation Once the layer L_i is reduced, if it has more than ω nodes, it needs to be relaxed: more nodes are to be merged, with possible loss of information.

Given a set of nodes $U \subset L_i$, defining the state of the resulting node by $\left(\max_{u \in U} I_u[\mathbf{t}]\right)_{\mathbf{t} \in \mathbf{Z}}$ ensures that no diversity satisfying assignment is lost.

Several strategies can be used for node selection:

Random $|L_i| - \omega + 1$ nodes are randomly selected and merged.

Small diversity Nodes with smallest diversities from assignments of $\mathbf{Z}$ can be selected, as they are more unlikely to reach satisfiability for $\text{DIV}_{\min}$ and be part of a solution.

High cost During the MDD computation, all edges are associated with cost 0: the estimated cost of a node u is the sum of unary costs of the values leading to it. Nodes with highest estimated costs are selected, as they are less likely to be on an optimum path.

In practice The relaxed $\text{DIV}_{\min}$ encoding MDD top-down compilation has been implemented. The aim was to find a trade-off between computation cost and information gain: the MDD has to be sufficiently large to improve the lower bound and reduce the number of explored nodes in the search tree; the MDD compilation and propagation has to be fast enough so that the extra information does speed up the search.

Unfortunately, for the relaxed MDD to be computed in a reasonable time (compared to the actual problem-solving time, with individual diversity constraints), the maximum layer width ω was such that the relaxation was too strong: the relaxed $\text{DIV}_{\min}$ constraint allowed all assignments and was useless.

3.5 Greedy DiverseSeq

The task at hand is the resolution of DiverseSet$(\mathcal{C}, D, M, \delta)$, *i.e.*, the generation of a set of M solutions with minimum cost, that satisfy a minimum pairwise diversity constraint. The exact computation being too expensive, we are tackling a greedy computation of DiverseSeq$(\mathcal{C}, D, M, \delta)$, a set of diverse good solutions that approximates DiverseSet . The DiverseSeq computation is iterative:

1. The CFN $\mathcal{C}$ is solved using branch-and-bound while maintaining soft local consistencies (see Chapter 1).
2. If a solution $\mathbf{t}$ is found, it is added to the ongoing solution sequence $\mathbf{Z}$.

3. If M solutions have been produced, the algorithm stops.
4. Otherwise, the cost function $\textsc{Dist}(\mathbf{X}, \mathbf{t}, D, \delta)$ is added to the previously solved problem.
5. We loop and solve the problem again (Step 1)

At Step 2, if no solution exists, the sequence of solutions $\mathbf{Z}$ can provably not be extended to length M and the problem has no solution (but a shorter sequence has been produced).

This basic schema has been improved in three different ways:

Incrementality Since the problems solved are increasingly constrained, all the equivalence preserving transformations and pruning that have been applied to enforce local consistencies at iteration $i-1$ are still valid in the following iterations. Instead of restarting from a problem $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C} \cup \bigcup_{1 \leq j < i} \{\textsc{Dist}(\mathbf{X}, \mathbf{Z}[j], D, \delta\})$, we reuse the problem solved at iteration $i-1$ after it has been made locally consistent, add the $\textsc{Dist}(\mathbf{X}, \mathbf{Z}[i-1], D, \delta)$ constraint and reinforce local consistencies. Similarly to incremental SAT solvers, adaptive variable ordering heuristics that have been trained at iteration $i-1$ are reused at iteration i.

Lower bound Since the problems solved are increasingly constrained, we know that the optimal cost oc^i obtained at iteration i cannot have a lower cost than the optimum cost oc^{i-1} reported at iteration $i-1$. When large plateaus are present in the energy landscape, this allows stopping the search as soon as a solution of cost oc^{i-1} is reached, avoiding a useless repeated proof of optimality.

Upper bound prediction Even if there are no plateaus in the energy landscape, there may be large regions with similar variations in energy. In this case, the difference in energy between oc^{i-1} and oc^i will remain similar for several iterations. Let $\Delta_i^h = \max_{\max(2,i-h) \leq j < i}(oc^j - oc^{j-1})$ be the maximum variation observed in the last h iterations (we used $h = 5$). At iteration i, we can first solve the problem with a temporary upper bound $k' = \min(k, oc_{i-1} + 2.\Delta_i^h)$ that should preserve a solution. If $k' < k$, this will lead to increased determinism, additional pruning, and possibly exponential savings. Otherwise, if no solution is found, the problem is solved again with the original upper bound k. We call this *predictive bounding*.

Each of these three improvements has the capacity to offer exponential time savings, and all are used in the experiments presented in the next sections.

3.6 First experiments on Bayesian networks

We implemented the iterative approach described above in its direct (ternary) decomposition as well as the dedicated hidden and dual versions for Hamming $\text{DIV}_{\min}$ decomposed in a conjunction of DIST functions above the CFN open-source solver Toulbar2 (see Chapter 1). All preprocessing algorithms dedicated to exact optimization that do not preserve suboptimal solutions had to be deactivated: variable elimination, dead-end elimination, and variable merging were deactivated at the root node.

We chose to enforce the strong Virtual arc consistency on the first problem solved. Its computational cost $O(\frac{e \cdot d^2 \cdot \top}{\varepsilon})$, where ε is the smallest representable cost in the fixed decimal point representation used, is high but is amortized over the M resolutions. During tree-search, the default EDAC local consistency was used.

All the experiments were performed on one core of a Xeon E5-2680 CPU at 2.50GHz with $\varepsilon = 10^{-6}$.

To show that our algorithm is not restricted to CPD problems and to compare it with the related δ-mode algorithm of [Chen *et al.* 2018], we first experimented on discrete Bayesian networks from `http://www.bnlearn.com/bnrepository` in the medium, large, very large, and massive categories. We used all the complete networks, with no change. Solving the Maximum Probability Explanation (MPE) problem on a Bayesian net (BN) reduces easily to the WCSP: the probability in a BN is the product of conditional probabilities: minimizing the sum of the $-\log$ of probabilities is equivalent to maximizing the product.

Table 3.1 reports the number of variables n, the maximum domain size d, and the number of 4-diverse solutions found within a maximum of 300 CPU seconds per instance for each representation. Even though this is not our main target, in the dual (fastest) case, we also computed the number of 4-diverse solutions that are 3-modes for comparison with [Chen *et al.* 2018]. As shown in Theorem 4, checking if an assignment $\mathbf{t}$ is a δ-mode of CFN $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ can be checked in polytime by solving $(\mathbf{X}, \mathbf{D}, \mathbf{C} \cup \{\text{DIST}(\mathbf{X}, \mathbf{t}, D, -\delta)\})$. We report the total CPU-time taken to find four 3-modes. These times include both the time to find all 4-diverse solutions and to filter out those that are not 3-modes (the filtering time was always small compared to the time needed to produce the 4-diverse solutions: checking all 4-diverse solutions of all problems took less than 6 minutes overall). For the two problems (*child* and *alarm*) that we share with the benchmarks used in [Chen *et al.* 2018], we see very similar CPU-times. We could not compare our approach on the 3 remaining problems used by [Chen *et al.* 2018]. They were learned from data and the authors could not provide us with the estimated parameters

			dual			hidden	3ary
name	n	d	4-d	3-m	CPU	4-d	4-d
alarm	37	4	147	28	0.38	128	141
andes	223	2	102	8	4.96	87	110
barley	48	67	126	92	2.74	125	113
child	20	6	172	8	1.65	153	154
diabetes	413	21	0	0	TO	0	0
hailfinder	56	11	109	24	0.65	92	102
hepar2	70	4	96	1	TO	87	93
insurance	27	5	186	30	0.40	164	153
link	724	4	117	117	6.69	113	93
mildew	35	100	107	25	7.71	100	97
munin	1041	21	16	2	TO	16	14
pathfinder	109	63	118	15	2.46	90	124
pigs	441	3	162	162	6.28	134	138
water	32	4	183	17	0.48	161	174
win95	76	2	132	21	0.93	115	119

Table 3.1: For each network, we give the network name, number of variables, and maximum domain size. Then for the dual encoding, we report the number of 4-diverse solutions found in 5 minutes, the number of 3-modes among these, and the CPU-time taken to produce four 3-modes, as in [Chen *et al.* 2018]. We then give the number of 4-diverse solutions produced in the same amount of time using the hidden and ternary encoding.

(these problems are small: the largest has 45 variables and Boolean domains ; it however took 380s to the algorithm of [Chen *et al.* 2018] to provide four 3-modes on it). The energy landscape of each problem has a strong influence on the fraction of $(\delta+1)$-diverse solutions that are δ-modes: on the *link* and *pigs* problems, which present a large plateau at optimum, our approach is very fast. Instead, the *hepar2* instance shows a very large nonflat basin around the optimum, which offers plenty of 4-diverse solutions but just one local minimum.

We also compared our greedy DiverseSeq-solving approach to a globally optimal approach which solves the harder DiverseSet problem. As described in Section 3.1, we made $M = 4$ copies of the *child* medium-size Bayesian network. We decomposed each $\textsc{Dist}(\mathbf{X}^i, \mathbf{X}^j, H, \delta)$ constraint on copy variable sets $\mathbf{X}^i$ and $\mathbf{X}^j$ in a sequence of ternary cost functions using extra Boolean variable set $\mathbf{E}^{i,j}$, with $\forall 1 \leq l \leq n, \mathbf{E}^{i,j}[l] = \mathbb{1}(\mathbf{X}^i[l] \neq \mathbf{X}^j[l])$ supplementary disequality constraints. The resulting DiverseSet child problem has 320 variables. We solved it with toulbar2 version 1.0, varying

δ from 1 to 15. It took more than 23 hours (resp. 3 hours) to solve the $\delta = 15$ (resp. $\delta = 14$) DiverseSet problem. Instead, solving DiverseSeq took 0.28 second for $\delta = 15$ and produced a sequence of fifteen solutions in 1.1 seconds. This efficiency comes with little side-effects in terms of cost: Figure 3.7 shows the total cost $\sum_{\mathbf{t}\in\mathbf{Z}} J_N(\mathbf{t})$ for the two approaches. A solution to the DiverseSeq problem can still be a solution of the DiverseSet problem for large Hamming distances (up to $\delta = 11$) and remains less than 12.9% above the optimum for $\delta = 15$.

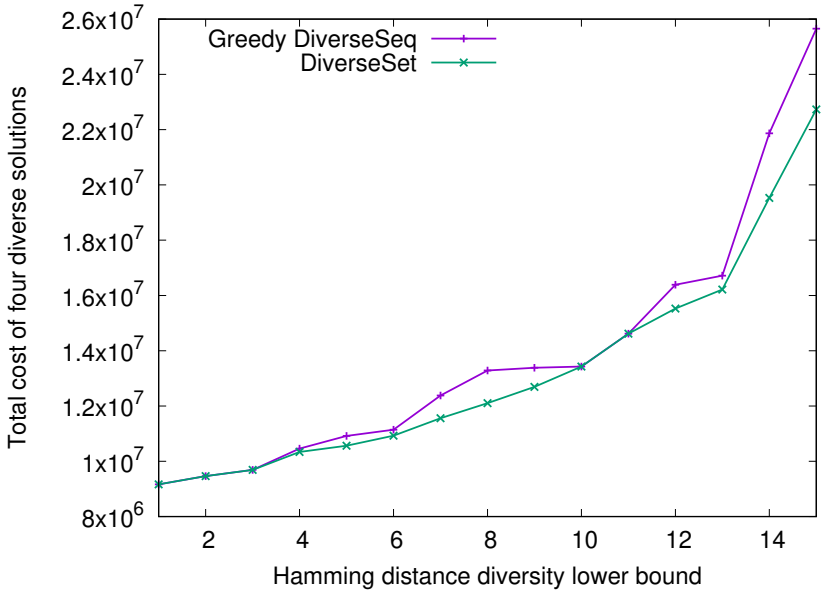


Figure 3.7: Comparison between greedy DiverseSeq and DiverseSet on the *child* Bayesian network.

3.7 Experiments on CPD instances

Following our main motivation for protein design, we extracted two sets of prepared protein backbones for full redesigns from the benchmark sets built by [Simoncini *et al.* 2015] and [Ollikainen & Kortemme 2013]. In [Simoncini *et al.* 2015], we selected the 20 proteins that had required the least CPU-time to solve, as indicated in the Excel sheet provided in the supplementary information of the paper. In [Ollikainen & Kortemme 2013], the instances were harder, and we selected the 17 instances that required less than 24 hours of CPU-time for the GMEC to be computed by Toulbar2. These instances are listed in Table 3.2.

Full redesign was performed on each structure, and CFN instances were

PDB ID	n	d	PDB ID	n	d	PDB ID	n	d
1aho	56	378	3i8z	50	354	1ten	81	392
2fjz	53	324	2cg7	82	380	1ucs	60	342
1b9w	78	386	3rdy	65	396	2bwf	69	347
2gkt	45	357	2erw	47	446	2evb	68	323
1f94	53	386	3vdj	67	391	2O37	60	386
2pne	77	401	2fht	64	346	2O9S	48	327
1hyp	66	385	1bxy	52	384	3F04	87	356
2pst	61	357	1ctf	68	349	3fym	70	348
1uln	66	367	1czp	76	373	3gqs	67	344
1uoy	56	337	1fqt	85	377	3gva	87	348
2ca7	44	348	1guu	47	350	3i2z	67	360
1yzm	46	294	1t8k	68	361			

Table 3.2: List of protein structures used in our benchmark set, for full redesign: pdb identifier, domain length n (number of variables in the resulting CFN) and maximum domain size d.

generated using the Dunbrack library [Shapovalov & Dunbrack Jr 2011] and Rosetta ref2015 score function [Park *et al.* 2016]. Resulting networks have from 44 to 87 variables, and maximum domain sizes range from 294 to 446 rotamers.

Predictive bounding contribution $M = 10$ solutions with diversity threshold $\delta = 10$ for each problem from [Simoncini *et al.* 2015] were generated, with and without predictive bounding. The worst CPU-time spent on the resolution without predictive bounding was 32 minutes. It was reduced to 17 minutes with predictive bounding. The average computation time was 201s per problem. This shows that predictive bounding provides a simple and efficient boost and that real CPD instances can be solved in a reasonable time, even when relatively large diversity requirements are used.

Diversity improves prediction quality For all instances, sets of $M = 10$ solutions were generated with diversity threshold δ ranging from 1 to 15. For $\delta = 1$, the set of solutions produced is just the set of the 10 best (minimum energy) sequences.

These CPD problems use real protein backbones, determined experimentally. A native sequence exists for these backbones, therefore it is possible to measure the improvements diversity brings with native sequence recovery (NSR) and native sequence similarity recovery (NSSR). NSR and NSSR measure how much the redesigned protein resembles the natural in terms

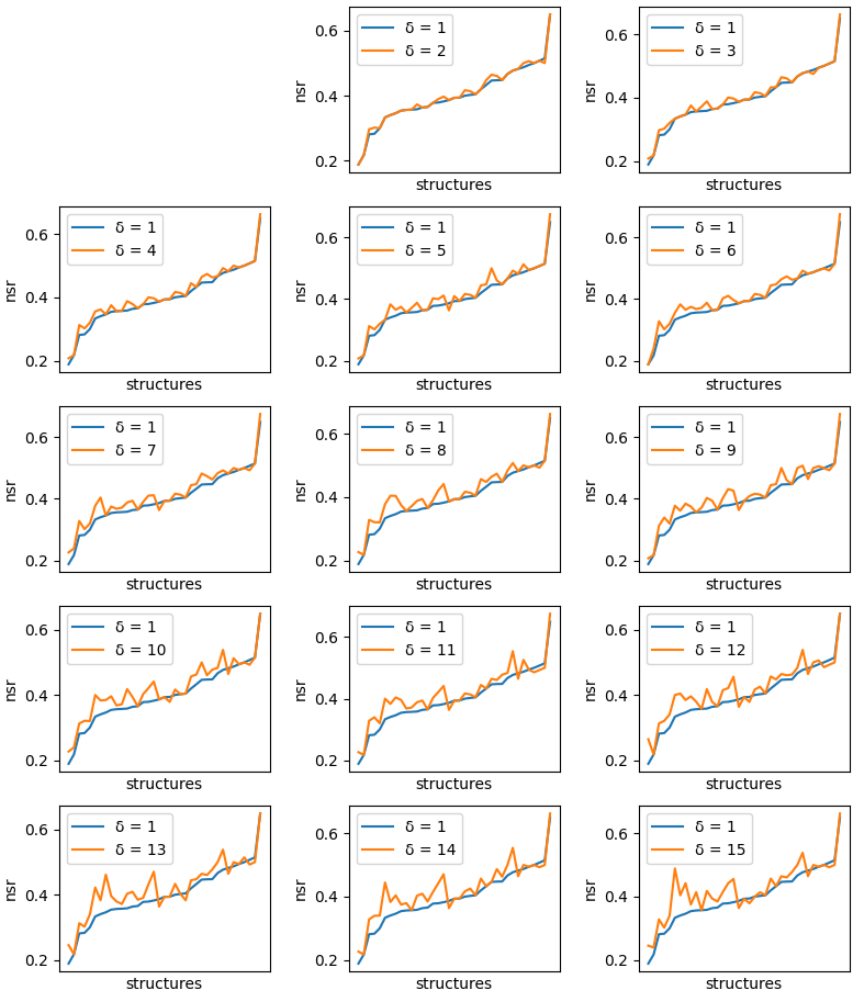


Figure 3.8: Comparison of the best NSR value among the 10 best sequences ($\delta = 1$) with the best NSR value of sequences predicted with diversity $\delta = 2 \ldots 15$, for all structures. Sequences are ranked by increasing best NSR among enumerated sequences.

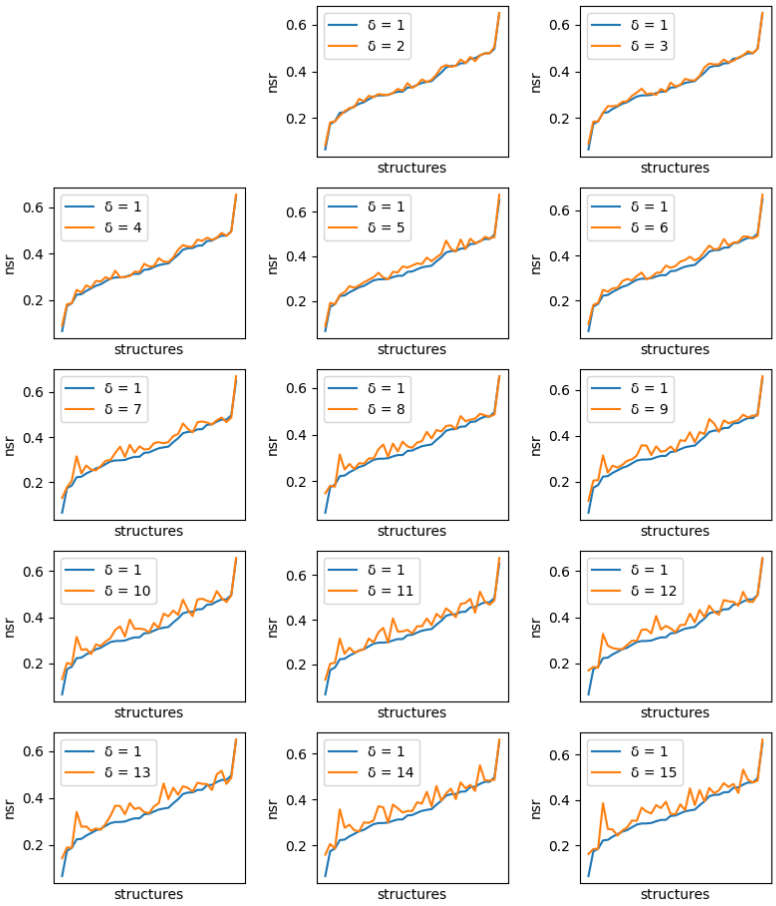


Figure 3.9: Comparison of the best NSSR value among the 10 best sequences ($\delta = 1$) with the best NSSR value of sequences predicted with diversity $\delta = 2 \ldots 15$, for all structures. Sequences are ranked by increasing best NSSR among enumerated sequences.

	Exact resolution		**Subopt. resolution**	
δ	**NSR**	**NSSR**	**NSR**	**NSSR**
2	2.88e-03	7.83e-04	4.11e-01	1.75e-01
3	3.87e-04	2.29e-05	1.14e-01	5.50e-02
4	4.42e-05	8.47e-07	4.48e-03	1.41e-02
5	8.11e-05	7.35e-07	1.98e-03	6.85e-03
6	1.51e-05	2.64e-07	7.47e-05	3.52e-05
7	1.88e-05	8.94e-07	8.86e-06	2.51e-06
8	1.27e-05	7.45e-07	8.19e-06	5.27e-06
9	2.76e-05	1.39e-07	2.07e-05	1.88e-06
10	1.14e-05	1.13e-06	1.78e-05	1.38e-05
11	4.27e-05	3.60e-06	2.32e-05	1.11e-05
12	6.63e-05	3.60e-06	1.75e-05	1.46e-05
13	4.43e-05	2.64e-06	2.48e-06	1.03e-05
14	2.29e-05	4.88e-06	5.26e-06	3.73e-07
15	3.92e-05	2.21e-06	2.68e-05	1.48e-05

Table 3.3: p-values for a unilateral Wilcoxon signed rank test comparing the sample of best NSR (resp. NSSR) for each $\delta = 2 \dots 15$ with $\delta = 1$, for optimal and suboptimal (3 kcal/mol allowed energy gap to real optimum) resolution.

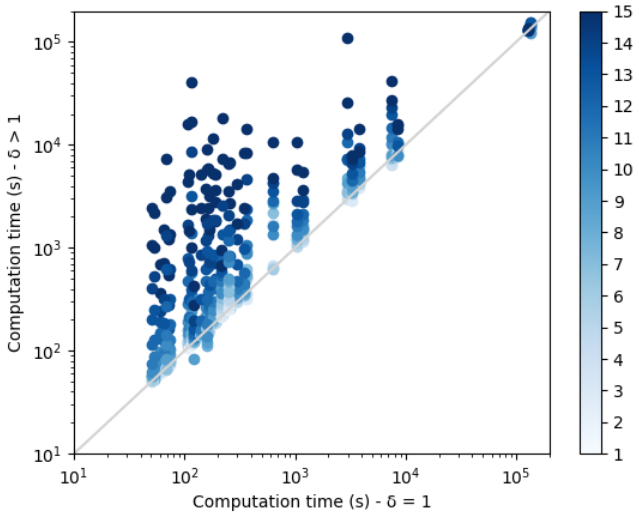


Figure 3.10: Comparison of the computation times of sequence sets without diversity $\delta = 1$, with sequence sets with diversity $\delta > 1$.

of sequence identity and similarity (see Chapter 2). If solution diversity helps, the maximum NSR/NSSR over the 10 sequences should improve when δ is large compared to when $\delta = 1$, as long as the costs remain close to the optimum. A solution cost too far from the optimum, which could be generated because of a diversity threshold set too high, would mean a poor quality of the solution. Even with $\delta = 15$, the maximum difference in energy we observed with the global minimum energy never exceeded 4.3 kcal/mol (with an average of 2.1 kcal/mol).

For each protein, and each diversity threshold $\delta = 2 \dots 15$, we compared the best NSR (resp. NSSR) that could be obtained with δ to the best obtained with diversity threshold 1, *i.e.*, the simple enumeration of the 10 best sequences. Results are plotted in Figure 3.8 (resp. Figure 3.9). They show a general increase in the best NSR (resp. NSSR) when diverse sequences are produced, even with small δ. To validate the diversity improvement of sequence quality, p-values for a unilateral Wilcoxon signed-rank test comparing the sample of best NSR (resp. NSSR) for each $\delta = 2 \dots 15$ with $\delta = 1$ were computed. They are shown in Table 3.3.

Each dissimilarity constraint adds n extra variables to the network (with the dual representation). These variable domain sizes increase with the diversity threshold δ. Computation times are plotted in Figure 3.10. As expected, a threshold increase leads to an exponential increase in the computation time.

The small set of points on the top right corner of the plot correspond to the structure 1FYM. For high diversity thresholds, the 24h computation time limit was reached and less than 10 sequences were produced.

Suboptimal diverse sequences Given that the optimized function is an approximation of the real intractable energy, solving the problem to optimality might seem exaggerated. The requirement for optimality that we have used in previous experiments can be trivially relaxed to a relative or absolute approximation guarantee using artificially tightened pruning rules as originally proposed in [Pohl 1970]. This is already implemented in the toulbar2 solver (using the `-agap` flag).

For diversity threshold $\delta = 2 \dots 15$, we generated sets of 10 suboptimal diverse sequences that satisfy the diversity constraints, with 3 kcal/mol allowed energy gap to the optimum. The maximum energy degradation we observed with the global minimum energy never exceeded 5.65 kcal/mol (with an average energy difference of 3.86 kcal/mol). This is slightly more than the 4.3 kcal/mol worse degradation (average 2.1 kcal/mol) of the resolution, without any allowed energy gap.

We compared these samples of suboptimal sequences to the set of 10 exact best sequences. Results for NSR and NSSR are shown in Figures 3.11 and 3.12 respectively, and corresponding p-values are displayed in Table 3.3 (unilateral Wilcoxin signed rank test). With dissimilarity threshold $\delta \geqslant 6$, it is clear that the set of diverse suboptimal sequences have better quality than the 10 best enumerated sequences. Moreover, as shown in Figure 3.13, for harder instances, suboptimal diverse resolution is faster than exact enumeration.

So, when predicting a library of sequences, if the instance is hard, it might be wise to generate suboptimal diverse sequences, instead of enumerating minimum energy sequences, without diversity. Doing so, there is a higher chance of predicting better sequences "in practice" faster.

Conclusion

Producing a sequence of diverse solutions is a very usual requirement when an approximate or learned model is used for optimal decoding. In this Chapter, we show that using an incremental CFN approach using diversity constraints represented as weighted automata that are densely encoded in a dedicated dual encoding together with predictive bounding, it is possible to produce sequences of solutions that satisfy guarantees on diversity on large Bayesian networks (in the "massive" category of the BN repository) as well as for large Computational Protein Design instances. This guarantee is obtained

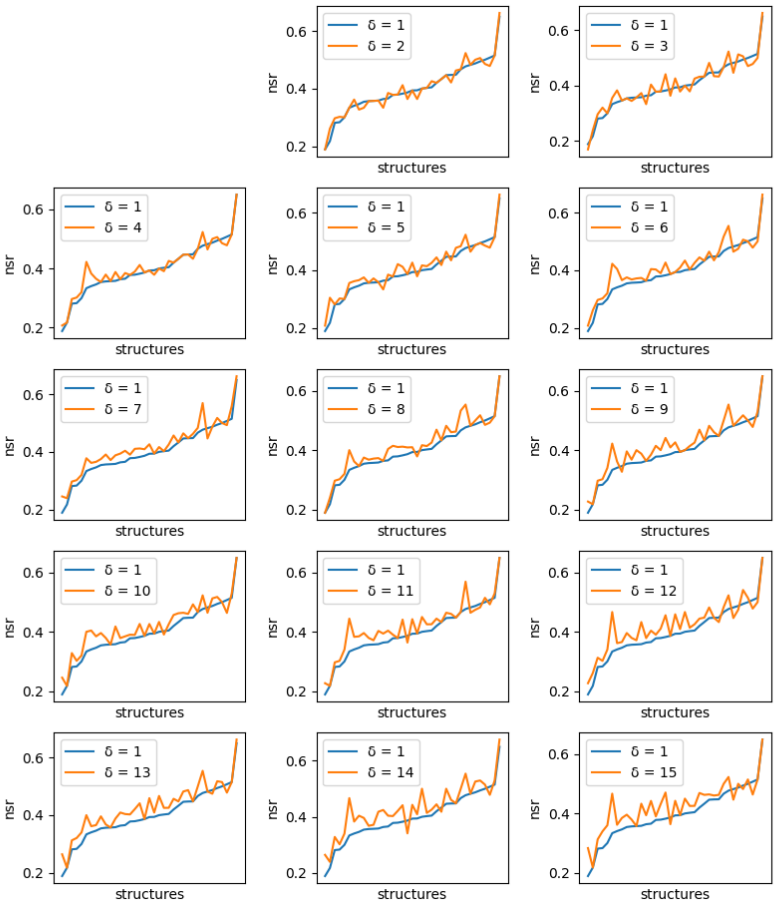


Figure 3.11: Comparison of the best NSR value among the 10 best sequences ($\delta = 1$) with the best NSR value of sequences predicted with diversity $\delta = 2 \ldots 15$ and allowed optimality gap of 3kcal/mol, for all structures. Sequences are ranked by increasing best NSR among enumerated sequences.

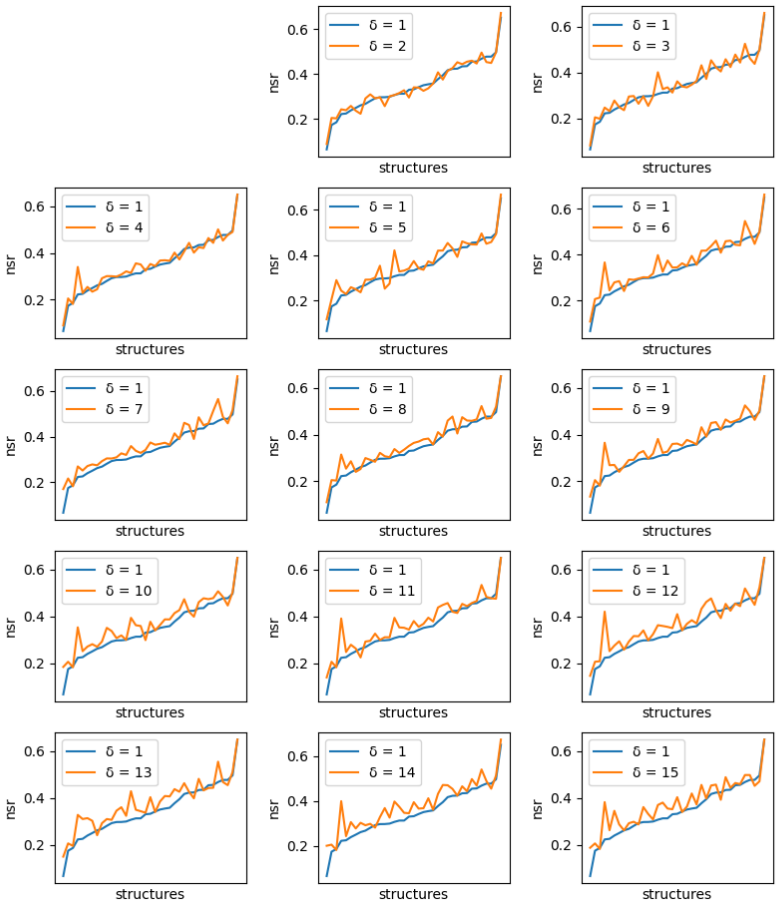


Figure 3.12: Comparison of the best NSSR value among the 10 best sequences ($\delta = 1$) with the best NSSR value of sequences predicted with diversity $\delta = 2 \ldots 15$ and allowed optimality gap of 3kcal/mol, for all structures. Sequences are ranked by increasing best NSSR among enumerated sequences.

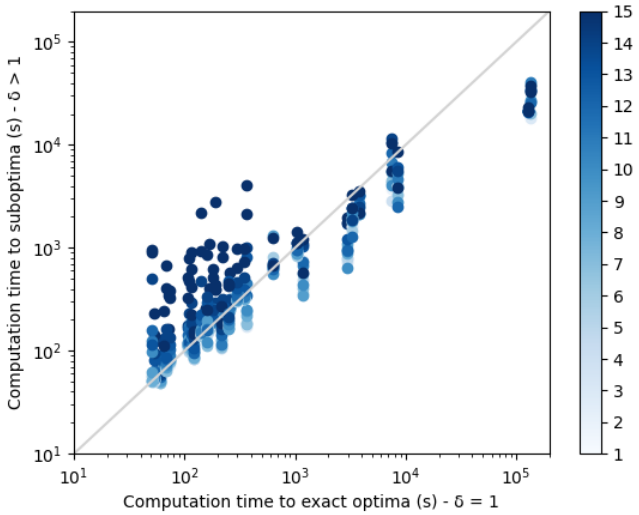


Figure 3.13: Comparison of the computation times of sequence sets without diversity $\delta = 1$, with suboptimal sequence sets with diversity $\delta > 1$. An energy gap of 3 kcal/mol is allowed to actual optimum.

on dense problems with non-submodular functions while also guaranteeing that each new solution produced is the best given the previously identified solutions.

We also showed that the stream of diverse solutions that our algorithm produces can be filtered and each solution efficiently identified as being a δ-mode or not. Compared to the results of [Chen *et al.* 2018], our approach looks computationally at least as efficient, while providing a much more extensive report on the shape of the energy landscape around the optimum.

On real protein design problems, we observe that small and large diversity requirements do improve the quality of sequence libraries when native proteins are fully redesigned. Moreover, large diversity requirements on suboptimal sequences also improve the quality of sequence libraries, compared to a simple enumeration of the minimum energy sequences. In the context of optimizing an approximate or learned function, the requirement for an optimal cost solution may be considered as exaggerated. Our guaranteed suboptimal resolution is useful, given that even computationally expensive approaches with asymptotic convergence results such as Simulated Annealing may fail with unbounded error [Simoncini *et al.* 2015].

Two directions could extend this work. The first direction is brought to light by the *diabetes* problem for which even the first element of DiverseSeq could not be produced in five minutes of CPU-time. This problem has tree-

width less than 4 and can be solved by Toulbar2 in less than 40 seconds if one exploits a min-fill tree decomposition during branch-and-bound search. It would be therefore desirable to show that the decomposed ternary or binary functions we use for encoding DIST can be arranged in such a way that tree-width can be preserved or not exaggeratedly increased. This would enable the efficient production of diverse solutions for otherwise unsolved structured instances. The second direction would be to identify a relaxed version of the NP-hard DIV_{min} constraint that would produce tighter bounds thanks to local consistencies. Despite non-negligible efforts in this direction using relaxed Multi-Valued Decision Diagrams, this has eluded us up to now.

Chapter 4

Positive and negative multi-state design

Contents

4.1 Multi-state protein design 112
4.1.1 Multi-state CPD as an optimization problem 112
4.1.2 Existing methods for multistate protein design 114
4.1.3 Complexity 115
4.2 Positive multi-state design as a WCSP 118
4.2.1 Positive MSD as a CFN 119
4.2.2 Experiments 120
4.2.3 CPU-time analysis 121
4.3 Negative multi-state design 123
4.3.1 Negative MSD as a bilevel WCSP 123
4.3.2 Preliminary tests 129

Introduction

Computational protein design answers the need for automatic methods for the prediction of new protein sequences, that will fold into desired structures, with enhanced properties and functionalities. In single state design, one protein conformation describes the input target structure, and a sequence of minimum energy on this backbone is sought. This problem being intractable, several simplifying assumptions are made. The protein backbone is assumed to be rigid, and amino acid side chain conformations are discretized into rotamers. These assumptions enable the modeling of the sequence energy as a cost function network. Efficient branch-and-bound and local consistency based algorithms are used, to find the global minimum energy conformation.

Unfortunately, as stated before, the rigid backbone, discrete rotamer simplifications ignore protein flexibility, even though proteins are known to not remain in a single position. They rather evolve in a set of low energy

conformations. To integrate flexibility in the design model, *multi-state design* requires an ensemble of backbone conformations as the description of the input structure. This favors the design of flexible proteins that move slightly around their "ideal" conformation, or perform large conformational changes.

Negative multi-state design allows for both *desired* and *undesired* structures as an input. The sought sequence is aimed at stabilizing the positive (desired) states while destabilizing the negative (undesired) conformations. This enables for example the modeling of affinity improvement: the desired state is the bound structure, and the undesired state the unbound structure. Target specificity can also be modeled: complexes with the protein bound to the desired and undesired ligands are the positive and negative states respectively.

Section 4.1 starts with the introduction of the positive and negative multistate computational design, as the minimization of the weighted average state energy (negative states are negatively weighted). It then provides a brief overview of existing MSD methods. The complexities of both positive and negative design problems are studied. Then, Section 4.2 shows how positive multistate design can be modeled with cost function networks, and that it improves design quality compared to single state design. Finally, Section 4.3 gives the negative design formulation as a bilevel weighted constraint satisfaction problem. It states the bases of the adaptation of the branch-and-bound search and the local consistencies to bilevel optimization.

Results on positive multistate design have been published in [Vucinic *et al.* 2020].

4.1 Multi-state protein design

4.1.1 Multi-state CPD as an optimization problem

The formulation of MSD is very similar to SSD. Multiple target structures are given as input, the sequence-conformation search space is defined, as well as an objective function to be optimized:

Input: the target structure The input target structure is given as either a set of positive conformations $\mathbf{B} = \mathbf{B}^+$, or a set of positive and negative conformations $\mathbf{B} = \mathbf{B}^+ \cup \mathbf{B}^-$, where the backbones in $\mathbf{B}^+$ are desired, and the ones in $\mathbf{B}^-$ are undesired. Without loss of generality, all backbones are assumed to have the same number of positions n.

Search space: sequences and conformations At each position $1 \leqslant i \leqslant n$, the set $\mathcal{S}_i$ denotes the allowed amino acid types in position i; and for each $a \in \mathcal{S}_i$ and backbone $\mathbf{B}^j \in \mathbf{B}$, the set $\mathcal{R}_i^j(a)$ corresponds to the possible

conformations, or rotamers, of amino acid a at position i in backbone $\mathbf{B}^j$. The set $\mathcal{R}_i^j$ denotes all the possible rotamers at position i in backbone $\mathbf{B}^j$.

Objective: fitness and objective functions Each backbone $\mathbf{B}^j$ is equiped with a pairwise decomposable energy function on its set of conformations:

$$\mathbf{E}^j : \mathbf{r} \in \prod_{i=1}^{n} \mathcal{R}_i^j \longmapsto E_\varnothing^j + \sum_{i=1}^{n} E_i^j(\mathbf{r}[i]) + \sum_{1 \leqslant i < i' \leqslant n} E_{ii'}^j(\mathbf{r}[i], \mathbf{r}[i'])$$

The *fitness* of an amino acid sequence on a set of backbones is defined as the sequence average minimum conformation energy obtained for all the backbones. Minimizing this fitness should lead to a sequence suitable for all states. As the number of states is fixed, minimizing the fitness corresponds to minimizing the sum of state minimum conformation energies.

The *positive* multi-state design, with positive input conformations $\mathbf{B} = \mathbf{B}^+$, consists in finding the optimum fitness sequence $\mathbf{s}^*$:

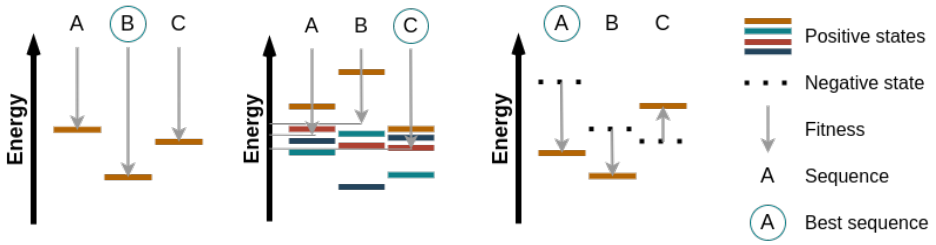
$$\mathbf{s}^* = \underset{\mathbf{s} \in \prod_i \mathcal{S}_i}{\arg\min} \left(\sum_{\mathbf{B}^j \in \mathbf{B}} \min_{\mathbf{r}^j(\mathbf{s}) \in \mathcal{R}^j(\mathbf{s})} \mathbf{E}^j(\mathbf{r}^j(\mathbf{s})) \right)$$

where $\mathcal{R}^j(\mathbf{s}) = \prod_i \mathcal{R}_i^j(\mathbf{s}[i])$ is the set of possible conformations of the sequence $\mathbf{s}$ on backbone $\mathbf{B}^j$.

When both positive and negative state sets are given as input, the objective function is defined as the *difference* between the fitness of the positive and negative states. The aim is to find a sequence that minimizes the minimum conformation energies on positive states and maximizes the minimum conformation energies on negative states. The resulting sequence should therefore adopt one or several of the positive state backbones, while being unlikely to adopt one of the negative state backbones, because all side chain positions on negative backbones should be unstable (with high energy). More precisely, *negative* multi-state design consists in finding the optimum sequence $\mathbf{s}^*$ defined by:

$$\mathbf{s}^* = \underset{\mathbf{s} \in \prod_i \mathcal{S}_i}{\arg\min} \left[\left(\sum_{\mathbf{B}^j \in \mathbf{B}^+} \min_{\mathbf{r}^j(\mathbf{s}) \in \mathcal{R}^j(\mathbf{s})} \mathbf{E}^j(\mathbf{r}^j(\mathbf{s})) \right) - \left(\sum_{\mathbf{B}^j \in \mathbf{B}^-} \min_{\mathbf{r}^j(\mathbf{s}) \in \mathcal{R}^j(\mathbf{s})} \mathbf{E}^j(\mathbf{r}^j(\mathbf{s})) \right) \right]$$

To sum up, the input of discrete rigid MSD consists of a set of positive backbone states, that represents the target structure, and a set of negative backbone states, that are to be avoided. Each amino acid sequence is mapped to an *objective* value, defined as the difference of the fitness of positive and negative states. The fitness of a set of (positive or negative) backbones



(a) SSD (b) Positive MSD (c) Negative MSD (d) Legend

Figure 4.1: Sequence evaluation in single state, positive multistate, and negative multistate design.

is defined as the sequence average minimum conformation energy over all backbones.

min-MSD Some design strategies use another fitness function, defined as the Boltzmann-weighted energy over the input states (defined as $\sum_j e^{-\beta E^j} \cdot E^j$ where $\beta = \frac{1}{k_B T}$). This gives an exponential advantage to the backbone with the lowest energy, and it suits well design instances where the input backbones describe "possible" conformations, without further knowledge on which will be adopted eventually. This can be approximated by the *minimum* energy: the fitness of a set of backbones is the minimum energy across all states. As detailed in [Vucinic *et al.* 2020], min-MSD can be modeled with a WCSP but design quality is worse than with the sum-based fitness function. This approach will not be described further here.

4.1.2 Existing methods for multistate protein design

4.1.2.1 Stochastic methods

Many SSD sampling methods have been extended to multi-state design [Davey & Chica 2012], with various objective functions, such as genetic algorithms [Havranek & Harbury 2003, Leaver-Fay *et al.* 2011a, Löffler *et al.* 2017], Monte-Carlo simulated annealing [Ambroggio & Kuhlman 2006], or the FASTER approach [Allen & Mayo 2010].

4.1.2.2 Exact methods

CLASSY and CLEVER Relying on cluster expansion and a learned energy function that only depends on amino acid types, [Grigoryan *et al.* 2009, Negron & Keating 2013, Negron & Keating 2014] use Integer Linear Programming to search the optimum multi-state sequence.

Dead-end elimination Single state dead-end elimination eliminates rotamers based on a dominance analysis while preserving the global optimum. It can not be applied as such on multistate design as states are further constrained by the fact that they must correspond to the same amino acid sequence. [Yanover *et al.* 2007] developed a type-dependent DEE that eliminates rotamers dominated by other rotamers of the same amino acid type. The search space is thereby reduced, but all sequence optima are preserved.

DEE/A* The OSPREY software [Gainza *et al.* 2013] embeds the *COMETS* algorithm [Hallen & Donald 2016, Hallen & Donald 2019] that uses type-dependent DEE and an adapted A* search for MSD. An A* tree is built that searches through sequences. For any promising sequence, a "tree within a tree" is searched for the optimum state conformations.

Interconnected CFN Recently, [Karimi & Shen 2018] presented iCFN, an exact MSD approach, based on cost function networks. A sequence search tree is first explored, using a DFBB strategy, with an adapted, non-incremental, node lower bound. Then, each state is searched for its optimum conformation, using the CFN approach based on branch-and-bound and local consistencies, as presented in Chapter 1.

4.1.3 Complexity

We know that the decision problem associated with single state design is NP-complete (see Chapter 2, [Pierce & Winfree 2002]). In this Subsection, the complexities of positive and negative multi-state design are studied.

Definition 54. PosDes is the decision problem associated with the positive MSD problem. It is defined as follows:

Instance
- A set of desired backbones $\mathbf{B} = \mathbf{B}^+$ of same length n
- For each position $1 \leqslant i \leqslant n$, a set of allowed amino acids $\mathcal{S}_i$
- For each position $1 \leqslant i \leqslant n$, and backbone $\mathbf{B}^j \in \mathbf{B}$, a set of allowed rotamers $\mathcal{R}_i^j = \bigcup_{s \in \mathcal{S}_i} \mathcal{R}_i^j(s)$
- For each backbone $\mathbf{B}^j$, a pairwise decomposable energy fonction on conformations $\mathbf{E}^j$
- An upper bound k

Question Does there exist a sequence $\mathbf{s} \in \prod_i \mathcal{S}_i$, and for each backbone $\mathbf{B}^j \in \mathbf{B}$ a state conformation $\mathbf{r}^j(\mathbf{s}) \in \prod_i \mathcal{R}_i^j(\mathbf{s}[i])$ such that:

$$\sum_{\mathbf{B}^j \in \mathbf{B}} \mathbf{E}^j(\mathbf{r}^j(\mathbf{s})) \leqslant k$$

Theorem 7. PosDes is NP-complete.

Proof. PosDes belongs to NP because it is possible to check a positive instance given a short certificate defined by a sequence $\mathbf{s} \in \prod_i \mathcal{S}_i$ and conformations $\mathbf{r}^j(\mathbf{s}) \in \prod_i \mathcal{R}_i^j(\mathbf{s}[i])$ for $\mathbf{B}^j \in \mathbf{B}$. It suffices to compute the joint fitness of all states $\sum_{\mathbf{B}^j} \mathbf{E}^j(\mathbf{r}^j(\mathbf{s}))$ and check wether it is lower than the threshold k.

PosDes is NP-hard since SSD is NP-hard and a subproblem of PosDes where $|\mathbf{B}^+| = 1$. □

Definition 55. NegDes is the decision problem associated with the negative MSD problem. It is defined as follows:

Instance
- A set of desired and undesired backbones $\mathbf{B} = \mathbf{B}^+ \cup \mathbf{B}^-$ of same length n
- For each position $1 \leqslant i \leqslant n$, a set of allowed amino acids $\mathcal{S}_i$
- For each position $1 \leqslant i \leqslant n$, and backbone $\mathbf{B}^j \in \mathbf{B}$, a set of allowed rotamers $\mathcal{R}_i^j = \bigcup_{s \in \mathcal{S}_i} \mathcal{R}_i^j(s)$
- For each backbone $\mathbf{B}^j \in \mathbf{B}$, a pairwise decomposable energy fonction on conformations $\mathbf{E}^j$
- An upper bound k

Question Does there exist a sequence $\mathbf{s} \in \prod_i \mathcal{S}_i$, and for each positive backbone $\mathbf{B}^j \in \mathbf{B}^+$, a positive state conformation $\mathbf{r}^j(\mathbf{s}) \in \prod_i \mathcal{R}_i^j(\mathbf{s}[i])$ such that any negative state conformation $\mathbf{r}^j(\mathbf{s}) \in \prod_i \mathcal{R}_i^j(\mathbf{s}[i])$ on each negative backbone $\mathbf{B}^j \in \mathbf{B}^-$ gives a difference in energy which is below a given threshold:

$$\sum_{\mathbf{B}^j \in \mathbf{B}^+} \mathbf{E}^j(\mathbf{r}^j(\mathbf{s})) - \sum_{\mathbf{B}^j \in \mathbf{B}^-} \mathbf{E}^j(\mathbf{r}^j(\mathbf{s})) \leqslant k$$

In other word, in order to decide the NegDes problem, a sequence and positive state conformations must be found, such that the previous inequality is verified for all negative state conformations that are consistent with the sequence. As there are an exponential number of conformations for a given sequence, an exponential number of inequalities must be verified.

Theorem 8. NegDes is Σ_2^{P}-complete. (*i.e.*, $\mathrm{NP}^{\mathrm{NP}}$-complete).

Proof. We must prove that 1. NegDes belongs to the class $\mathrm{NP}^{\mathrm{NP}}$; 2. any problem in $\mathrm{NP}^{\mathrm{NP}}$ reduces to NegDes in polynomial time.

First, let us notice that for any negative conformations $\mathbf{r}^j(\mathbf{s}) \in \prod_i \mathcal{R}_i^j(\mathbf{s}[i]), \mathbf{B}^j \in \mathbf{B}^-$, the condition

$$\sum_{\mathbf{B}^j \in \mathbf{B}^+} \mathbf{E}^j(\mathbf{r}^j(\mathbf{s})) - \sum_{\mathbf{B}^j \in \mathbf{B}^-} \mathbf{E}^j(\mathbf{r}^j(\mathbf{s})) \leqslant k$$

is equivalent to:

$$\sum_{\mathbf{B}^j \in \mathbf{B}^+} \mathbf{E}^j(\mathbf{r}^j(\mathbf{s})) \leqslant k + \sum_{\mathbf{B}^j \in \mathbf{B}^-} \mathbf{E}^j(\mathbf{r}^j(\mathbf{s}))$$

Hence NegDes is equivalent to: "Does there exist a sequence $\mathbf{s} \in \mathcal{S}_i$ such that: $\sum_{\mathbf{B}^j \in \mathbf{B}^+} \min_{\mathbf{r}^j(\mathbf{s}) \in \mathcal{R}^j(\mathbf{s})} \mathbf{E}^j(\mathbf{r}^j(\mathbf{s})) \leqslant k + \sum_{\mathbf{B}^j \in \mathbf{B}^-} \min_{\mathbf{r}^j(\mathbf{s}) \in \mathcal{R}^j(\mathbf{s})} \mathbf{E}^j(\mathbf{r}^j(\mathbf{s}))$?"

Assuming we had an NP-oracle, that could solve any instance of SSD, it would be possible to verify a certificate given by a sequence $\mathbf{s} \in \prod_i \mathcal{S}_i$.

It is possible to check wether the certificate is positive by calling the SSD NP-oracle to compute the minimum conformation energy on all states $\mathbf{B}^j \in \mathbf{B}$: $\min_{\mathbf{r}^j(\mathbf{s}) \in \mathcal{R}^j(\mathbf{s})} \mathbf{E}^j(\mathbf{r}^j(\mathbf{s}))$, and combining these energy to check:

$$\sum_{\mathbf{B}^j \in \mathbf{B}^+} \min_{\mathbf{r}^j(\mathbf{s}) \in \mathcal{R}^j(\mathbf{s})} \mathbf{E}^j(\mathbf{r}^j(\mathbf{s})) \leqslant k + \sum_{\mathbf{B}^j \in \mathbf{B}^-} \min_{\mathbf{r}^j(\mathbf{s}) \in \mathcal{R}^j(\mathbf{s})} \mathbf{E}^j(\mathbf{r}^j(\mathbf{s}))$$

The following $\mathrm{NP}^{\mathrm{NP}}$-complete problem, called ∃∀3DNF was introduced in Chapter 1:

Given two sets of propositional variables $\mathbf{p} = (p_1, \ldots, p_n)$ *and* $\mathbf{q} = (q_1, \ldots, q_n)$*, and a boolean formula over these variables* $H(\mathbf{p}, \mathbf{q})$ *in disjunctive normal form (DNF), with each cube a conjunction of three literals, is there a valuation* $\nu_{\mathbf{p}}$ *of* $\mathbf{p}$ *such that for any valuation* $\nu_{\mathbf{q}}$ *of* $\mathbf{q}$ *of the variables of* $\mathbf{q}$*,* $\nu_{\mathbf{p}}\nu_{\mathbf{q}}(H(\mathbf{p}, \mathbf{q}))$ *is true?*

Let us reduce ∃∀3DNF to NegDes. Let $H(\mathbf{p}, \mathbf{q}) = C_1 \vee \cdots \vee C_k$, with $\mathbf{p} = (p_1, \ldots, p_n), \mathbf{q} = (q_1, \ldots, q_m)$ be a ∃∀3DNF instance. For each $1 \leqslant i \leqslant k$, let l_i^1, l_i^2, l_i^3 denote de literals of C_i. Let us construct an instance of NegDes with $n + m + k$ variables, represented as a CFN:

- For each variable $p \in \mathbf{p}$ (resp. $q \in \mathbf{q}$), we introduce the variable V_p (respt. V_q) with domain $\{T, F\}$, representing the valuation of p (respt. q).
- For each cube C_i of $H(\mathbf{p}, \mathbf{q})$, we introduce the variable V_{C_i} with domain $\{l_i^1, l_i^2, l_i^3\}$

For each cube C_i and each variable $x \in \mathbf{p} \cup \mathbf{q}$ that appears in C_i, the binary cost between V_{C_i} and V_x is defined as follows:

$$E_{x,C_i} : (v, l_i) \in \mathbf{D}_{V_x} \times \mathbf{D}_{V_{C_i}} \longmapsto \begin{cases} 1 & \text{if } v = T \text{ and } l_i = x \\ 1 & \text{if } v = F \text{ and } l_i = \neg x \\ 0 & \text{otherwise.} \end{cases}$$

If the literal l_i is satisfied by the valuation v of its variable x, the corresponding binary cost is 1. The total energy is the sum of the binary

terms: $E = \sum_{x,C_i} E_{x,C_i}$. Given an assignment on all variables, the energy is zero if and only if all binary terms are zero. This means that for each cube C_i, the variable V_{C_i} is assigned a literal l_i that is not satisfied by the valuation $\nu_{\mathbf{p}}\nu_{\mathbf{q}}$ defined by the assignment. Finally, we consider the NegDes instance with a single negative backbone:

"Does there exist $\mathbf{s} \in \prod_i \mathbf{D}_{V_{p_i}}$ such that:"

$$-\left(\min_{\mathbf{r}\in\prod_i \mathbf{D}_{V_{C_i}}\times\prod_j \mathbf{D}_{V_{Q_j}}} E(\mathbf{s},\mathbf{r})\right) \leqslant -1$$

- If the $\exists\forall$3DNF instance is true, there exists a valuation $\nu_{\mathbf{p}}$ such that $\nu_{\mathbf{p}}H(\mathbf{p},\mathbf{q})$ is a tautology. So, if $\mathbf{s}$ is the assignment of variables $V_p, p \in \mathbf{p}$ that corresponds to $\nu_{\mathbf{p}}$, then, for any assignment of $V_q, q \in \mathbf{q}$, there always exists a cube C_i, all of whose literals are satisfied, hence the binary cost is greater than 1. Or equivalently:

  $$\min_{\mathbf{r}\in\prod_i \mathbf{D}_{V_{C_i}}\times\prod_j \mathbf{D}_{V_{Q_j}}} E(\mathbf{s},\mathbf{r}) \geqslant 1$$

  Therefore, the NegDes instance is true.

- Conversely, if the NegDes instance is true, there exists $\mathbf{s} \in \prod_i \mathbf{D}_{V_{p_i}}$ corresponding to a valuation $\nu_{\mathbf{p}}$, such that the energy of any assignment of the remaining variables is greater than 1, meaning that $H(\mathbf{p},\mathbf{q})$ is a tautology.

Note that the NegDes instance consists of $n+m+k$ variables, each with domain size less than 3, and $k \cdot (n+m)$ binary energies, that can be described in $3 \cdot 2$-sized matrix, where each coefficient is straightforward to compute. Therefore, the reduction is valid and polynomial. □

These theorems show that the negative multistate problem is inherently harder to solve than SSD or positive MSD problems.

4.2 Positive multi-state design as a WCSP

The modeling of single state design as a cost function network was detailed in Chapter 2. SSD and positive MSD belong to the same complexity class. This section describes how positive MSD can be expressed as a CFN.

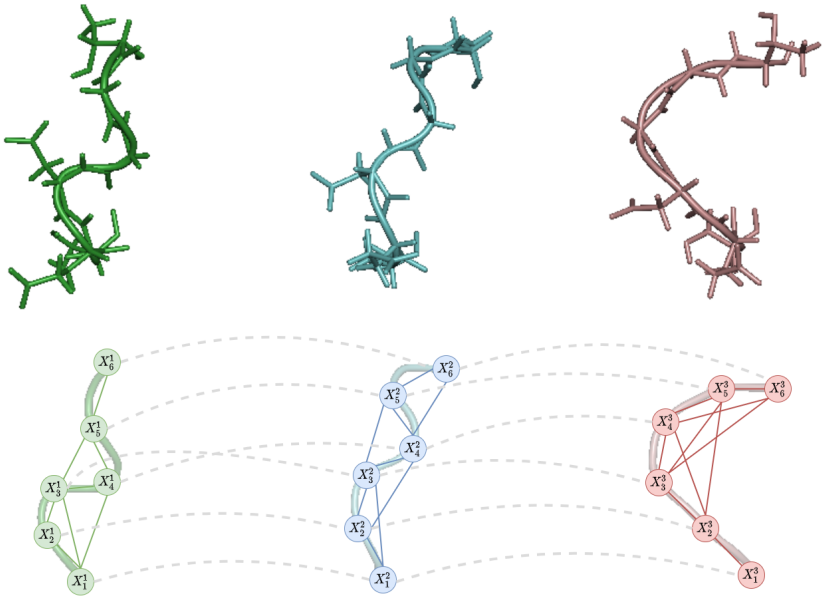


Figure 4.2: Schematic cost function network encoding positive multistate design on three conformations. Solid lines represent binary energy functions. Doted lines represent sequence constraints.

4.2.1 Positive MSD as a CFN

In positive MSD, a sequence $\mathbf{s}^*$ that best stabilizes all states $\mathbf{B}^j \in \mathbf{B}^+$ is sought, *i.e.*, such that:

$$\mathbf{s}^* = \underset{\mathbf{s}\in\prod_i \mathcal{S}_i}{\arg\min} \sum_{\mathbf{B}^j\in\mathbf{B}^+} \left[\min_{\mathbf{r}^j\in\mathcal{R}^j(\mathbf{s}[i])} \left[E^j_\varnothing + \sum_i E^j_i(\mathbf{r}^j[i]) + \sum_{i<i'} E^j_{ii'}(\mathbf{r}^j[i],\mathbf{r}^j[i']) \right] \right]$$

This problem can not be tackled by solving the SSD for each state $\mathbf{B}^j \in \mathbf{B}^+$ and summing the energies because state optimum sequences may differ. However, cost function networks have the ability to encode hard constraints, that can forbid partial assignments using cost $\top$. These constraints are used to ensure that all positions in different states correspond to the same amino acid. More precisely, the CFN $\mathcal{C}^+ = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ corresponding to the positive multistate instance is defined with:

- There is one variable per position i in each state $\mathbf{B}^j$:

$$\mathbf{X} = \{X^j_i | i \in [\![1, n]\!], \mathbf{B}^j \in \mathbf{B}\}$$

- The domain of the variable X^j_i corresponds to all possible rotamers in

position i on backbone $\mathbf{B}^j$:

$$\mathbf{D} = \{ \mathbf{D}_i^j | i \in [\![1, n]\!], \mathbf{B}^j \in \mathbf{B} \} \qquad \mathbf{D}_i^j = \mathcal{R}_i^j = \bigcup_{s \in \mathcal{S}_i} \mathcal{R}_i^j(s)$$

- Cost functions include all state energy terms $E_\varnothing^j$, $E_i^j, \forall 1 \leqslant i \leqslant n$, and $E_{ii'}^j, 1 \leqslant i < i' \leqslant n$. Moreover, for each position $1 \leqslant i \leqslant n$ and each pair of consecutive backbones $\mathbf{B}^j, \mathbf{B}^{j+1} \in \mathbf{B}$, a binary constraint-encoding cost function $SS_i^{j,j+1}$ ensures that rotamers in position i on states $\mathbf{B}^j$ and $\mathbf{B}^{j+1}$ share the same amino acid type. More precisely, for any pair of possible rotamers $(r, r') \in \mathbf{D}_{X_i^j} \times \mathbf{D}_{X_i^{j+1}}$, we have:

$$SS_i^{j,j+1}(r, r') = \begin{cases} 0 & \text{if } \exists s \in \mathcal{S}_i \text{ s.t. } r \in \mathcal{R}_i^j(s), r' \in \mathcal{R}_i^{j+1}(s) \\ \top & \text{otherwise.} \end{cases}$$

The resulting CFN has $l \cdot n$ variables, where $l = |\mathbf{B}|$ is the number of states, and n the number of sequence positions; and $l \cdot \frac{n \cdot (n+1)}{2}$ energy terms and $n \cdot (l-1)$ sequence constraints.

The minimum cost solution of this network defines a sequence and for each state, a conformation. By definition, the CFN cost is the sum of all state energy terms. Moreover, the sequence constraints ensure that all state conformations correspond to the same sequence.

Note that the expression of the positive MSD as a single CFN enables the enumeration of suboptimal sequences using SCP branching [Traoré *et al.* 2016].

4.2.2 Experiments

The experiments presented here were performed by Jelena Vucinic. They are published, along with others in [Vucinic *et al.* 2020]. They show that positive multistate design enhances predicted sequence qualities, compared to single state design. A comparison with iCFN [Karimi & Shen 2018] also shows that the single CFN modeling of MSD enables to tackle much larger instances.

4.2.2.1 Benchmark preparation

The benchmark consists of 30 protein structures that have been extracted from the PDB. Half of these structures are NMR resolved, that contain at least 20 conformations. The other half is X-ray resolved, and 100 conformations were generated using the Backrub ensemble method for flexible protein backbone modeling in Rosetta [Davis *et al.* 2006].

For each structure, sets of conformations were submitted to an RMSD-based hierarchical clustering [Berenger *et al.* 2012], and the four most diverse conformations were selected. Full redesign was performed, and energy matrices were computed using Dunbrack rotamer library [Shapovalov & Dunbrack Jr 2011], and Rosetta beta_nov_16 scoring function [Alford *et al.* 2017].

The CFNs were solved using the solver Toulbar2. Dead-end elimination was deactivated, the tree-decomposition was used, with VAC bounds. Computations were done on an Intel(R) Xeon(R) CPU E5-2630 at 2.30GHz with 24 GB of RAM.

4.2.2.2 Comparing SSD and MSD

In order to compare the design qualities, the native sequence recovery (NSR) and native sequence similarity recovery (NSSR) were used (see Chapter 2). For SSD, the full redesign was performed on each backbone, and the NSR and NSSR have been computed as the average over the four conformations. Results are given in table 4.1.

MSD achieves on average an NSR of 64,7% and 66,4% and an NSSR of 74,4% and 73,9% for respectively back rubbed X-ray and NMR structure datasets. For all but one NMR structure, MSD provides the best native sequence recovery. The p-value of a Wilcoxon signed-rank test comparing average SSD and MSD was $2,5 \cdot 10^{-6}$. MSD performed 15,6% and 8% better than SSD on NMR structures and X-ray structures respectively. The better performance of MSD, as the optimization of the sequence on all states, on NMR structures could be explained by the fact that the NMR ensemble do correspond to likely states of the observed protein, and back-rubbed X-ray structures just define a set of possible states.

4.2.3 CPU-time analysis

The CPU-time for solving single and multi-state instances depending on the instance size (in bytes of the compressed file that describes it) is plotted in Figure 4.3. Empirically, the CPU-time grows exponentially with the instance size, which is consistent with the NP-completeness of the single and multi-state problems. Given their larger size, MSD problems seem simpler to solve than SSD problems. Overall, experiments show that full single or multi-state redesigns can be exactly solved on a standard computer in reasonable time, for proteins of size < 100 amino acids.

	Search space		NSR (%)		NSSR (%)	
PDB ID	SSD	MSD	SSD	MSD	SSD	MSD
5vso	$1.3 \cdot 10^{181}$	$8.5 \cdot 10^{723}$	48.0	62.7	58.6	70.6
5l7b	$2.6 \cdot 10^{170}$	$6.4 \cdot 10^{680}$	52.9	59.5	69.6	75.4
5mmc	$5.6 \cdot 10^{158}$	$4.1 \cdot 10^{634}$	54.2	58.5	59.6	61.5
1gb1	$2.5 \cdot 10^{137}$	$5.9 \cdot 10^{547}$	48.7	60.7	62.9	67.8
5t8a	$9.4 \cdot 10^{133}$	$5.9 \cdot 10^{535}$	39.7	37.7	52.5	49.2
2l5t	$2.2 \cdot 10^{185}$	$7.2 \cdot 10^{738}$	58.4	83.1	71.1	90.9
2n6r	$3.4 \cdot 10^{168}$	$4.0 \cdot 10^{673}$	53.9	67.1	64.5	73.7
1bmw	$5.2 \cdot 10^{229}$	$1.1 \cdot 10^{914}$	53.4	79.8	64.9	84.1
5ix5	$4.4 \cdot 10^{148}$	$2.0 \cdot 10^{593}$	52.6	63.2	62.8	72.1
5x0s	$1.2 \cdot 10^{119}$	$1.4 \cdot 10^{470}$	42.9	58.5	51.9	75.5
6ews	$8.1 \cdot 10^{155}$	$4.3 \cdot 10^{622}$	46.8	69.8	65.8	82.5
6fwn	$2.8 \cdot 10^{188}$	$2.4 \cdot 10^{751}$	55.8	72.9	63.2	78.8
6qf8	$2.3 \cdot 10^{188}$	$4.0 \cdot 10^{750}$	54.3	68.4	64.4	77.6
6jlt	$6.9 \cdot 10^{188}$	$1.5 \cdot 10^{755}$	54.9	65.7	62.5	71.1
6hkc	$4.5 \cdot 10^{174}$	$6.6 \cdot 10^{694}$	45.0	66.6	59.3	78.6
Average			50.8	66.4	62.2	73.9

(a) NMR structures

	Search space		NSR (%)		NSSR (%)	
PDB ID	SSD	MSD	SSD	MSD	SSD	MSD
1hyp	$2.1 \cdot 10^{166}$	$3.2 \cdot 10^{664}$	61.8	68.9	71.3	81.1
1hoe	$2.2 \cdot 10^{171}$	$5.8 \cdot 10^{684}$	60.1	77.0	67.2	82.4
1mjc	$4.3 \cdot 10^{165}$	$10.0 \cdot 10^{661}$	56.5	59.4	67.5	69.6
1pga	$4.7 \cdot 10^{137}$	$1.5 \cdot 10^{550}$	52.7	73.2	62.9	80.3
2b8i	$7.7 \cdot 10^{189}$	$5.1 \cdot 10^{758}$	48.4	57.1	63.3	71.4
4y2k	$2.5 \cdot 10^{161}$	$1.7 \cdot 10^{644}$	62.7	70.8	66.1	75.4
1f94	$2.9 \cdot 10^{134}$	$1.4 \cdot 10^{537}$	64.3	71.4	74.2	80.9
1who	$2.6 \cdot 10^{227}$	$6.1 \cdot 10^{907}$	61.4	68.1	72.1	80.9
1tud	$3.1 \cdot 10^{146}$	$5.0 \cdot 10^{585}$	45.8	48.3	55.0	55.0
1yu5	$9.4 \cdot 10^{165}$	$2.9 \cdot 10^{663}$	64.1	68.6	72.4	74.6
1bxy	$5.8 \cdot 10^{147}$	$7.6 \cdot 10^{590}$	53.3	60.0	64.5	73.3
1ctf	$5.2 \cdot 10^{164}$	$1.9 \cdot 10^{658}$	57.9	60.9	65.6	68.1
1guu	$1.2 \cdot 10^{123}$	$1.4 \cdot 10^{492}$	53.4	66.6	69.6	82.4
1wvn	$8.0 \cdot 10^{179}$	$4.5 \cdot 10^{718}$	41.9	50.0	57.1	62.2
1ucs	$5.9 \cdot 10^{153}$	$7.9 \cdot 10^{614}$	66.1	70.3	73.0	78.1
Average			56.7	64.7	66.8	74.4

(b) X-ray structures

Table 4.1: Native sequence recoveries and native similarity sequence recoveries for SSD and MSD on both NMR (a) and X-ray (b) structure datasets. For each problem, the average SSD and MSD search space sizes are given (the search space size is defined as the product of all variable domain sizes).

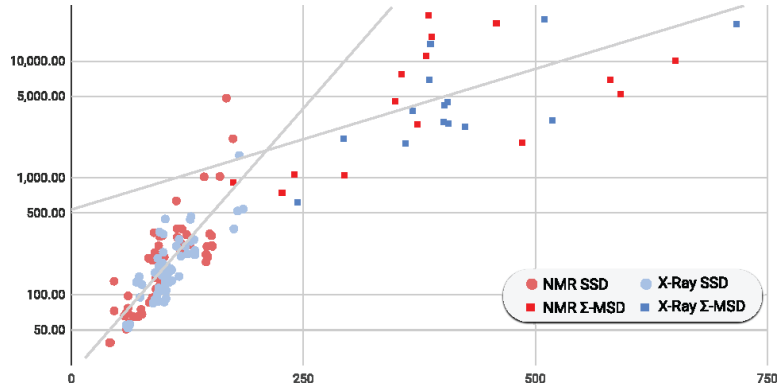


Figure 4.3: CPU-time (s) versus problem size (MB) for SSD and MSD problems. Each point represents an instance. Plot reprinted from [Vucinic *et al.* 2020].

4.2.3.1 Comparison with iCFN

The recent iCFN solver [Karimi & Shen 2018] is not able to solve positive MSD as it is defined here, with fitness defined as the average state energy. It solves the min-MSD problem, which optimizes the minimum state energy. [Vucinic *et al.* 2020] converted the MSD instances from [Karimi & Shen 2018] and solved them with Toulbar2, which was much faster than iCFN, by a non-constant factor that increases with problem size.

The benchmark problems presented above were also converted to the iCFN format, but none of them could be solved by iCFN after 65 hours, even for the smallest problem 1pga, which is solved by Toulbar2 in less than 20 minutes. Several design problems were therefore prepared, with a decreasing number of mutable residues. It was only when the number of mutable residues was reduced to 5, and all other positions were fixed that iCFN was able to provide a solution in less than 24 hours. Then, iCFN produced the solution in 247s. Toulbar2 solved the problem in 14,57s.

4.3 Negative multi-state design

4.3.1 Negative MSD as a bilevel WCSP

In negative multistate design, the input consists of two sets of desired $\mathbf{B}^+$ and undesired $\mathbf{B}^-$ backbones $\mathbf{B} = \mathbf{B}^+ \cup \mathbf{B}^-$. Each backbone $\mathbf{B}^j \in \mathbf{B}$ is associated with a pairwise decomposable energy function denoted $\mathbf{E}^j$. As stated in Section 4.1, the task at hand is the identification of a sequence $\mathbf{s}^*$

such that:

$$\mathbf{s}^* = \arg\min_{\mathbf{s}\in\prod_i \mathcal{S}_i} \left[\left(\sum_{\mathbf{B}^j\in\mathbf{B}^+} \min_{\mathbf{r}^j(\mathbf{s})\in\mathcal{R}^j(\mathbf{s})} \mathbf{E}^j(\mathbf{r}^j(\mathbf{s})) \right) - \left(\sum_{\mathbf{B}^j\in\mathbf{B}^-} \min_{\mathbf{r}^j(\mathbf{s})\in\mathcal{R}^j(\mathbf{s})} \mathbf{E}^j(\mathbf{r}(\mathbf{s})) \right) \right]$$

Positive multistate design can be expressed as a WCSP, by combining backbone networks, and adding same sequence constraints. Therefore, with $\mathbf{E}^+ = \sum_{\mathbf{B}^j\in\mathbf{B}^+} \mathbf{E}^j$ and $\mathbf{E}^- = \sum_{\mathbf{B}^j\in\mathbf{B}^-} \mathbf{E}^j$ the aggregated energy functions on the positive and negative states respectively; and $\mathcal{C}^+$ and $\mathcal{C}^-$ the cost function networks encoding these functions: $C_{\mathcal{C}^+} = \mathbf{E}^+$, $C_{\mathcal{C}^-} = \mathbf{E}^-$, the problem can be rewritten:

$$\mathbf{s}^* = \arg\min_{\mathbf{s}\in\prod_i \mathcal{S}_i} \left[\min_{\mathbf{r}^+(\mathbf{s})\in\mathcal{R}^+(\mathbf{s})} \mathbf{E}^+(\mathbf{r}^+(\mathbf{s})) - \min_{\mathbf{r}^-(\mathbf{s})\in\mathcal{R}^-(\mathbf{s})} \mathbf{E}^-(\mathbf{r}^-(\mathbf{s})) \right]$$

where $\mathcal{R}^+(\mathbf{s})$ ($\mathcal{R}^-(\mathbf{s})$ respectively) is the set of conformations of the sequence $\mathbf{s}$ on all positive (negative respectively) backbones:

$$\mathcal{R}^+(\mathbf{s}) = \prod_{\mathbf{B}^j\in\mathbf{B}^+} \prod_{i=1}^{n} \mathcal{R}_i^j(\mathbf{s}[i]) \qquad \mathcal{R}^-(\mathbf{s}) = \prod_{\mathbf{B}^j\in\mathbf{B}^-} \prod_{i=1}^{n} \mathcal{R}_i^j(\mathbf{s}[i])$$

Two player game The negative design problem can be seen as a two-player game. Player 1 chooses a sequence and corresponding conformations on its positive backbones. Player 2 then chooses conformations on its negative backbones, that match the first player's sequence. Player 1 aim is to minimize its conformation energy while maximizing its adversaries. The goal of player 2 is to have a low energy conformation.

The answer to the negative design query corresponds to a game where both players played perfectly.

Expressing feasibility Given a sequence $\mathbf{s}$, a conformation $\mathbf{r}$ is *feasible* if its rotamers correspond to the amino acids of $\mathbf{s}$. In cost function networks, feasibilities can be expressed by constraint-encoding cost functions, that take values 0 or $\top$. Let us denote by SS the sequence constraint-encoding cost function:

$$\mathbf{s}^* = \arg\min_{\mathbf{s}\in\mathcal{S}} \min_{\mathbf{r}^+\in\mathcal{R}^+} \mathbf{E}^+(\mathbf{r}^+) + SS(\mathbf{s},\mathbf{r}^+) - \min_{\mathbf{r}^-\in\mathcal{R}^-} \mathbf{E}^-(\mathbf{r}^-) + SS(\mathbf{s},\mathbf{r}^-)$$

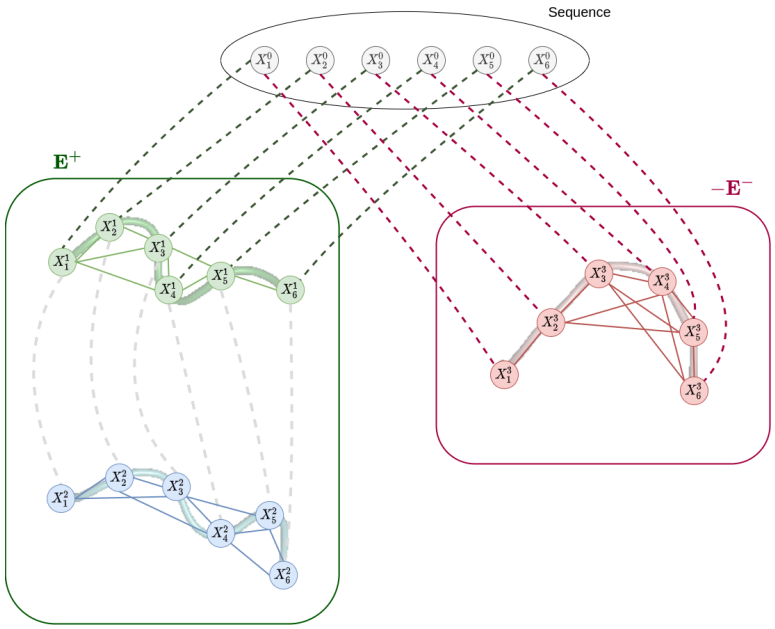


Figure 4.4: Structure of a CFN representing the fitness function of the negative design with two positive states and one negative state.

where $\mathcal{R}^+$ ($\mathcal{R}^-$ respectively) is the set of all possible conformations on all positive (negative respectively) backbones:

$$\mathcal{R}^+ = \prod_{\mathbf{B}^j \in \mathbf{B}^+} \prod_{i=1}^{n} \mathcal{R}_i^j \qquad \mathcal{R}^- = \prod_{\mathbf{B}^j \in \mathbf{B}^-} \prod_{i=1}^{n} \mathcal{R}_i^j$$

Note that there always exists a conformation $\mathbf{r}^-$ such that $\mathbf{E}^-(\mathbf{r}^-) < \top$. So, a "legal" conformation will always be chosen in the negative state energy minimization, and minimization over $\mathbf{r}^-$ can never produce a cost equal to $-\top$.

4.3.1.1 Branch-and-bound on the sequence space

Given any sequence $\mathbf{s}$, and corresponding conformations $\mathbf{r}^j$ on positive and negative backbones, if $l = |\mathbf{B}|$, the objective value is :

$$\mathbf{E}(\mathbf{s}, \mathbf{r}^1, \ldots, \mathbf{r}^l) = \sum_{\mathbf{B}^j \in \mathbf{B}^+} \mathbf{E}^j(\mathbf{r}^j) + SS(\mathbf{s}, \mathbf{r}^j) + \sum_{\mathbf{B}^j \in \mathbf{B}^-} -\mathbf{E}^j(\mathbf{r}^j) + SS(\mathbf{s}, \mathbf{r}^j)$$

This objective function can be expressed as a cost function network $\mathcal{C}^{seq}$ as follows:

- It has variables representing the sequence, the positive conformations,

and the negative conformations:

  - For each position i in the protein sequence, the variable X_i^{seq} represents the amino acid type at position i
  - For each position i and backbone $\mathbf{B}^j \in \mathbf{B}^+ \cup \mathbf{B}^-$, the variable X_i^j represents the rotamer of position i on backbone $\mathbf{B}^j$.

- Domains of sequence variables are the allowed amino acid in the corresponding position:

$$\mathbf{D}_{X_i^{seq}} = \mathcal{S}_i$$

  The domain of the rotamer variable X_i^j is all the possible rotamers at position i on backbone $\mathbf{B}^j$:

$$\mathbf{D}_{X_i^j} = \mathcal{R}_i^j$$

- The cost functions consist of energy terms from positive backbone energies, opposites of energy terms from negative backbone energies, and same sequence constraints between sequence variables and backbone variables from *one* positive backbone, denoted SS^+, and between sequence variables and conformation variables from *one* negative backbone, denoted SS^-:

$$\begin{aligned}
&\left\{ E_i^j, E_{ii'}^j | i, i' \in [\![1, n]\!], \mathbf{B}^j \in \mathbf{B}^+ \right\} \\
\cup &\left\{ -E_i^j, -E_{ii'}^j | i, i' \in [\![1, n]\!], \mathbf{B}^j \in \mathbf{B}^- \right\} \\
\cup &\left\{ SS_i^+ |\ i \in [\![1, n]\!] \right\} \\
\cup &\left\{ SS_i^- |\ i \in [\![1, n]\!] \right\}
\end{aligned}$$

Figure 4.4 gives a schematic representation the structure of such a network.

Local consistencies on $\mathcal{C}^{seq}$ The lower bound $c_\varnothing^{seq}$ in $\mathcal{C}^{seq}$ provides a lower bound on the objective function, that can be improved by local consistency enforcements. Necessary propagations might move costs between the opposite negative part $-\mathbf{E}^-$ and the sequence and positive part of the network, through the sequence constraint-encoding functions. The total amount of cost that exited $-\mathbf{E}^-$ is stored in Δ_{neg} (it might be negative if costs were projected to $-\mathbf{E}^-$). As detailed later, this quantity will be useful to retrieve total assignment objective values.

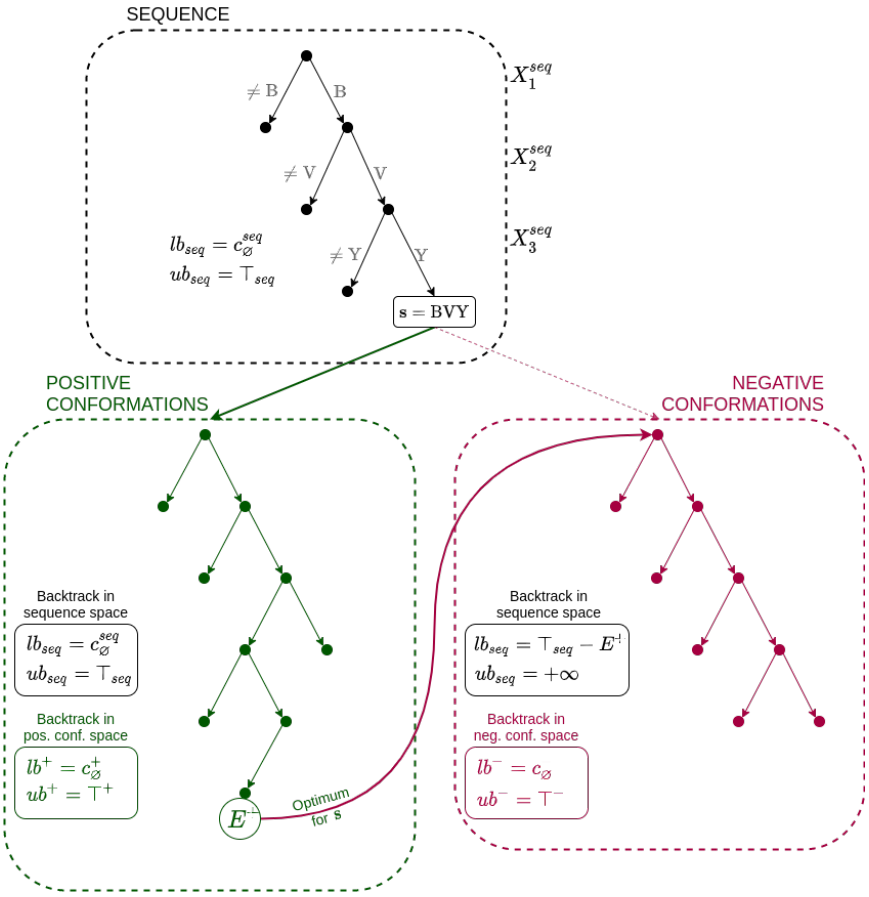


Figure 4.5: Schematic illustration of the bilevel search. Once a full sequence assignment is reached, the positive conformation space is explored to optimality with bounds for sequence and positive conformation spaces. Once the optimum E^+ is found, the negative conformation space is exlored to optimality, with bounds for sequence and negative conformation spaces.

4.3.1.2 Exploration of the sequence space

Assignments on sequence variables are explored, in a branch-and-bound manner: a lower bound on the fitness function, for any choice of conformation is:

$$\min_{\mathbf{r}^+ \in \mathcal{R}^+} \min_{\mathbf{r}^- \in \mathcal{R}^-} \left(\mathbf{E}^+ - \mathbf{E}^- \right)$$

A lower bound on this function can be computed easily and incrementally by enforcing local consistencies on the network $\mathcal{C}^{seq}$ (encoding $\mathbf{E}^+ - \mathbf{E}^-$) and taking its lower bound $c_\emptyset^{seq}$.

Every time a full sequence, and the positive and negative optimum conformations are explored, the corresponding fitness value becomes an

upper bound on the best fitness.

Once the sequence is fully assigned, the problem becomes:

$$\left(\min_{\mathbf{r}^+} \mathbf{E}^+(\mathbf{r}^+)\right) - \left(\min_{\mathbf{r}^-} \mathbf{E}^-(\mathbf{r}^-)\right)$$

Each minimization can be performed independently from the other.

4.3.1.3 Positive conformation search

Once the sequence is fully assigned, the best positive conformation is sought. Network $\mathcal{C}^{seq}$ is still used, as the branch-and-bound search tree can be pruned in two ways:

Bilevel pruning As for the sequence level search, the lower bound $c_{\varnothing}^{seq}$ in the network $\mathcal{C}^{seq}$ is a lower bound on the optimum value of the sequence objective function. If this lower bound exceeds the global upper bound in a node, it means that the corresponding sequence assignment does not provide a good enough positive conformation. The sequence assignment does not need to be explored further: we can backtrack in the sequence space.

Simple pruning All sequence variables are assigned, so the subnetwork corresponding to the positive backbone variables and functions can be solved as a "classic" network: its empty-scoped cost function is a lower bound on the solution costs, and each time a new better set of conformations is found, it provides a new upper bound on the optimum positive conformation energy. When the lower bound is higher than the upper bound, the current branch can be pruned and backtrack happens, in the positive conformation space.

4.3.1.4 Negative conformation search

Let us assume that the best sequence $\mathbf{s}^*$ seen so far, with best positive and negative conformations has a fitness value E^*; and that the search lead to a new sequence $\mathbf{s}$, for which we found an optimum positive conformation $\mathbf{r}^+$. Same sequence constraints are propagated in the negative conformation network, and optimum conformations on negative states are sought, using branch-and-bound and local consistency pruning. We want the new fitness $\mathbf{E}(\mathbf{s})$ to be better than the current best one E^*, *i.e.*, we want to find an optimum negative conformation $\mathbf{r}^-$ such that:

$$\mathbf{E}^+(\mathbf{r}^+) - \mathbf{E}^-(\mathbf{r}^-) < E^*$$

This is equivalent to:

$$\mathbf{E}^-(\mathbf{r}^-) > E^* - \mathbf{E}^+(\mathbf{r}^+)$$

The value $E^* - \mathbf{E}^+(\mathbf{r}^+)$ provides a lower bound for the exploration of the negative conformations: if we find a solution with lower energy, the conformation search can stop and backtrack happens in the sequence space.

Intuitively, if negative conformations are too stable, then there exists another sequence, that better favors positive states compared to negative states. The current sequence is therefore not worth exploring.

A schematic illustration of the sequence and conformation search trees can be found in Figure 4.5.

4.3.1.5 Retrieving the fitness value

Let us assume that sequence, positive, and negative conformation variables have been assigned, with respective cost values c^{seq}, c^+, and c^-.

Remember that in order to prune the sequence conformation space, costs have propagated in $\mathcal{C}^{seq}$, possibly transferring costs between sequence, positive, and opposite negative subnetworks. The amount of costs that were transferred from the opposite negative functions to the sequence functions was stored in Δ_{neg}, and needs to be subtracted from the cost value:

$$\mathbf{E} = c^{seq} + c^+ - c^- - \Delta_{neg}$$

4.3.2 Preliminary tests

The bilevel strategy was integrated into the ToulBar2 solver. This Subsection presents preliminary results regarding its efficiency. Computations were done on an Intel Xeon CPU E5-2687W at 2.30GHz.

4.3.2.1 Comparison with iCFN

We compared our bilevel resolution method with the recent iCFN solver [Karimi & Shen 2018]. We used one positive and one negative state from three multistate instances available on the iCFN website (see the supplementary materials of [Karimi & Shen 2018]). They correspond to the negative redesign of a T-cell receptor, and positions 26, 28, and 100 are muted one by one. The iCFN instances were converted to ToulBar2 CFN format and were combined into bilevel instances. Results are presented in Table 4.2. On the three tested instances, our approach is five to ten times faster.

Mutable position	**26**	**28**	**100**
N_{flex}	18	18	21
D	3135	2122	2653
iCFN	103.37	81.50	62.07
Toulbar2	8.52	17.23	11.28

Table 4.2: Comparison of the CPU-times (in seconds) for iCFN and Toulbar2-bilevel solving, for solving the negative design, with one positive and one negative state. N_{flex} is the number of flexible positions, besides the one mutable residue. D is the maximum domain size.

N_{mut}	**1**	**2**	**3**	**4**	**5**
N_{flex} ($d = 3$ Å)	5	7	8	12	13
D	94	163	175	217	217
Toulbar2	**0.021**	**0.051**	**0.910**	26.760	586.699
Toulbar2 + VAC	0.024	0.056	0.934	**26.508**	**577.772**

N_{mut}	**1**	**2**	**3**	**4**	**5**
N_{flex} ($d = 6$ AA)	15	26	29	42	48
D	275	283	283	283	283
Toulbar2	**0.229**	**1.000**	**17.865**	570.710	701.27
Toulbar2 + VAC	0.292	1.146	19.01	**558.828**	**672.973**

Table 4.3: Computation times (in seconds) for different instance sizes of the negative multi-state design for binding energy minimization, Toulbar2, and with Toulbar2 and VAC enforcement as a preprocessing step. d is the distance from mutable residues within which other residues are flexible, N_{mut} is the number of mutable residues, N_{flex} is the number of flexible residues, and D the maximum domain size.

4.3.2.2 Binding energy minimization

We confronted our solving method to the binding energy minimization of a protein complex: XRCC1 N-term domain and DNA polymerase beta. The positive state is the bound structure (PDB code: 3k75) and the negative state the unbound state (PDB code: 3lqc). We incrementally set the N_{mut} first residues to be mutable, among positions 391, 409, 411, 422, and 424. For each combination of mutable residues, the residues within $d = 3$ or 6 Å of mutable residues are set to be flexible (N_{set} denotes the number of flexible positions). The Dunbrack library [Shapovalov & Dunbrack Jr 2011] and Rosetta ref2015 scoring function [Park *et al.* 2016] were used, for energy matrices computation. We solve the negative design problem to optimality. We compared resolution efficiencies of Toulbar2 with default local consistencies (EDAC enforced at each search node) and with VAC enforced as a preprocessing step. Computation times are presented in Table 4.3.

Smallest instances are solved instantaneously. Computation time increases exponentially with the instance size, as expected given the $\mathrm{NP}^{\mathrm{NP}}$-completeness of the negative design problem. However, even the largest instance, that corresponds to the bilevel exploration of a $9 \cdot 10^{13}$ sequence and $4.8 \cdot 10^{117}$ conformation search space is solved in less than 12 minutes. As expected, VAC enforcement as a preprocessing step becomes useful as the instance size becomes larger: then, time spent enforcing VAC is balanced out by time gains in conformational search.

VAC is particularly efficient for solving side chain packing problems, *i.e.*, searching conformations once the amino acid sequence is set. We suspect that exponential gains could result from VAC enforcement before each positive and negative conformation search.

Conclusion

This Chapter presented adapted CFN-based algorithms for both *positive* and *negative* multi-state design.

We have shown that positive MSD is not harder than the NP-complete SSD problem. It captures some of the flexibility of proteins as it enables the stabilization of an ensemble of backbone states, representing small movements around an optimal conformation or resulting from larger conformational switches. Positive MSD can be reduced to a discrete optimization problem, modeled as a cost function network, and solved with efficient dedicated algorithms. Computational experiments showed that this method outperforms the recent guaranteed multi-state algorithm iCFN and that MSD clearly enhances prediction quality. Full multi-state redesigns of proteins of size close to 100 amino acids became accessible. Because it just relies on a reduction to

a cost function network, MSD inherits all the capabilities of CFN solvers like Toulbar2, such as the exhaustive sequence enumeration within a threshold of the optimum, or the diversity control from sequences of interest and the greedy prediction of diverse good quality sequences, as presented in Chapter 3.

Negative multi-state design was shown to be intrinsically much harder than SSD and positive MSD, as it requires a second level of energy optimization. However, negative states are crucial when the design target is to generate specificity. We proposed a CFN-based bilevel optimization method, that uses existing local consistencies for lower and upper bounding. Preliminary tests suggest that this approach has the potential to greatly outperform existing exact negative design methods, such as iCFN, and to exactly solve bilevel instances of unprecedented size.

This potential is yet to be confirmed, and computational and experimental assessment of the quality of negative MSD prediction will be needed, to validate the benefits of the use of negative over positive design. Besides, useful CFN capabilities, such as diversity control and exhaustive or diverse sequence enumeration will need to be adapted to the bilevel optimization. Another useful improvement would be the addition of maximum energy degradation thresholds, on positive and negative states. This would enable for example the binding energy minimization while maintaining the stability of both bound (positive) and unbound (negative) states.

Conclusion

This thesis contributed to the development of algorithms for the weighted constraint satisfaction problem, mostly motivated by energy minimization in computational protein design applications.

Because the optimized function is never a perfect representation of real and often intractable problems, a dissimilarity constraint was developed, along with an efficient method for the generation of a library of solutions, with guarantees both on solution diversity and quality. The diversity constraint can be used to embed solution knowledge into the network. This is particularly useful in CPD applications where patented or native sequences exist.

The greedy diverse solution generation was shown to provide a large efficiency improvement compared to the exact diverse set of minimum cost solutions, with little side-effects in terms of cost. It was able to produce diverse solutions on large Bayesian networks, and on large CPD instances in reasonable time. Diverse protein sequences have a better chance of presenting a good quality solution compared to simple minimum energy enumeration. This remains true even if the exact resolution of the approximate criterion is relaxed, and a controlled gap to the optimum is tolerated. Our method is very useful to efficiently explore the search space and provide a diverse library of potential sequences for experimental testing.

Several simplifications are made on the single state computational protein design problem to make it reasonable, and enable its representation as a cost function network. These discretization assumptions put aside protein flexibility. One way of reintegrating flexibility information is to describe the target structure with several backbones, or states. This thesis states that the positive multi-state design approach remains decision NP-complete and is therefore not harder than the already efficiently solved single state design problem. Cost function network modeling and concise in silico results of multi-state designs are mentioned.

Although negative design is one step higher in the polynomial hierarchy, it is crucial for the design of protein-ligand binding or oligomeric association specificity. Input backbones are divided into the desired states, which are to be stabilized, and the undesired, which are to be avoided. The corresponding fitness function can still be represented as a cost function network, and therefore benefits from equivalent preserving transformations that enforce local consistencies. However, the query becomes bilevel. Solving branch-and-bound algorithms were therefore adapted to tackle the required bilevel optimization.

Perspectives

Regarding our work on diversity, the $\text{Div}_{\min}$ constraint that gathers all dissimilarity information has been shown to be NP-hard. Significant efforts have been invested in the relaxation of this constraint, using multi-valued decision diagrams, hoping to enhance cost propagation and pruning, thus speeding the search. Unfortunately, this has eluded us up to now.

Another improvement direction of this work comes from the observation that a solution of the *diabetes* Bayesian network can not be produced in reasonable time unless the problem structure is exploited via a tree decomposition.

The negative design approach requires further in silico tests, to attest that it does design molecules with improved binding or specificity. Further comparisons with existing approaches, like iCFN or OSPREY, would also be welcome.

Other useful CFN capabilities could be adapted to the bilevel resolution, such as the diversity requirement, and the enumeration of sequences (with or without diversity). Design might aim at disfavoring the negatives states, while maintaining them stable, so it would be useful to be able to set a maximum energy degradation threshold. Such a threshold could also be enforced on positive states, to ensure that the design remains energetically close to its optimum.

The developed bi-level algorithm should be able to tackle non-CPD instances, with soft cost functions between positive and negative problems. This has yet to be verified. Further developments could obviously tackle quantified optimization, with an arbitrary number of levels. Note that each quantifier adds to the complexity of the problem, and the generalization is far from straightforward.

The protein design developments presented in this thesis were performed in response to actual needs in design projects in the context of collaborations with protein engineering teams. They were implemented in a protein design software that relies on Rosetta for energy matrices computation and Toulbar2 for optimization. They were then used for the prediction of diverse libraries of sequences with a controlled number of mutations in a positive multi-state design approach. Experimental tests were then performed to select the most promising sequence among this library. Another starting collaboration project will make use of the bilevel optimization approach for the negative design of a new protein, that should be fully experimentally tested.

Appendices

Appendix A

French Summary

A.1 Introduction

Le Sudoku est un casse-tête combinatoire. Dans sa forme la plus courante, une grille de taille 9×9 doit être complétée avec des chiffres, en respectant des règles simples : chaque ligne, colonne et sous-grille de taille 3×3 doit contenir tous les chiffres de 1 à 9. Pourtant, étant donnée une grille préremplie, parcourir tous les remplissages possibles jusqu'à en trouver un qui satisfait les règles peut demander un temps plus grand que l'âge de l'univers, même si le fait de considérer une solution ne requiert qu'une microseconde. Le Sudoku est un très bon exemple de problème de *satisfaction de contraintes*, dans lequel des valeurs doivent être affectées à des variables, en respectant un ensemble de règles simplement exprimées.

Certains problèmes nécessitent d'exprimer des préférences ou des incertitudes. Ils peuvent alors s'appuyer sur une généralisation de la satisfaction de contraintes : le problème de *satisfaction de contraintes pondéré*. Les affectations de variables sont associées à une fonction de coût jointe, qui se décompose en une combinaison de plusieurs fonctions simples. Généralement, ces fonctions simples impliquent peu de variables, ou s'expriment succinctement dans un langage dédié. L'objectif est non seulement de trouver une solution qui satisfait toutes les contraintes, mais aussi de s'assurer que cette solution a un coût minimal. La satisfaction de contraintes pondérée permet d'exprimer une large palette de problèmes concrets, comme le design de protéines.

Les protéines sont des molécules complexes qui régissent le fonctionnement des cellules chez l'homme, les plantes, les microbes, etc. Elles sont constituées d'une succession de molécules simples appelées *acides aminés*. En solution, la plupart des protéines se replient en une forme 3D déterminée par les propriétés physico-chimiques de leurs acides aminés. Parce qu'elles assument une grande variété de fonctions et ont un grand potentiel applicatif en médecine, biocarburants, chimie verte, etc, de nouvelles séquences protéiques sont recherchées, aux propriétés et fonctions nouvelles ou améliorées. Comme la fonction d'une protéine est étroitement liée à sa structure, les méthodes de design computationnel visent à trouver une séquence qui se replie sur une structure cible, qui correspond aux propriétés et fonctionnalités

désirées. Une formulation générale de ce problème étant hors de portée, des hypothèses simplificatrices sont faites. Ces hypothèses permettent entre autres la modélisation du problème comme un réseau de fonctions de coût et donc sa résolution efficace.

De nouvelles protéines ont été conçues avec succès, mais cette approche de design mono-état reste peu fiable : les hypothèses simplificatrices sur le squelette et les acides aminés ne tiennent pas compte de la flexibilité des protéines et le critère optimisé n'est qu'une approximation de l'énergie réelle de la molécule, qui ne représente pas toujours sa fonction. C'est pourquoi la séquence prédite ne présente pas forcément les propriétés et fonctionnalités ciblées. Les méthodes computationnelles de design visent alors à prédire de petites librairies de séquences candidates les plus pertinentes possible, qui seront ensuite testées expérimentalement.

L'objectif principal de cette thèse a été le développement d'algorithmes pour améliorer la performance de la méthode de design computationnel de protéines, basée sur les réseaux de fonctions de coût. Ce manuscrit comprend quatre chapitres.

Le chapitre 1 introduit les concepts de base de la programmation par contraintes pondérée. Après un survol rapide de la programmation par contraintes et ses stratégies de résolution, les structures de valuation et les modèles graphiques sont présentés. Les réseaux de fonctions de coût sont des modèles graphiques avec une structure de valuation additive. Ils sont décrits en détail, ainsi que les algorithmes visant à résoudre efficacement les problèmes d'optimisation qui leur sont associés.

Le chapitre 2 présente les notions basiques sur les protéines, leurs structures et les mécanismes de repliement. Le cadre du design computationnel de protéines est fixé, dans sa formulation mono-état, avec la description des arguments d'entrée, de l'espace de recherche et du critère à optimiser. La modélisation sous la forme d'un réseau de fonctions de coût est donnée, ainsi que les méthodes de résolution existantes.

Le chapitre 3 vise à permettre une meilleure exploration de l'espace de recherche des séquences, en garantissant la diversité et la qualité des solutions pour la génération d'une librairie de séquences pertinentes. Une contrainte de diversité est développée, qui permet la production gloutonne de séquences, pour lesquelles on peut prouver qu'elles sont à la fois diverses et de basse énergie. Les expériences menées sur des instances de design montrent que les ensembles de séquences générés avec diversité ont une probabilité plus importante de fournir de bonnes séquences en pratique, par rapport aux solutions d'énergie minimale non-contraintes.

Le chapitre 4 montre que les algorithmes de recherche pour le design mono-état peuvent être généralisés aux designs multi-états positif et négatif,

permettant ainsi l'intégration d'information de flexibilité dans la modélisation. Le design multi-états positif demande en entrée plusieurs conformations du squelette de la protéine pour décrire la structure cible (au lieu d'une seule conformation). La structure ciblée est d'autant plus précise dans le design multi-états négatif, que des états désirés et non-désirés sont spécifiés.

Contributions de la thèse

Implémentation d'une contrainte de diversité Le contrôle de la diversité vis-à-vis de séquences d'intérêt est une contribution significative aux outils de design de protéines. En effet, les designs visent souvent à améliorer une caractéristique spécifique d'une protéine existante, en mutant sa séquence. Peu de positions doivent être mutées, pour atteindre la caractéristique ciblée, tout en conservant les autres propriétés de la molécule.

Une contrainte de diversité a été développée, qui permet l'intégration de cette information de séquence dans la modélisation en réseau de fonctions de coût. Cette contrainte permet également la génération incrémentale de librairies de bonnes séquences diverses. À notre connaissance, c'est la première fois qu'une méthode d'optimisation combinatoire fournit des garanties à la fois sur la diversité et la qualité des solutions de l'ensemble renvoyé. Ces travaux ont été publiés dans [Ruffini *et al.* 2019a, Ruffini *et al.* 2019b].

Étude de complexité du design multi-états On sait que le design mono-état est NP-complet. Le design multi-états peut être intrinsèquement plus dur que le design mono-état, mais ce supplément de complexité n'intervient qu'avec la prise en compte d'états négatifs. Le design multi-états positif est capable de capturer de la flexibilité ainsi que des états conformationnels distincts du squelette tout en restant "seulement" NP-complet. Le design négatif est nécessaire pour améliorer la spécificité ou minimiser l'énergie de liaison. Il est en revanche $\mathrm{NP}^{\mathrm{NP}}$-complet. La preuve de complexité a été publiée dans [Vucinic *et al.* 2020].

Optimisation bi-niveau pour le design multi-états négatif Le problème de design négatif $\mathrm{NP}^{\mathrm{NP}}$-complet peut être formulé comme une requête bi-niveau sur un réseau de fonctions de coût. Les mécanismes d'optimisation existants, comme la séparation-évaluation (branch-and-bound) et les cohérences locales, ont été adaptés à l'optimisation bi-niveau. Une évaluation plus approfondie sera nécessaire pour estimer l'efficacité de l'algorithme proposé et la qualité du design, mais les tests préliminaires semblent plutôt prometteurs.

A.2 Réseaux de fonctions de coût et optimisation

Le chapitre 1 introduit les concepts de base de la programmation par contraintes pondérée, qui est utilisée tout au long de la thèse. Ce paradigme permet de représenter succinctement une fonction sur un grand nombre de variables discrètes, exprimée comme une combinaison de fonctions simples, et de répondre à des requêtes sur cette fonction.

Un réseau de fonctions de coût (CFN) est un triplet $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ où :

- $\mathbf{X} = (X_1, \ldots, X_n)$ est un ensemble de n variables
- $\mathbf{D} = (\mathbf{D}_1, \ldots, \mathbf{D}_n)$ est un ensemble de domaines. Chaque variable X_i prend ses valeurs dans le domaine $\mathbf{D}_i$
- $\mathbf{C}$ est un ensemble de fonctions de coût

Pour un ensemble de variables $\mathbf{S} \subseteq \mathbf{X}$, $\mathbf{D}_\mathbf{S}$ désigne le produit Cartésien des domaines des variables de $\mathbf{S}$. Pour un n-uplet de valeurs $\mathbf{t}$, $\mathbf{t}[\mathbf{S}]$ désigne la projection de $\mathbf{t}$ sur $\mathbf{S}$. Une fonction $c_\mathbf{S} \in \mathbf{C}$, avec une portée $\mathbf{S} \subseteq \mathbf{X}$, est une function $c_\mathbf{S}$ qui associe un coût entier inférieur ou égal à $\top \in \mathbb{Z} \cup \{\infty\}$ à chaque n-uplet de $\mathbf{D}_\mathbf{S}$. $\top$ est associé aux n-uplets interdits. Un CFN est en Forme Normale si tout $c_\mathbf{S} \in \mathbf{W}, \mathbf{S} \neq \varnothing$ est non négative et $\mathbf{C}$ contient une fonction constante (de portée vide) $c_\varnothing$ de valeur arbitraire dans $\mathbb{Z}$.

Un CFN $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ définit une fonction de coût entière jointe $C_\mathcal{C}$:

$$\begin{array}{rccc} C_\mathcal{C} : & \mathbf{D}_\mathbf{X} & \longrightarrow & \{0, \cdots, \top\} \\ & \mathbf{t} & \longmapsto & \sum\limits_{c_\mathbf{S} \in \mathbf{C}}^{\top} c_\mathbf{S}(\mathbf{t}[\mathbf{S}]) \end{array}$$

où $a +^\top b = \min(\top, a + b)$ est l'addition bornée à $\top$. Dans le reste du document, nous supposons que tous les CFN sont sous forme normale (calculable en temps linéaire). Dans ce cas, $c_\varnothing$ est un minorant de la fonction jointe $C_\mathcal{C}$.

Une fonction $c_\mathbf{S}$ dont l'image est égale à $\{0, \top\}$ est une *contrainte*. Une affectation de $\mathbf{X}$ dont le coût est strictement inférieur à $\top$ est *faisable*. Le problème de satisfaction de contraintes pondérées (WCSP) consiste à trouver une affectation faisable de $\mathbf{X}$ minimisant $C_\mathcal{C}(\mathbf{X})$. Si $\top = 1$, le WCSP est le problème de satisfaction de contraintes (CSP). Cette observation permet de déduire que le problème de décision associé au WCSP est NP-complet.

L'une des techniques les plus efficaces pour résoudre le WCSP consiste à maintenir une cohérence locale souple (MCLS) pendant une recherche en Branch and Bound [Hurley *et al.* 2016, Cooper *et al.* 2010, Allouche *et al.* 2015]. Ces algorithmes explorent un arbre (généralement) binaire dans lequel un CFN est associé à chaque nœud. La racine est le réseau $\mathcal{C}$

à résoudre. Pour créer une branche, une variable non affectée X_i et une valeur v de son domaine sont sélectionnées. Sur la branche de gauche, X_i est affectée à v et à droite, v est supprimée des valeurs possibles pour X_i. Pour éviter d'explorer l'arbre entier, un minorant lb est calculé. Si lb est supérieur ou égal au coût $\top$, la branche est élaguée. Si une feuille est atteinte, elle décrit une solution (faisable) et le majorant $\top$ est mis à jour (une solution améliorante est cherchée). En pratique, cette combinaison comporte deux composants essentiels: la méthode de sélection de la variable de branchement, qui affecte en priorité les variables qui entraînent un élagage puissant et l'utilisation de $c_\varnothing$ comme minorant, après l'application de cohérences locales souples [Cooper *et al.* 2010].

Les cohérences locales souples (SLC) sont des propriétés qui peuvent être établies sur un CFN $\mathcal{C} = (\mathbf{X}, \mathbf{D}, \mathbf{C})$ pour le transformer en un CFN équivalent $\mathcal{C}' = (\mathbf{X}, \mathbf{D}', \mathcal{C}')$ qui satisfait la propriété SLC appliquée, qui a un minorant $c_\varnothing$ plus grand et des domaines éventuellement réduits : toute valeur pour laquelle on peut facilement établir qu'elle mènera à des affectations de coût $\top$ peut être supprimée sans risque. Il existe différentes SLC qui offrent des minorants de plus en plus précis : cohérences de nœuds, $\varnothing$-inverse, directionnelle, d'arc, existentielle, virtuelle et optimale [Schiex 2000, Cooper *et al.* 2010]. À l'exception des trois premières, elles se réduisent toutes à la cohérence d'arc [Rossi *et al.* 2006] lorsque $\top = 1$ (CSP). Dans le cas des champs de Markov (MRF), les cohérence locales peuvent être vues comme des versions incrémentales des algorithmes de *message passing* (comme TRWS [Kolmogorov 2006]), qui traitent en plus les informations déterministes (coût $\top$) comme la cohérence locale le ferait en programmation par contraintes. Cette dernière propriété est cruciale notamment dans le cadre d'une recherche en Branch and Bound : chaque fois que le majorant $\top$ est mis à jour parce qu'une nouvelle solution a été trouvée, la quantité d'information déterministe est plus importante, et les coupes plus nombreuses dans l'arbre de recherche. Comme les CFN ne font intervenir que des coûts entiers, les cohérences locales existantes peuvent être établies en temps polynomial en la taille des tables utilisées pour représenter les fonctions de coût. Cependant, si des représentations compactes des fonctions globales sont utilisées, établir les cohérences peut devenir NP-difficile.

A.3 Design computationnel de protéines

Le chapitre 2 présente le cadre du design computationnel de protéines (CPD). Le CPD vise à répondre au besoin de méthodes automatiques pour la conception de protéines nouvelles, aux fonctions et propriétés améliorées. Il est montré que ce problème peut être traité grâce aux stratégies d'optimisation

dans les réseaux de fonctions de coûts présentées dans le chapitre 1.

Une protéine est une chaîne de molécules simples appelées acides aminés. Les acides aminés sont composés d'un noyau commun fixe formé d'atomes de carbone, d'azote et d'oxygène, et d'une chaîne latérale variable et flexible, aux propriétés chimiques variées. La nature de la chaîne latérale définit le type d'acide aminé. Dans la nature, il existe 20 chaînes latérales qui définissent 20 acides aminés. Dans une protéine, les parties invariables des acides aminés sont liées entre elles en une chaîne pour former le *squelette* de la protéine. La protéine se *replie* en une structure 3D qui est déterminée par la séquence d'acides aminés. Une chaîne latérale peut posséder jusqu'à 4 axes de rotations par rapport au squelette. La structure 3D définie par le squelette et l'ensemble des positions des chaînes latérales est appelée *conformation*; elle détermine l'activité chimique et la fonction de la protéine.

Le problème de design computationnel de protéines à squelette fixe consiste, à partir d'un squelette 3D rigide, à déterminer la séquence d'acides aminés qui se repliera sur ce squelette. Pour résoudre ce problème, l'approche usuelle consiste à utiliser une fonction d'énergie qui estime l'énergie de la protéine en fonction de la nature et de l'orientation de toutes ses chaînes latérales. On peut alors identifier la nature et la conformation des acides aminés qui correspondent à l'énergie minimale (ou stabilité maximale). La fonction d'énergie étant supposée décomposable en termes binaires, et le squelette étant supposé rigide, l'énergie de la protéine peut être décrite au moyen d'un modèle graphique binaire avec une variable par position dans la chaîne protéique, et un domaine qui décrit l'ensemble des types de chaînes latérales possibles, avec chacun un ensemble discret d'orientations déterminées a priori (appelée bibliothèque de rotamères). Une affectation d'énergie minimale définit une séquence d'acides aminés qui peut être synthétisée et testée.

Cette approche du design computationnel de protéines basée sur l'optimisation combinatoire dans les réseaux de fonctions de coût s'appuie sur plusieurs simplifications. Le squelette cible est supposé être rigide, l'espace des conformations est discrétisé, et la fonction d'énergie est décomposée en une somme de termes unaires et binaires. Si ces hypothèses permettent l'expression d'un problème hors de portée, elles ont un certain nombre de limites. D'abord, les séquences prédites ont une basse énergie sur le squelette cible, mais rien ne garantit qu'il n'existe pas une meilleure structure (d'énergie encore plus faible) pour la séquence. En outre, seule l'énergie de la molécule est représentée, pour optimiser la stabilité, en dépit de la flexibilité de la protéine. Au total, le challenge majeur du CPD réside dans l'exploration de l'espace exponentiel des conformations et dans l'intégration des bonnes propriétés

des protéines qui régissent leur stabilité, repliement, fonctionnalité, liaison, affinité, spécificité, etc.

A.4 Diversité et qualité garanties des solutions

A.4.1 Introduction

La programmation par contraintes permet de modéliser efficacement des problèmes pour lesquels les solutions et non-solutions sont séparées par un ensemble de propriétés, ou contraintes, que les solutions doivent satisfaire. La satisfaction de contraintes pondérée représente des coûts et de l'incertitude, quand l'information disponible est plus graduelle. Les réseaux de fonctions de coût modélisent un critère et des contraintes de faisabilité comme une somme de fonctions de coût, où les coûts infinis capturent l'infaisabilité.

En général, la fonction jointe n'est pas une représentation parfaite du problème réel, et les requêtes d'optimisation ne permettent pas d'identifier une unique affectation satisfaisante. Par exemple, il peut exister des critères supplémentaires non représentés dans le modèle. Même quand un critère existe, il est fréquent que les solutions optimales ne correspondent pas aux meilleures solutions en pratique. Ce cas de figure peut par exemple être rencontré quand le modèle graphique a été appris à partir d'un jeu de données: à cause de l'échantillonnage fini, le modèle appris attribue souvent une probabilité sous-optimale aux solutions extraites de l'ensemble apprentissage/test ; ou lorsque le modèle graphique est une estimation d'un modèle difficile à représenter. Alors, produire un ensemble de solutions diverses de coût faible paraît être une bonne idée.

En design de protéines, les fonctions d'énergie décomposables ne permettent que d'estimer un modèle de mécanique quantique qui est hors de portée des supercalculateurs les plus puissants. Par ailleurs, ces fonctions sont uniquement centrées sur la stabilité de la molécule, au détriment de sa fonctionnalité. Cela rend le processus de conception de protéines peu fiable. Une approche classique consiste à produire plusieurs protéines qui devront ensuite être testées expérimentalement. Idéalement, cette bibliothèque est constituée de séquences diverses de faible énergie, l'idée étant que la diversité augmente les chances de trouver une protéine fonctionnelle.

Comme leurs applications sont importantes, les séquences protéiques peuvent faire l'objet de brevets : une nouvelle séquence devra satisfaire une contrainte de distance de Hamming par rapport aux séquences brevetées. En outre, les structures cibles du CPD sont souvent issues de protéines existantes, dont la structure a été résolue expérimentalement. Dans ce cas, une séquence native existe, dont les caractéristiques sont le résultat de milliards d'années d'évolution. Une contrainte de distance maximum vis-à-vis de cette séquence

native peut être introduite, pour que la séquence prédite hérite de certaines de ces bonnes propriétés naturelles.

Dans le chapitre 3, une méthode est proposée pour produire un ensemble de solutions de coût faible, qui satisfont une contrainte de diversité inter-solutions. Ces travaux ont été publiés dans [Ruffini *et al.* 2019a, Ruffini *et al.* 2019b].

A.4.2 Assurer la diversité et la qualité

Motivés par le problème de design de protéines, nous considérons des semi-métriques définies par la somme de dissimilarités pour chaque variable, données par une matrice symétrique positive à diagonale nulle D. Pour deux affectations $\mathbf{t_S}$ et $\mathbf{t'_S}$ d'un ensemble de variables $\mathbf{S}$, la dissimilarité $d(\mathbf{t_S}, \mathbf{t'_S}) = \sum_{X_i \in \mathbf{S}} D(\mathbf{t_S}[X_i], \mathbf{t'_S}[X_i])$ est bien une semi-métrique. La distance de Hamming est définie par $H(i,j) = \mathbb{1}(i \neq j)$. Notons qu'une matrice de similarité S pour les protéines peut être transformée en une matrice de dissimilarité comme suit : $D(i,j) = \frac{1}{2}(S(i,i) + S(j,j)) - S(i,j)$.

Definition 56. Pour un ensemble de solutions $\mathbf{Z}$, sa dissimilarité moyenne est est définie par $\bar{d}(\mathbf{Z}) = \frac{1}{|\mathbf{Z}|} \sum_{\mathbf{t} \neq \mathbf{t'} \in \mathbf{Z}} d(\mathbf{t}, \mathbf{t'})$ et sa dissimilarité minimale par $\check{d}(\mathbf{Z}) = \min_{\mathbf{t} \neq \mathbf{t'} \in \mathbf{Z}} d(\mathbf{t}, \mathbf{t'})$.

Definition 57. Soient $\mathbf{S}, \mathbf{S'} \subseteq \mathbf{X}, |\mathbf{S}| = |\mathbf{S'}|$. Soient D une matrice de dissimilarité et δ un minorant de diversité. La fonction de coût globale $\textsc{Dist}(\mathbf{S}, \mathbf{S'}, D, \delta)$ est égale à 0 si $sign(\delta).d(S, S') \geq \delta$ et k sinon.

Soient $N = (\mathbf{X}, \mathbf{W})$ un CFN, D une matrice de dissimilarité, M un entier et δ un seuil de diversité. Le problème $\textsc{DiverseSet}(N, D, M, \delta)$ est celui de produire un ensemble $\mathbf{Z}$ de M solutions de N tel que $\forall \mathbf{t} \neq \mathbf{t'} \in \mathbf{Z}$, $\textsc{Dist}(\mathbf{t}, \mathbf{t'}, D, \delta) = 0$ et $\sum_{\mathbf{t} \in \mathbf{Z}} J_N(\mathbf{t})$ soit minimale. Ce problème peut être très difficile à résoudre même sur de petites instances, aussi nous considérons le problème proche $\textsc{DiverseSeq}(N, D, M, \delta)$ qui consiste à trouver une liste $\mathbf{Z}$ de M solutions de N telle que pour tout $1 \leq i \leq M$, $\mathbf{Z}[i]$ est telle que $\forall j < i, \textsc{Dist}(\mathbf{Z}[i], \mathbf{Z}[j], D, \delta) = 0$ et de coût minimal.

Pour un CFN N à n variables, résoudre DiverseSet nécessite d'affecter simultanément nM variables. Ce problème peut être résolu en créant M copies de N avec des ensembles de variables $\mathbf{X}^1 \dots \mathbf{X}^M$ et en ajoutant $\frac{M.M-1}{2}$ contraintes $\textsc{Dist}(\mathbf{X}^i, \mathbf{X}^j, D, \delta)$ pour tous $1 \leq i < j \leq M$ (si $k < \infty$, toutes les occurrences de k doivent être remplacées par $M(k-1)+1$).

À la place, nous nous attaquons au problème glouton DiverseSeq en affectant itérativement n variables M fois, ce qui peut être exponentiellement plus rapide, vu la NP-difficulté du WCSP. Étant donné un ensemble de solutions $\mathbf{Z}$ déjà déterminé, définissons la fonction de coût globale $\textsc{Div}_{\min}(\mathbf{X}, \mathbf{Z}, D, \delta) = \bigoplus_{\mathbf{t} \in \mathbf{Z}} \textsc{Dist}(\mathbf{X}, \mathbf{t}, D, \delta)$. Le CFN $(\mathbf{X}, \mathbf{W} \cup$

$\{\text{DIV}_{\min}(\mathbf{X}, \mathbf{Z}, D, \delta)\}$) est résolu à plusieurs reprises, avec $\mathbf{Z} = \varnothing$ au début, puis en ajoutant la solution trouvée à $\mathbf{Z}$ itérativement jusqu'à atteindre $|\mathbf{Z}| = M$, ou jusqu'à ce qu'il n'existe plus de solution. Comme les CFN résolus sont de plus en plus contraints, le coût des solutions successives est croissant. Si des solutions $\mathbf{t}$ (brevets) sont données a priori, elles peuvent être prises en compte en ajoutant les contraintes correspondantes $\text{DIST}(\mathbf{X}, \mathbf{t}, D, \delta)$ avant la première résolution.

A.4.3 Représentation de la contrainte de diversité

Étant donnés un CSP et un ordre sur ses variables, une contrainte est définie par son ensemble fini de n-uplets autorisés. Cet ensemble définit un langage régulier qui peut être encodé par un automate [Pesant 2004] et AC peut être établie en temps linéaire en la taille de l'automate. Les contraintes $\text{DIV}_{\min}$ et DIST peuvent donc être encodées par un automate. Pour DIST, il est établi qu'un automate compact existe [Pesant 2004]. La NP-complétude de $\text{DIV}_{\min}$ signifie qu'il n'existe pas d'automate compact pour $\text{DIV}_{\min}$ (sauf si P=NP). Pour un CFN, les SLC transforment la contrainte DIST en fonctions qui font intervenir des coûts autres que 0 et k. Les SLC peuvent toujours être établies avec un automate *pondéré* [Lee & Leung 2012] qui définit un langage des n-uplets autorisés avec leur coût associé.

Un automate pondéré encode la contrainte $\text{DIST}(\mathbf{X}, \mathbf{t}, D, \delta)$ avec:

- $(\delta + 1).(n + 1)$ états s_i^d qui représentent le fait que les valeurs de $X_1 \dots X_i$ sont à distance d de $\mathbf{t}[X_1 \dots X_i]$ (ou $\geq \delta$ pour les états s_i^δ).
- pour les valeurs r de X_i, il y a une transition de coût 0 de s_i^d vers $s_{i+1}^{\min(d+D(r,t[i+1]),\delta)}$.
- l'état initial est s_0^0 et l'état acceptant est s_n^δ

Cet automate pondéré contient $O(n.(\delta+1).d)$ transitions. Une affectation $\mathbf{t}'$ de $\mathbf{X}$ est acceptée par cet automate ssi $d(\mathbf{t}', \mathbf{t}) \geq \delta$.

Les SLC peuvent être établies en temps polynomial en la taille de l'automate grâce à des algorithmes de flot minimal ou de programmation dynamique [Lee & Leung 2012]. Par ailleurs, comme dans le cas CSP [Bessière & Van Hentenryck 2003], les SLC peuvent être établies sur un automate pondéré représentant une fonction de coût en décomposant la fonction globale en une séquence de fonctions de coûts ternaires [Allouche *et al.* 2012]. Cet objectif peut être atteint par l'ajout de $n+1$ variables états $Q_i, 0 \leq i \leq n$ et n fonctions ternaires $w^A_{\{Q_i, X_{i+1}, Q_{i+1}\}}$ telles que $w^A_{\{Q_i, X_{i+1}, Q_{i+1}\}} = c$ ssi il existe une transition de l'état Q_i vers l'état Q_{i+1} avec la valeur X_{i+1} et le poids c dans l'automate. Les variables Q_0 et Q_n ont des domaines réduits contenant uniquement les états initiaux et terminaux respectivement.

Dans le cas CSP, établir AC sur la décomposition est équivalent à l'établir sur la contrainte globale originale. Ceci permet d'établir AC sur des fonctions de coût globales avec un automate compact sans nouvel algorithme. Pour les CFN cependant, établir une SLC sur la décomposition peut être plus faible que l'établir sur la fonction globale. Pour retrouver la propriété d'équivalence, il faut que l'ordre des variables utilisé pour construire l'automate soit compatible avec l'ordre utilisé pour la cohérence d'arc directionnelle [Allouche *et al.* 2012]. Hors, les contraintes Div_{min} et Dist sont insensibles à l'ordre des variables, et la condition d'ordonnancement de [Allouche *et al.* 2012] peut toujours être satisfaite. Étant donné la diversité des SLC définies sur les CFN, la décomposition semble être une approche intéressante pour encoder Dist et Div_{min}.

Dans le cas de la contrainte Dist, chaque variable état Q_i a $(\delta+1).(n+1)$ valeurs et la table de coûts de la fonction w^A est de taille $(\delta+1)^2.(n+1)^2.d_i$, où d est la taille du domaine de X_i. La contrainte Div_{min} nécessiterait quant à elle une table de taille $(\delta+1)^{2M}.(n+1)^2.d$. Pour accélérer l'établissement des SLC, nous exploitons les propriétés de Dist et D pour réduire cette complexité.

A.4.4 Compression de l'encodage de la contrainte de diversité

Dans la sous-section 3.3.5, nous montrons que l'encodage d'une contrainte Dist en une séquence de $n+1$ contraintes ternaires décrites par des tables de taille $(\delta+1)^2.(n+1)^2.d$ peut être réduit de plusieurs façons. Pour Dist, on sait que les états s_i^d ne peuvent être atteints qu'après exactement i transitions et sont propres à la variable Q_i: les domaines des variables $Q_i, 0 < i < n$ peuvent être réduits aux $\delta+1$ états s_i^d. De plus, nos semi-métriques sont définies par une somme croissante d'éléments positifs de D, donc chaque état s_i^d peut atteindre l'état acceptant s_n^δ ssi la dissimilarité maximale (notée md_i) qui peut être atteinte à partir de la variable i jusqu'à la variable n est supérieure à $\delta - d$. Toutes ces dissimilarités maximales peuvent être calculées en amont en un parcours des variables de $\mathbf{X}$, puisque $md_1 = 0; md_i = md_{i-1} + max_{r,s \in D_i \times D_{i+1}} D(r,s)$. Dans le cas de la distance de Hamming, la distance ne peut augmenter que de 1 à chaque variable, et cette valeur est simplement $n-i$. Un raisonnement symétrique peut être appliqué à partir de l'état initial s_0^0. Ces simplifications réduisent la taille des tables des fonctions de coût à $O((\delta+1)^2.d)$. Même si de tels états non atteignables seraient systématiquement éliminés par AC dans le cas booléen, un grand nombre d'EPT pourraient être appliquées dans le cas CFN.

En outre, les fonctions ternaires de la décomposition de l'automate peuvent être elles-même décomposées en fonctions binaires, grâce aux représen-

tations duale ou cachée [Bacchus & Van Beek 1998]. Ces représentations nécessitent la description de $O(\delta.d + \#D^2\delta^2)$ and $O(\delta.d + \#D\delta^2)$ n-uplets respectivement ($O(\delta.d + \delta^2)$ dans le cas Hamming) au lieu des $O(d.\delta^2)$ n-uplets des fonctions ternaires.

A.4.5 DiverseSeq glouton

Pour résoudre le problème DiverseSeq(N, D, M, δ), on peut utiliser l'approche gloutonne suivante : nous résolvons le CFN N via un Branch and Bound. Si une solution $\mathbf{t}$ est trouvée, elle est ajoutée à la séquence de solutions en cours $\mathbf{Z}$. Si M solutions ont été produites, nous nous arrêtons. Sinon, une contrainte Dist$(\mathbf{X}, \mathbf{t}, D, \delta)$ est ajoutée au problème précédent. Nous effectuons une boucle et le résolvons à nouveau. Si aucune solution n'existe, la séquence $\mathbf{Z}$ ne peut pas être étendue à la longueur M et le problème n'a pas de solution (mais une séquence plus courte a été produite). Ce schéma peut être amélioré de trois manières qui peuvent chacune offrir des gains exponentiels en temps.

1) les problèmes étant de plus en plus contraints, le filtrage appliqué à l'itération $i-1$ reste valide pour les itérations suivantes. Au lieu de redémarrer à partir d'un problème $N = (\mathbf{X}, \mathbf{W} \cup_{1 \leq j < i} \{\ Dist(\mathbf{X}, \mathbf{Z}[j], D, \delta\})$, nous réutilisons le problème résolu à l'itération $i-1$ après filtrage pour ajouter la contrainte Dist$(\mathbf{X}, \mathbf{Z}[i-1], D, \delta)$ et filtrer de nouveau. De plus, les paramètres des heuristiques de choix de variables appris à l'itération $i-1$ sont réutilisés à l'itération i.

2) les problèmes étant de plus en plus contraints, le coût optimal oc^i obtenu à l'itération i ne peut avoir un coût inférieur au coût oc^{i-1}. Lorsque de grands plateaux sont présents, cela permet d'arrêter la recherche dès qu'une solution de coût oc^{i-1} est atteinte, évitant ainsi une preuve d'optimalité.

3) même s'il n'y a pas de plateau dans le paysage énergétique, il peut y avoir de grandes régions avec des variations d'énergie similaires. Dans ce cas, la différence d'énergie entre oc^{i-1} et oc^i restera similaire pour plusieurs itérations. Soit $\Delta_i^h = \max_{\max(2,ih) \leq j < i}(oc^j - oc^{j-1})$, la variation maximale dans les dernières h itérations ($h = 5$ semble raisonnable). A l'itération i, nous résolvons le problème avec un majorant temporaire $k' = \min(k, oc_{i-1} + 2, \Delta_i^h)$. Si $k' < k$, cela entraînera un déterminisme accru et un élagage éventuellement plus efficace. Si aucune solution n'est trouvée, le problème est résolu à nouveau avec un majorant initial égal à k. Le majorant est donc prédit.

A.4.6 Expérimentations

Nous avons implémenté l'approche itérative décrite ci-dessus au moyen d'une décomposition directe (ternaire) ainsi que ses versions cachée et duale

pour DIV_{min} décomposée en une conjonction de fonctions DIST au-dessus du solveur de CFN C++ open source toulbar2 [Hurley *et al.* 2016]. Toulbar2 implémente divers algorithmes de pré-traitement dédiés à l'optimisation exacte. Nous avons dû désactiver au niveau du nœud racine tous les algorithmes de pré-traitement ne préservant pas les solutions sous-optimales: élimination des variables, substituabilité et fusion de variables. Nous avons choisi d'appliquer la cohérence d'arc virtuelle au premier problème résolu. Le coût de calcul de VAC, en $O(\frac{ed^2k}{\varepsilon})$, où ε est la plus petite énergie représentable en décimal dans la résolution utilisée, est important. Mais il est amorti sur les M résolutions. Lors de la recherche arborescente, la cohérence locale EDAC usuelle a été utilisée.

Dans les sections 3.6 et 3.7, nous montrons que des séquences de solutions qui satisfont aux garanties de diversité peuvent être obtenues sur de grands réseaux Bayésiens ainsi que sur des grandes instances de CPD. Cette garantie est obtenue sur des problèmes denses de fonctions non sous-modulaires tout en garantissant que chaque nouvelle solution produite est la meilleure compte tenu des solutions précédemment identifiées.

Nous montrons également que la séquence de solutions diverses produite par notre algorithme peut être filtrée et chaque solution efficacement identifiée comme étant un δ-mode (ou non). Dans les quelques problèmes où nous avons pu comparer notre algorithme aux résultats de [Chen *et al.* 2018], notre approche semble tout aussi efficace en temps de calcul, tout en fournissant des informations plus détaillées sur la forme du paysage énergétique autour de l'optimum.

Nous avons également testé notre approche sur des problèmes de design de protéines. Pour cela, nous avons utilisé deux ensembles de squelettes de protéines issus de [Simoncini *et al.* 2015], où nous avons sélectionné les 20 instances qui étaient résolues le plus rapidement, et de [Ollikainen & Kortemme 2013], où nous avons sélectionné les 17 instances pour lesquelles le GMEC était identifié par Toulbar2 en moins de 24h CPU. Un redesign complet a été fait sur ces squelettes, avec la librairie de rotamères Dunbrack [Shapovalov & Dunbrack Jr 2011] et la fonction de score Rosetta ref2015 [Park *et al.* 2016]. Les réseaux obtenus ont entre 44 et 87 variables avec des tailles de domaines maximum entre 294 et 446 rotamères.

Sur ces instances, nous observons que la qualité des librairies de séquences est améliorée grâce aux conditions de diversité. De plus, produire des solutions sous-optimales avec diversité permet aussi d'améliorer la qualité des librairies de séquence, par rapport à l'énumération simple de séquences de basse énergie.

A.4.7 Conclusion

Produire une séquence de solutions diverses est une exigence habituelle dans l'optimisation d'une fonction approximative ou apprise. Dans le chapitre 3, nous montrons qu'il est possible de produire des séquences de solutions avec garanties de diversité sur de grands réseaux bayésiens et sur de grandes instances de design de protéines, en utilisant une approche incrémentale d'optimisation sur les réseaux de fonctions de coût, avec des contraintes de diversité représentées par des automates pondérés. La garantie de diversité est obtenue sur des problèmes avec des fonctions non sous-modulaires, et il est aussi garanti que la solution est la meilleure, compte tenu des solutions identifiées précédemment.

Sur de vraies instances de design de protéines, nous observons que l'exigence de diversité permet d'améliorer la qualité des librairies de séquences prédites. De plus, la qualité des séquences sous-optimales diverses est aussi meilleure que la simple énumération des séquences d'énergie minimale. Dans le cadre de l'optimisation d'une fonction apprise ou approchée, l'exigence d'optimalité de la solution peut paraître exagérée. Notre résolution sous-optimale est pertinente en pratique, étant donné que même des approches coûteuses avec convergence asymptotique comme le recuit simulé peuvent échouer avec une erreur non bornée.

Il y a deux suites possibles à ces travaux. La première direction est mise en évidence par le problème *diabetes* pour lequel le premier élément de DiverseSeq ne peut être produit rapidement. Ce problème a une largeur d'arbre inférieure à 4 et peut être résolu en moins de 40 secondes si l'on exploite une décomposition arborescente (min-fill) pendant la recherche. Il faudrait donc montrer que les fonctions décomposées que nous utilisons pour encoder Dist peuvent être agencées de manière à ce que la largeur de l'arbre puisse être préservée ou peu augmentée. La seconde direction consisterait à identifier une version relâchée de la contrainte NP-difficile Div_{min} capable de produire de bons minorants par filtrage. Malgré des efforts non négligeables dans cette direction en utilisant des diagrammes de décision multi-valués, nous n'avons pas abouti à des résultats convaincants.

A.5 Design multi-états positif et négatif

A.5.1 Introduction

Le design computationnel de protéines vise à répondre au besoin de méthodes automatiques pour la prédiction de nouvelles séquences protéiques qui se replient en une structure désirée, aux propriétés et fonctions améliorées.

Dans le cadre du design mono-état, la structure ciblée est décrite par une unique conformation, et une séquence d'énergie minimale sur le squelette correspondant est recherchée. Ce problème est hors de portée, et des hypothèses simplificatrices sont faites. Le squelette de la protéine est supposé être rigide, et les conformations des chaines latérales des acides aminés sont discrétisées. Ces conformations discrètes sont appelées les *rotamères*. Ces simplifications permettent la modélisation de la fonction d'énergie par un réseau de fonctions de coût. Des algorithmes efficaces basés sur un branch and bound et des cohérences locales permettent alors de déterminer la conformation d'énergie minimale (GMEC).

Malheureusement, ces hypothèses de rigidité et de discrétisation ne permettent pas de représenter la flexibilité de la protéine, alors que les protéines n'adoptent pas qu'une unique position : elles évoluent plutôt dans un ensemble de conformations de basse énergie. Pour intégrer de la flexibilité dans le modèle de design, le design multi-états permet la donnée d'entrée de plusieurs états conformationnels pour décrire la structure cible. Cela permet le design de protéines flexibles, qui adoptent plusieurs positions proches de leur conformation "idéale", ou qui effectuent des changements conformationnels plus larges.

Le design multi-états *négatif* permet la donnée à la fois de structures *désirées* et *non-désirées* en entrée. La séquence recherchée doit stabiliser les états positifs (désirés) tout en déstabilisant les conformations négatives (non-désirées). On peut ainsi concevoir des protéines avec une affinité améliorée : l'état désiré correspond à la structure liée et l'état non-désiré à la structure non-liée. La spécificité peut aussi être modélisée : les complexes protéine liée aux ligands désirés et non-désirés sont les états positifs et négatifs respectivement.

A.5.2 Design de protéines muti-états

Dans le cadre du design multi-états, avec squelette rigide et rotamères discrets, la structure d'entrée est spécifiée par un ensemble d'états positifs et négatifs $\mathbf{B} = \mathbf{B}^+ \cup \mathbf{B}^-$. Les états positifs sont ciblés et les états négatifs sont à éviter. Sans perte de généralité, on peut supposer que tous les états ont le même nombre de positions n. Pour chaque position $1 \leqslant i \leqslant n$, l'ensemble $\mathcal{S}_i$ représente les acides aminés autorisés à la position i. Pour chaque type d'acide aminé $s \in \mathcal{S}_i$, et chaque état $B^j \in \mathbf{B}$, l'ensemble $\mathcal{R}_i^j(s)$ représente l'ensemble des rotamères de s autorisés sur le squelette B^j. L'ensemble des rotamères de B^j autorisé à la position i est noté $\mathcal{R}_i^j = \bigcup_{s \in \mathcal{S}_i} \mathcal{R}_i^j$.

Soit E^j la fonction qui a une séquence de rotamères $\mathbf{r} \in \prod_{i=1}^n$ associe l'énergie de la conformation correspondante sur le squelette B^j. On suppose que E^j est décomposable en termes binaires.

Étant donnée une séquence d'acides aminés $\mathbf{s} \in \prod_{i=1}^{n} \mathcal{S}_i$, la pertinence de $\mathbf{s}$ sur un ensemble d'états est définie comme l'énergie moyenne des meilleures conformations des états. La minimisation de cette énergie moyenne devrait permettre la prédiction d'une séquence convenable pour chacun des états. Comme le nombre d'états est fixé, minimiser la moyenne revient à minimiser la somme des énergies des états. Ainsi, le design multi-états positif, avec conformations données $\mathbf{B} = \mathbf{B}^+$, revient à trouver la séquence s^* de pertinence optimale :

$$\mathbf{s}^* = \arg \min_{\mathbf{s} \in \prod_i \mathcal{S}_i} \left(\sum_{B^j \in \mathbf{B}^+} \min_{\mathbf{r} \in \prod_i \mathcal{R}_i^j(\mathbf{s}[i])} \mathbf{E}^j(\mathbf{r}) \right)$$

Étant donnés deux ensembles d'états conformationels désirés et non-désirés $\mathbf{B} = \mathbf{B}^+ \cup \mathbf{B}^-$, la fonction objectif à minimiser est définie comme la différence de pertinence entre les états positifs $\mathbf{B}^+$ et les états négatifs $\mathbf{B}^-$. Ainsi, le design multi-états négatif correspond à la prédiction d'une séquence $\mathbf{s}^*$ telle que :

$$\mathbf{s}^* = \arg \min_{\mathbf{s} \in \prod_i \mathcal{S}_i} \left[\left(\sum_{B^j \in \mathbf{B}^+} \min_{\mathbf{r} \in \prod_i \mathcal{R}_i^j(\mathbf{s}[i])} \mathbf{E}^j(\mathbf{r}) \right) - \left(\sum_{B^j \in \mathbf{B}^-} \min_{\mathbf{r} \in \prod_i \mathcal{R}_i^j(\mathbf{s}[i])} \mathbf{E}^j(\mathbf{r}) \right) \right]$$

Complexité On sait que le problème de design mono-état est NP-complet [Pierce & Winfree 2002]. On peut montrer que le problème de décision associé au design multi-états positif (avec conformations discrètes et squelette rigide) est aussi NP-complet. En revanche, le problème de décision associé au design multi-états négatif est intrinsèquement plus difficile : il est $\Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}}$-complet.

A.5.3 Design multi-états positif

Le design computationnel mono-état peut être modélisé comme un problème de satisfaction de contraintes pondéré (WCSP), représenté par un réseau de fonctions de coût. Les designs mono- et multi-état(s) correspondent à la même classe de complexité. Intuitivement, cela suggère que le design multi-états peut aussi être modélisé par un WCSP.

Le design multi-états ne peut pas être résolu par la prédiction de la séquence optimale de chaque état et en sommant leurs énergies, parce que les séquences optimales pour chaque état pris séparément peuvent être différentes. Toutefois, les réseaux de fonctions de coût permettent d'encoder des contraintes dures, qui interdisent certaines affectations partielles, en utilisant le coût $\top$. Ces contraintes peuvent être utilisées dans le cadre du design multi-états positif, pour s'assurer que pour chaque position, les conformations choisies dans les différents états correspondent au même type

d'acide aminé.

Plus précisément, pour chaque état $B^j \in \mathbf{B} = \mathbf{B}^+$, on construit le réseau représentant la fonction d'énergie du squelette B^j. Le réseau multi-état est défini comme suit :

- Ses variables correspondent à l'union de toutes les variables des réseaux de chaque état
- Ses fonctions sont l'union de toutes les fonctions d'énergie des réseaux, auxquelles s'ajoutent des fonctions encodant les contraintes séquence : pour chaque position et chaque paire d'états, on doit s'assurer que les rotamères dans les deux états correspondent au même type d'acide aminé.

Par définition, la fonction de ce réseau correspond à la somme des énergies des états, et une solution valide est un ensemble de conformations des états correspondant à une même séquence. Le problème peut être résolu avec les approches d'optimisation combinatoire usuelles, comme celles utilisées dans le logiciel Toulbar2.

Cette approche a été appliquée à 30 structures issues de la PDB déterminées par RMN ou rayons X, redesignées entièrement. Pour comparer la qualité des séquences prédites par les approches mono- et multi-état(s), les valeurs de recouvrement et de similarité de séquence (NSR et NSSR) ont été utilisées. Le design multi-états permet une amélioration de 15% et 8% sur les structures RMN et rayons X respectivement.

Une analyse du temps de calcul CPU permet d'observer que le temps de calcul croît exponentiellement avec la taille de l'instance à résoudre, ce qui est cohérent étant donnée la NP-difficulté des problèmes. En revanche, étant donnée leur taille plus importante, les problèmes multi-états paraissent plus faciles à résoudre que leurs formulations mono-état. Globalement, ces expériences montrent qu'un redesign mono- et multi-état(s) complet peut être effectué en temps raisonnable sur des protéines de taille < 100 acides aminés.

A.5.4 Design multi-états négatif

Dans le cadre du design multi-états négatif, la donnée d'entrée correspond à deux ensembles de squelettes désirés et non-désirés : $\mathbf{B} = \mathbf{B}^+ \cup \mathbf{B}^-$. À chaque squelette $B^j \in \mathbf{B}$ est associée une fonction d'énergie $\mathbf{E}^j$ décomposable en termes binaires. Comme mentionné précédemment, il s'agit d'identifier la séquence $\mathbf{s}^*$ telle que :

$$\mathbf{s}^* = \arg\min_{\mathbf{s} \in \prod_i \mathcal{S}_i} \left[\left(\sum_{B^j \in \mathbf{B}^+} \min_{\mathbf{r} \in \prod_i \mathcal{R}_i^j(\mathbf{s}[i])} \mathbf{E}^j(\mathbf{r}) \right) - \left(\sum_{B^j \in \mathbf{B}^-} \min_{\mathbf{r} \in \prod_i \mathcal{R}_i^j(\mathbf{s}[i])} \mathbf{E}^j(\mathbf{r}) \right) \right]$$

Le design multi-états positif s'exprime comme un problème de satisfaction de contraintes pondéré, en combinant les réseaux des états et en ajoutant des contraintes de séquence. De cette façon, on peut construire deux réseaux $\mathcal{C}^+$ et $\mathcal{C}^-$ qui représentent la somme des fonctions d'énergie des états positifs et négatifs respectivement : si on note $C_{\mathcal{C}^+}$ et $C_{\mathcal{C}^-}$ les fonctions jointes de $\mathcal{C}^+$ et $\mathcal{C}^-$ respectivement, on a :

$$C_{\mathcal{C}^+} = \mathbf{E}^+ = \sum_{B^j \in \mathbf{B}^+} \mathbf{E}^j \qquad C_{\mathcal{C}^-} = \mathbf{E}^- = \sum_{B^j \in \mathbf{B}^-} \mathbf{E}^j$$

On peut alors reformuler le problème comme suit :

$$\mathbf{s}^* = \underset{\mathbf{s} \in \prod_i \mathcal{S}_i}{\arg\min} \left[\min_{\mathbf{r}^+(\mathbf{s}) \in \mathcal{R}^+(\mathbf{s})} \mathbf{E}^+(\mathbf{r}^+(\mathbf{s})) - \min_{\mathbf{r}^-(\mathbf{s}) \ \in \mathcal{R}^-(\mathbf{s})} \mathbf{E}^-(\mathbf{r}^-(\mathbf{s})) \right]$$

où, pour une séquence $\mathbf{s} \in \prod_i \mathcal{S}_i$, l'ensemble $\mathcal{R}^+(\mathbf{s})$ (resp. $\mathcal{R}^-(\mathbf{s})$) est celui des conformations possibles de la séquence $\mathbf{s}$ sur tous les états positifs (resp. négatifs) :

$$\mathcal{R}^+(\mathbf{s}) = \prod_{B^j \in \mathbf{B}^+, i \in [\![1,n]\!]} \mathcal{R}_i^j(\mathbf{s}[i]) \qquad \mathcal{R}^-(\mathbf{s}) = \prod_{B^j \in \mathbf{B}^-, i \in [\![1,n]\!]} \mathcal{R}_i^j(\mathbf{s}[i])$$

Jeu à 2 joueurs Le problème de design négatif peut être vu comme un jeu à deux joueurs. Le joueur 1 doit trouver une séquence et des conformations correspondantes sur les squelettes positifs. Le joueur 2 doit alors trouver des conformations sur les squelettes négatifs, qui correspondent à la séquence imposée par le joueur 1. Le but du joueur 1 est de minimiser l'énergie de ses conformations, tout en maximisant celle de son adversaire. Le but du joueur 2 est d'avoir des conformations de basse énergie.

La solution du design négatif correspond à une partie au cours de laquelle les deux joueurs ont joué parfaitement.

Expression de la faisabilité Étant donnée une séquence $\mathbf{s}$, une conformation $\mathbf{r}$ est *faisable* lorsque ses rotamères correspondent aux acides aminés de la séquence. Dans les réseaux de fonctions de coût, la faisabilité peut être exprimée par des fonctions de coût qui prennent les valeurs 0 ou $\top$. Ces fonctions encodent alors une contrainte. Soient SS la fonction qui encode la contrainte de séquence, et $\mathcal{R}^+ = \prod_{B^j \in \mathbf{B}^+, i \in [\![1,n]\!]} \mathcal{R}_i^j$ (resp. $\mathcal{R}^- = \prod_{B^j \in \mathbf{B}^-, i \in [\![1,n]\!]} \mathcal{R}_i^j$) l'ensemble de toutes les conformations des états

positifs (resp. négatifs). On peut alors écrire :

$$\mathbf{s}^* = \arg\min_{\mathbf{s}\in\mathcal{S}} \min_{\mathbf{r}^+\in\mathcal{R}^+} \mathbf{E}^+(\mathbf{r}^+) + SS(\mathbf{s},\mathbf{r}^+) - \min_{\mathbf{r}^-\in\mathcal{R}^-} \mathbf{E}^-(\mathbf{r}^-) + SS(\mathbf{s},\mathbf{r}^-)$$

Notons que quelle que soit la séquence, il existe toujours une conformation négative $\mathbf{r}^-(\mathbf{s})$ telle que $\mathbf{E}^-(\mathbf{r}^-(\mathbf{s})) < \top$. Ainsi, une conformation "faisable" sera toujours préférée dans la minimisation de l'énergie sur les états négatifs. Ainsi, la minimisation sur $\mathbf{r}^-$ ne peut jamais produire de coût égal à $-\top$.

Optimisation biniveau Pour une séquence s et des conformations $\mathbf{r}^j$ des backbones positifs et négatifs, la valeur de la fonction objectif est alors donnée par :

$$\mathbf{E}(\mathbf{s},\mathbf{r}^1,\ldots,\mathbf{r}^l) = \sum_{\mathbf{B}^j\in\mathbf{B}^+} \mathbf{E}^j(\mathbf{r}^j) + SS(\mathbf{s},\mathbf{r}^j) + \sum_{\mathbf{B}^j\in\mathbf{B}^-} -\mathbf{E}^j(\mathbf{r}^j) + SS(\mathbf{s},\mathbf{r}^j)$$

On peut ainsi représenter la fonction objectif finale en combinant les réseaux des états positifs et les opposés des réseaux des états négatifs. On ajoute ensuite des variables séquence qui représentent le type d'acide aminé à chaque position et des fonctions encodant les contraintes de séquence. Sur le réseau résultant, on peut appliquer les cohérences locales pour avoir une fonction constante la plus grande possible, qui est un minorant de la fonction de coût globale. Comme on cherche une séquence qui *minimise* la fonction objectif, on peut utiliser ce minorant pour couper dans l'espace des séquences.

Une fois que toute la séquence est affectée, on peut explorer les espaces des conformations positives et négatives, avec les mécanismes usuels de minimisation.

Tests préliminaires La stratégie d'optimisation biniveau a été intégrée au solveur Toulbar2, puis testée.

Dans un premier temps, nous avons comparé notre méthode de résolution avec celle proposée par [Karimi & Shen 2018]. Nous avons utilisé un état positif et un état négatif dans 3 instances différentes mises à disposition sur le site de iCFN. Ces instances correspondent au redesign négatif d'un récepteur des cellules T, avec les positions 26,28 et 100 mutées une par une. Les instances iCFN ont été converties dans le format CFN de Toulbar2. Notre approche est 5 à 10 fois plus rapide qu'iCFN sur ces instances.

Ensuite, nous avons appliqué notre méthode à la minimisation d'énergie de liaison d'un complexe protéique : le domaine N-terminal XRCC1 et l'ADN polymérase beta. L'état positif correspond à l'état lié, et l'état négatif à l'état non-lié. Nous avons incrémentalement muté les N_{mut} premiers résidus

parmi les positions 391, 409, 411, 422, 424. Les résidus à moins de 3 Å puis de 6 Å sont flexibles et les autres fixes.

Les instances les plus petites sont résolues instantanément, et le temps de calcul augmente exponentiellement avec la taille de l'instance, ce qui est cohérent avec la NP^{NP}-difficulté du problème. Toutefois, même les plus grandes instances, qui correspondent à l'exploration biniveau de $9 \cdot 10^{13}$ séquences et $4.8 \cdot 10^{117}$ conformations, sont résolues en moins de 12 minutes.

A.5.5 Conclusion

Dans le chapitre 4, nous montrons que le design multi-états positif n'est pas plus dur que le problème mono-état, qui est NP-difficile. Il permet en outre de capturer la flexibilité des protéines, puisqu'il stabilise un ensemble d'états du squelette, qui représentent des petits mouvements autour d'une optimisation optimale, ou de plus larges changements conformationnels. Le design multi-états positif peut être modélisé par un problème d'optimisation dans un réseau de fonctions de coût, et résolu grâce à des algorithmes efficaces dédiés. Les expérimentations montrent que cette méthode est plus performante que l'algorithme récent iCFN, et que le design multi-états améliore clairement la qualité des séquences prédites. En outre, le design multi-états positif modélisé par réseau de fonctions de coût bénéficie des aptitudes des solveurs tels que Toulbar2, comme l'énumération exhaustive de séquences qui sont à distance fixée de l'optimum, ou du contrôle de diversité et la génération incrémentale de librairies de séquences diverses.

Le design multi-états négatif est intrinsèquement plus difficile que les designs mono-état et multi-états positif. En revanche, les états négatifs sont cruciaux quand il s'agit de générer de la spécificité. Nous proposons une méthode d'optimisation biniveau basée sur les réseaux de fonctions de coût pour résoudre ce problème, qui s'appuie sur les mécanismes de cohérences locales existants pour améliorer les majorants/minorants au cours de la recherche. Les tests préliminaires suggèrent que cette approche peut potentiellement surpasser les méthodes de résolutions exactes du design négatif, comme iCFN.

Ce potentiel doit être confirmé, et des tests computationnels et expérimentaux doivent être effectués pour estimer la qualité des prédictions du design négatif. Par ailleurs, les aptitudes des réseaux de fonctions de coût comme l'énumération de séquence et la génération de solutions diverses doivent être adaptées à l'optimisation biniveau. Une autre amélioration utile serait la possibilité de limiter la dégradation de l'énergie des états positifs et négatifs. Cela permettrait par exemple de minimiser l'énergie de liaison, tout en maintenant la stabilité des états positifs (liés) et négatifs (non-liés).

A.6 Conclusion

Cette thèse a contribué au développement d'algorithmes pour le problème de satisfaction de contraintes pondéré, principalement appliqué à la minimisation d'une fonction d'énergie pour le design computationnel de protéines.

La fonction optimisée n'est jamais une représentation parfaite des problèmes concrets qui sont souvent hors de portée. Une contrainte de diversité a donc été développée, ainsi qu'une méthode efficace de génération de librairies de solutions, avec des garanties à la fois de diversité et de qualité. Cette contrainte peut être utilisée pour modéliser de la connaissance basée sur des solutions d'intérêt. Cette fonctionnalité est particulièrement utile pour le design de protéines, pour lequel des séquences brevetées ou natives existent.

Nous avons montré que la génération gloutonne de solutions diverses constistue une amélioration efficace de l'ensemble exact de solutions diverses de coût minimal, avec peu de dégradation en terme de coût. Elle est capable de calculer des solutions diverses sur des réseaux Bayésiens de grande taille, et sur des grandes instances de design de protéines en un temps raisonnable. Les séquences protéiques diverses ont plus de chance de présenter une séquence de bonne qualité "en pratique", par rapport à l'énumération simple des séquences de basse énergie (sans diversité). Cette observation reste vraie quand la résolution exacte du critère est relâchée, et qu'un écart contrôlé à l'optimum est toléré. Notre méthode permet d'explorer efficacement l'espace de recherche et de fournir une librairie de séquences potentielles diverses à tester expérimentalement.

Plusieurs simplifications sont faites pour le design computationnel de protéines mono-état, qui visent à rendre le problème raisonnable, et à permettre sa modélisation par un réseau de fonctions de coût. Ces hypothèses de discrétisation écartent la flexibilité des protéines. Une façon de ré-intégrer l'information de flexibilité est de décrire la structure cible par un ensemble de conformations du squelette, ou états. Cette thèse affirme que le problème de décision associé à cette approche positive du design multi-états reste NP-complet, et n'est pas plus difficile que le problème mono-état, pour lequel des méthodes de résolution efficaces existent. La modélisation du problème de design multi-états positif est donnée, et des résultats in silico sont mentionnés.

Le design négatif est un cran plus haut dans la hiérarchie polynomiale, mais il est crucial pour le design de liaison protéine-ligand ou de spécificité. Les conformations du squelette ciblé sont réparties en deux catégories : les états désirés, qui doivent être stabilisés, et les états non-désirés, qui

doivent être évités. La fonction de pertinence correspondante peut toujours être représentée par un réseau de fonctions de coût, et bénéficie des transformations préservant l'équivalence qui appliquent les cohérences locales. En revanche, la requête devient bi-niveau. Les algorithmes de résolution par séparation-évaluation ont donc été adaptés pour résoudre l'inévitable problème d'optimisation bi-niveau.

Perspectives

En ce qui concerne les travaux portant sur la diversité, nous avons montré que la propagation de la contrainte $\textsc{Div}_{\min}$ qui rassemble toutes les informations de dissimilarité est NP-complète. Des efforts significatifs ont été investis dans la relaxation de cette contrainte, basés sur les diagrammes de décision multi-valués, pour espérer améliorer les propagations et les coupes dans l'arbre de recherche. Malheureusement, ces efforts sont pour l'instant restés vains.

Une autre piste d'amélioration de ces travaux est suggérée par le fait qu'une solution optimale du réseau Bayésien *diabetes* ne peut être atteinte en temps raisonnable à moins d'exploiter la structure du réseau via une décomposition arborescente.

L'approche proposée pour résoudre le problème de design négatif doit être plus amplement testée in silico, pour s'assurer qu'elle permet bien la prédiction de séquence à la liaison ou spécificité améliorée. La comparaison avec des approches existantes, comme iCFN ou OSPREY pourrait également être approfondie.

Plusieurs aptitudes des réseaux de fonctions de coût pourraient aussi être adaptées à l'optimisation bi-niveau, comme le contrôle de la diversité ou l'énumération de séquences (avec et sans diversité). En outre, une instance de design négatif pourrait viser à défavoriser un état tout en maintenant sa stabilité. Il serait alors utile de pouvoir définir un seuil de dégradation maximal sur les états négatifs. Un tel seuil pourrait aussi être appliqué aux états positifs, pour garantir que le design reste proche de l'optimum en terme d'énergie.

L'algorithme bi-niveau développé ici devrait être capable de résoudre des problèmes concrets non-issus d'instances de design de protéines, avec des fonctions de coût souples (avec des coûts autres que 0 et $\top$) entre les problèmes positif et négatif. Cela n'a pas été proprement vérifié. Les développements futurs pourraient toucher à l'optimisation quantifiée, avec un nombre arbitraire de niveaux. Il faut cependant noter que chaque niveau de quantification ajoute à la complexité du problème, et que cette généralisation est loin d'être directe.

Les améliorations à la résolution du problème de design de protéines présentées dans ce manuscrit ont été apportées en réponse à un besoin réel

dans les projets de design en collaboration avec des équipes d'ingénierie de protéines. Les algorithmes ont été implémentés dans un logiciel de design qui fait appel à Rosetta pour calculer les matrices d'énergie et à Toulbar2 pour l'optimisation. Ils ont été utilisés pour prédire une librairie de séquences diverses avec un nombre contrôlé de mutations dans le cadre d'un projet de design multi-états positif. Des tests expérimentaux ont ensuite été conduits afin de sélectionner la séquence la plus pertinente dans la librairie. Une autre collaboration récente utilisera l'approche d'optimisation biniveau pour le design négatif d'une nouvelle protéine, qui sera ensuite testée expérimentalement.

Bibliography

[Adhikary *et al.* 1997] Junas Adhikary, Geir Hasle and Gunnar Misund. *Constraint technology applied to forest treatment scheduling.* In Proc. of Practical Application of Constraint Technology (PACT97), London, UK, volume 1997, 1997. (Cited on page 6.)

[Aji & McEliece 2000] Srinivas M Aji and Robert J McEliece. *The generalized distributive law.* IEEE transactions on Information Theory, vol. 46, no. 2, pages 325–343, 2000. (Cited on page 29.)

[Alford *et al.* 2017] Rebecca F Alford, Andrew Leaver-Fay, Jeliazko R Jeliazkov, Matthew J O'Meara, Frank P DiMaio, Hahnbeom Park, Maxim V Shapovalov, P Douglas Renfrew, Vikram K Mulligan, Kalli Kappel*et al. The Rosetta all-atom energy function for macromolecular modeling and design.* Journal of chemical theory and computation, vol. 13, no. 6, pages 3031–3048, 2017. (Cited on page 121.)

[Allen & Mayo 2010] Benjamin D Allen and Stephen L Mayo. *An efficient algorithm for multistate protein design based on FASTER.* Journal of computational chemistry, vol. 31, no. 5, pages 904–916, 2010. (Cited on page 114.)

[Allouche *et al.* 2012] David Allouche, Christian Bessière, Patrice Boizumault, Simon De Givry, Patricia Gutierrez, Samir Loudni, Jean-Philippe Metivier and Thomas Schiex. *Filtering decomposable global cost functions.* In AAAI Conference on Artificial Intelligence. AAAI Press, 2012. (Cited on pages 46, 145 and 146.)

[Allouche *et al.* 2014] David Allouche, Isabelle André, Sophie Barbe, Jessica Davies, Simon de Givry, George Katsirelos, Barry O'Sullivan, Steve Prestwich, Thomas Schiex and Seydou Traoré. *Computational protein design as an optimization problem.* Artificial Intelligence, vol. 212, pages 59–79, 2014. (Cited on pages 64, 69 and 71.)

[Allouche *et al.* 2015] David Allouche, Simon De Givry, George Katsirelos, Thomas Schiex and Matthias Zytnicki. *Anytime hybrid best-first search with tree decomposition for weighted CSP.* In International Conference on Principles and Practice of Constraint Programming, pages 12–29. Springer, 2015. (Cited on pages 29 and 140.)

[Allouche *et al.* 2016] David Allouche, Christian Bessiere, Patrice Boizumault, Simon De Givry, Patricia Gutierrez, Jimmy HM Lee, Ka Lun

Leung, Samir Loudni, Jean-Philippe Métivier, Thomas Schiex*et al. Tractability-preserving transformations of global cost functions.* Artificial Intelligence, vol. 238, pages 166–189, 2016. (Cited on pages 46, 47, 48, 49, 85 and 88.)

[Ambroggio & Kuhlman 2006] Xavier I Ambroggio and Brian Kuhlman. *Computational design of a single amino acid sequence that can switch between two distinct protein folds.* Journal of the American Chemical Society, vol. 128, no. 4, pages 1154–1161, 2006. (Cited on page 114.)

[Anfinsen 1973] Christian B Anfinsen. *Principles that govern the folding of protein chains.* Science, vol. 181, no. 4096, pages 223–230, 1973. (Cited on page 59.)

[Bacchus & Van Beek 1998] Fahiem Bacchus and Peter Van Beek. *On the conversion between non-binary and binary constraint satisfaction problems.* In AAAI/IAAI, pages 310–318, 1998. (Cited on pages 90 and 147.)

[Barták 1999] Roman Barták. *Constraint programming: In pursuit of the holy grail.* In Proceedings of the Week of Doctoral Students (WDS99), volume 4, pages 555–564. MatFyzPress Prague, 1999. (Cited on pages 12 and 13.)

[Batra *et al.* 2012] Dhruv Batra, Payman Yadollahpour, Abner Guzman-Rivera and Gregory Shakhnarovich. *Diverse m-best solutions in markov random fields.* In European Conference on Computer Vision, pages 1–16. Springer, 2012. (Cited on pages 81 and 82.)

[Berenger *et al.* 2012] Francois Berenger, Rojan Shrestha, Yong Zhou, David Simoncini and Kam YJ Zhang. *Durandal: fast exact clustering of protein decoys.* Journal of computational chemistry, vol. 33, no. 4, pages 471–474, 2012. (Cited on page 121.)

[Bergman *et al.* 2011] David Bergman, Willem-Jan Van Hoeve and John N Hooker. *Manipulating MDD relaxations for combinatorial optimization.* In International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems, pages 20–35. Springer, 2011. (Cited on pages 85 and 94.)

[Bergman *et al.* 2016] David Bergman, Andre A Cire, Willem-Jan Van Hoeve and John Hooker. Decision diagrams for optimization, volume 1. Springer, 2016. (Cited on pages 59 and 85.)

[Berman *et al.* 2000] Helen M Berman, John Westbrook, Zukang Feng, Gary Gilliland, Talapady N Bhat, Helge Weissig, Ilya N Shindyalov and Philip E Bourne. *The protein data bank.* Nucleic acids research, vol. 28, no. 1, pages 235–242, 2000. (Cited on page 59.)

[Bertele & Brioschi 1973] Umberto Bertele and Francesco Brioschi. *On non-serial dynamic programming.* J. Comb. Theory, Ser. A, vol. 14, no. 2, pages 137–148, 1973. (Cited on pages 18, 29 and 30.)

[Bertsekas *et al.* 1999] Dimitri P Bertsekas, W Hager and O Mangasarian. *Nonlinear Programming. Athena Scientific Belmont.* Massachusets, USA, 1999. (Cited on page 83.)

[Bessière & Régin 2001] Christian Bessière and Jean-Charles Régin. *Refining the Basic Constraint Propagation Algorithm.* In IJCAI, volume 1, pages 309–315, 2001. (Cited on page 13.)

[Bessière & Van Hentenryck 2003] C. Bessière and P. Van Hentenryck. *To be or not to be ... a global constraint.* In Proceedings of CP'03, pages 789–794, 2003. (Cited on page 145.)

[Bessiere 1994] Christian Bessiere. *Arc-consistency and arc-consistency again.* Artificial intelligence, vol. 65, no. 1, pages 179–190, 1994. (Cited on page 13.)

[Bhella 2019] David Bhella. *Cryo-electron microscopy: an introduction to the technique, and considerations when working to establish a national facility.* Biophysical reviews, vol. 11, no. 4, pages 515–519, 2019. (Cited on page 59.)

[Bodlaender & Koster 2010] Hans L Bodlaender and Arie MCA Koster. *Treewidth computations I. Upper bounds.* Information and Computation, vol. 208, no. 3, pages 259–275, 2010. (Cited on page 32.)

[Borning 1981] Alan Borning. *The programming language aspects of ThingLab, a constraint-oriented simulation laboratory.* ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 3, no. 4, pages 353–387, 1981. (Cited on page 6.)

[Boussemart *et al.* 2004] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre and Lakhdar Sais. *Boosting systematic search by weighting constraints.* In ECAI, volume 16, page 146, 2004. (Cited on page 50.)

[Brooks *et al.* 2009] Bernard R Brooks, Charles L Brooks III, Alexander D Mackerell Jr, Lennart Nilsson, Robert J Petrella, Benoît

Roux, Youngdo Won, Georgios Archontis, Christian Bartels, Stefan Boresch*et al. CHARMM: the biomolecular simulation program.* Journal of computational chemistry, vol. 30, no. 10, pages 1545–1614, 2009. (Cited on page 64.)

[Callaway 2020] Ewen Callaway. *'It will change everything': DeepMind's AI makes gigantic leap in solving protein structures.* Nature, 2020. (Cited on page 61.)

[Carton 2008] Olivier Carton. Langages formels, calculabilité et complexité, volume 28. Vuibert, 2008. (Cited on pages 15 and 18.)

[Case *et al.* 2005] David A Case, Thomas E Cheatham III, Tom Darden, Holger Gohlke, Ray Luo, Kenneth M Merz Jr, Alexey Onufriev, Carlos Simmerling, Bing Wang and Robert J Woods. *The Amber biomolecular simulation programs.* Journal of computational chemistry, vol. 26, no. 16, pages 1668–1688, 2005. (Cited on page 64.)

[Chan *et al.* 1998] Peter Chan, Kamel Heus and Georges Weil. *Nurse scheduling with global constraints in CHIP: Gymnaste.* In Proc of Practical Application of Constraint Technology (PACT98), London, UK, 1998. (Cited on page 6.)

[Charpentier *et al.* 2018] Antoine Charpentier, David Mignon, Sophie Barbe, Juan Cortes, Thomas Schiex, Thomas Simonson and David Allouche. *Variable neighborhood search with cost function networks to solve large computational protein design problems.* Journal of chemical information and modeling, vol. 59, no. 1, pages 127–136, 2018. (Cited on page 50.)

[Chen *et al.* 2009] Cheng-Yu Chen, Ivelin Georgiev, Amy C Anderson and Bruce R Donald. *Computational structure-based redesign of enzyme activity.* Proceedings of the National Academy of Sciences, vol. 106, no. 10, pages 3764–3769, 2009. (Cited on page 72.)

[Chen *et al.* 2013] Chao Chen, Vladimir Kolmogorov, Yan Zhu, Dimitris Metaxas and Christoph Lampert. *Computing the M most probable modes of a graphical model.* In Proc. of Artificial Intelligence and Statistics, pages 161–169, 2013. (Cited on page 81.)

[Chen *et al.* 2014] Chao Chen, Han Liu, Dimitris Metaxas and Tianqi Zhao. *Mode estimation for high dimensional discrete tree graphical models.* In Proceedings of Advances in neural information processing systems, pages 1323–1331, 2014. (Cited on page 81.)

[Chen *et al.* 2016] Cong Chen, Changhe Yuan and Chao Chen. *Solving M-Modes Using Heuristic Search.* In Proc. of IJCAI'16, pages 3584–3590, 2016. (Cited on page 81.)

[Chen *et al.* 2018] Cong Chen, Changhe Yuan, Ze Ye and Chao Chen. *Solving M-Modes in Loopy Graphs Using Tree Decompositions.* In Proc. of the International Conference on Probabilistic Graphical Models, pages 145–156, 2018. (Cited on pages 81, 97, 98, 108 and 148.)

[Chica *et al.* 2010] Roberto A Chica, Matthew M Moore, Benjamin D Allen and Stephen L Mayo. *Generation of longer emission wavelength red fluorescent proteins using computationally designed libraries.* Proceedings of the National Academy of Sciences, vol. 107, no. 47, pages 20257–20262, 2010. (Cited on page 72.)

[Chow & Perett 1997] Kam Pui Chow and M Perett. *Airport counter allocation using constraint logic programming.* In Proceedings of the third international conference on practical application of constraint technology. The Practical Application Company., 1997. (Cited on page 6.)

[Chowdry *et al.* 2007] Arnab B Chowdry, Kimberly A Reynolds, Melinda S Hanes, Mark Voorhies, Navin Pokala and Tracy M Handel. *An object-oriented library for computational protein design.* Journal of computational chemistry, vol. 28, no. 14, pages 2378–2388, 2007. (Cited on page 66.)

[Consortium 2007] UniProt Consortium. *The universal protein resource (UniProt).* Nucleic acids research, vol. 36, no. suppl_1, pages D190–D195, 2007. (Cited on page 60.)

[Cook 1971] Stephen A Cook. *The complexity of theorem-proving procedures.* In Proceedings of the third annual ACM symposium on Theory of computing, pages 151–158, 1971. (Cited on page 16.)

[Cooper & Schiex 2004] Martin Cooper and Thomas Schiex. *Arc consistency for soft constraints.* Artificial Intelligence, vol. 154, no. 1-2, pages 199–227, 2004. (Cited on pages 34 and 39.)

[Cooper *et al.* 1994] Martin C Cooper, David A Cohen and Peter G Jeavons. *Characterising tractable constraints.* Artificial Intelligence, vol. 65, no. 2, pages 347–361, 1994. (Cited on page 18.)

[Cooper *et al.* 2007] Martin Cooper, Simon De Givry and Thomas Schiex. *Optimal soft arc consistency.* In 20th International Joint Conference

on Artificial Intelligence-IJCAI 2007, page 6. Morgan Kaufmann, 2007. (Cited on pages 43 and 44.)

[Cooper *et al.* 2008] Martin C Cooper, Simon De Givry, Martí Sánchez-Fibla, Thomas Schiex and Matthias Zytnicki. *Virtual Arc Consistency for Weighted CSP.* In AAAI, pages 253–258, 2008. (Cited on pages 44, 45 and 46.)

[Cooper *et al.* 2010] Martin C Cooper, Simon De Givry, Martı Sánchez, Thomas Schiex, Matthias Zytnicki and Tomas Werner. *Soft arc consistency revisited.* Artificial Intelligence, vol. 174, no. 7-8, pages 449–478, 2010. (Cited on pages 44, 46, 140 and 141.)

[Cooper *et al.* 2020] Martin Cooper, Simon de Givry and Thomas Schiex. *Graphical Models: Queries, Complexity, Algorithms.* Leibniz International Proceedings in Informatics, vol. 154, pages 4–1, 2020. (Cited on pages 19, 24, 29, 30 and 42.)

[Cormen *et al.* 2009] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest and Clifford Stein. Introduction to algorithms. MIT press, 2009. (Cited on pages 9, 18 and 48.)

[Cplex 2009] IBM ILOG Cplex. *V12. 1: User's Manual for CPLEX.* International Business Machines Corporation, vol. 46, no. 53, page 157, 2009. (Cited on page 71.)

[Crochemore & Rytter 1994] Maxime Crochemore and Wojciech Rytter. Text algorithms. Maxime Crochemore, 1994. (Cited on page 47.)

[Dahiyat & Mayo 1997] Bassil I Dahiyat and Stephen L Mayo. *De novo protein design: fully automated sequence selection.* Science, vol. 278, no. 5335, pages 82–87, 1997. (Cited on page 72.)

[Davey & Chica 2012] James A Davey and Roberto A Chica. *Multistate approaches in computational protein design.* Protein Science, vol. 21, no. 9, pages 1241–1252, 2012. (Cited on page 114.)

[Davis *et al.* 2006] Ian W Davis, W Bryan Arendall III, David C Richardson and Jane S Richardson. *The backrub motion: how protein backbone shrugs when a sidechain dances.* Structure, vol. 14, no. 2, pages 265–274, 2006. (Cited on page 120.)

[De Givry *et al.* 2005] Simon De Givry, Federico Heras, Matthias Zytnicki and Javier Larrosa. *Existential arc consistency: Getting closer to full arc consistency in weighted CSPs.* In IJCAI, volume 5, pages 84–89, 2005. (Cited on pages 37 and 41.)

[De Givry *et al.* 2006] Simon De Givry, Thomas Schiex and Gerard Verfaillie. *Exploiting tree decomposition and soft local consistency in weighted CSP*. In AAAI, volume 6, pages 1–6, 2006. (Cited on page 32.)

[De Givry *et al.* 2013] Simon De Givry, Steven D Prestwich and Barry O'Sullivan. *Dead-end elimination for weighted CSP*. In International Conference on Principles and Practice of Constraint Programming, pages 263–272. Springer, 2013. (Cited on page 50.)

[Dechter & Frost 2002] Rina Dechter and Daniel Frost. *Backjump-based backtracking for constraint satisfaction problems*. Artificial Intelligence, vol. 136, no. 2, pages 147–188, 2002. (Cited on page 50.)

[Dechter & Mateescu 2007] Rina Dechter and Robert Mateescu. *AND/OR search spaces for graphical models*. Artificial intelligence, vol. 171, no. 2-3, pages 73–106, 2007. (Cited on pages 32 and 71.)

[Dechter & Pearl 1988] Rina Dechter and Judea Pearl. *Network-based heuristics for constraint-satisfaction problems*. In Search in artificial intelligence, pages 370–425. Springer, 1988. (Cited on page 32.)

[Dechter & Pearl 1989] Rina Dechter and Judea Pearl. *Tree clustering for constraint networks*. Artificial Intelligence, vol. 38, no. 3, pages 353–366, 1989. (Cited on page 32.)

[Dechter & Rish 2003] Rina Dechter and Irina Rish. *Mini-buckets: A general scheme for bounded inference*. Journal of the ACM (JACM), vol. 50, no. 2, pages 107–153, 2003. (Cited on pages 28, 41 and 71.)

[Dechter 1997] Rina Dechter. *Mini-buckets: A general scheme for generating approximations in automated reasoning*. In IJCAI, volume 97, pages 1297–1303. Citeseer, 1997. (Cited on pages 28, 32, 41 and 71.)

[Dechter 1999] Rina Dechter. *Bucket elimination: A unifying framework for reasoning*. Artificial Intelligence, vol. 113, no. 1-2, pages 41–85, 1999. (Cited on pages 24, 29 and 30.)

[Dechter 2003] Rina Dechter. Constraint processing. Morgan Kaufmann, 2003. (Cited on pages 6, 12, 13 and 18.)

[Dechter 2013] Rina Dechter. *Reasoning with probabilistic and deterministic graphical models: Exact algorithms*. Synthesis Lectures on Artificial Intelligence and Machine Learning, vol. 7, no. 3, pages 1–191, 2013. (Cited on pages 19, 29 and 30.)

[Dechter 2019] Rina Dechter. *Reasoning with probabilistic and deterministic graphical models: exact algorithms*. Synthesis Lectures on Artificial Intelligence and Machine Learning, vol. 13, no. 1, pages 1–199, 2019. (Cited on pages 19 and 24.)

[DeGrado *et al.* 1985] William F DeGrado, Frank G Prendergast, Henry R Wolfe Jr and Jos A Cox. *The design, synthesis, and characterization of tight-binding inhibitors of calmodulin*. Journal of cellular biochemistry, vol. 29, no. 2, pages 83–93, 1985. (Cited on page 71.)

[Desmet *et al.* 1992] Johan Desmet, Marc De Maeyer, Bart Hazes and Ignace Lasters. *The dead-end elimination theorem and its use in protein side-chain positioning*. Nature, vol. 356, no. 6369, pages 539–542, 1992. (Cited on page 66.)

[Dill & MacCallum 2012] Ken A Dill and Justin L MacCallum. *The protein-folding problem, 50 years on*. science, vol. 338, no. 6110, pages 1042–1046, 2012. (Cited on page 60.)

[Dill *et al.* 2008] Ken A Dill, S Banu Ozkan, M Scott Shell and Thomas R Weikl. *The protein folding problem*. Annu. Rev. Biophys., vol. 37, pages 289–316, 2008. (Cited on page 60.)

[Dincbas & Simonis 1991] Mehmet Dincbas and Helmut Simonis. *Apache-a constraint based, automated stand allocation system*. In Proc. of Advanced Software Technology in Air Transport (ASTAIR91), London, UK, 1991. (Cited on page 6.)

[Dubois & Prade 1980] Didier Dubois and Henri Prade. *Systems of linear fuzzy constraints*. Fuzzy sets and systems, vol. 3, no. 1, pages 37–48, 1980. (Cited on page 21.)

[Flerova *et al.* 2017] Natalia Flerova, Radu Marinescu and Rina Dechter. *Weighted heuristic anytime search: new schemes for optimization over graphical models*. Annals of Mathematics and Artificial Intelligence, vol. 79, no. 1-3, pages 77–128, 2017. (Cited on page 32.)

[Freuder & Quinn 1985] Eugene C Freuder and Michael J Quinn. *Taking Advantage of Stable Sets of Variables in Constraint Satisfaction Problems*. In IJCAI, volume 85, pages 1076–1078. Citeseer, 1985. (Cited on page 32.)

[Freuder & Wallace 1992] Eugene C Freuder and Richard J Wallace. *Partial constraint satisfaction*. Artificial Intelligence, vol. 58, no. 1-3, pages 21–70, 1992. (Cited on pages 19 and 24.)

[Freuder 1978] Eugene C Freuder. *Synthesizing constraint expressions.* Communications of the ACM, vol. 21, no. 11, pages 958–966, 1978. (Cited on page 6.)

[Freuder 1982] Eugene C Freuder. *A sufficient condition for backtrack-free search.* Journal of the ACM (JACM), vol. 29, no. 1, pages 24–32, 1982. (Cited on pages 6, 18 and 32.)

[Freuder 1991] Eugene C Freuder. *Eliminating interchangeable values in constraint satisfaction problems.* In AAAI, volume 91, pages 227–233, 1991. (Cited on page 50.)

[Gainza *et al.* 2013] 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.* In Methods in enzymology, volume 523, pages 87–107. Elsevier, 2013. (Cited on pages 65, 67, 71 and 115.)

[Gaspin & Westhof 1994] Christine Gaspin and Eric Westhof. *The determination of the secondary structures of RNA as a constraint satisfaction problem*, 1994. (Cited on page 6.)

[Gaspin *et al.* 2005] Christine Gaspin, Thomas Schiex and Matthias Zytnicki. *Bound arc consistency for weighted CSPs.* In 7th International CP-2005 Workshop on Preferences and Soft Constraints, 2005. (Cited on page 39.)

[Georgiev *et al.* 2006] Ivelin Georgiev, Ryan H Lilien and Bruce R Donald. *Improved pruning algorithms and divide-and-conquer strategies for dead-end elimination, with application to protein design.* Bioinformatics, vol. 22, no. 14, pages e174–e183, 2006. (Cited on page 66.)

[Georgiev *et al.* 2008] Ivelin Georgiev, Ryan H Lilien and Bruce R Donald. *The minimized dead-end elimination criterion and its application to protein redesign in a hybrid scoring and search algorithm for computing partition functions over molecular ensembles.* Journal of computational chemistry, vol. 29, no. 10, pages 1527–1542, 2008. (Cited on page 67.)

[Goldstein 1994] Robert F Goldstein. *Efficient rotamer elimination applied to protein side-chains and related spin glasses.* Biophysical journal, vol. 66, no. 5, pages 1335–1340, 1994. (Cited on page 66.)

[Grigoryan *et al.* 2009] Gevorg Grigoryan, Aaron W Reinke and Amy E Keating. *Design of protein-interaction specificity gives selective bZIP-binding peptides.* Nature, vol. 458, no. 7240, pages 859–864, 2009. (Cited on page 114.)

[Gu & Bourne 2009] Jenny Gu and Philip E Bourne. Structural bioinformatics, volume 44. John Wiley & Sons, 2009. (Cited on pages 54 and 59.)

[Hadzic *et al.* 2008] Tarik Hadzic, Esben Rune Hansen and Barry O'Sullivan. *On automata, MDDs and BDDs in constraint satisfaction.* In Proceedings of the ECAI 2008 Workshop on Inference Methods based on Graphical Structures of Knowledge, 2008. (Cited on page 87.)

[Hadžić *et al.* 2009] Tarik Hadžić, Alan Holland and Barry O'Sullivan. *Reasoning about optimal collections of solutions.* In International Conference on Principles and Practice of Constraint Programming, pages 409–423. Springer, 2009. (Cited on page 81.)

[Hallen & Donald 2016] Mark A Hallen and Bruce R Donald. *COMETS (Constrained Optimization of Multistate Energies by Tree Search): A provable and efficient protein design algorithm to optimize binding affinity and specificity with respect to sequence.* Journal of Computational Biology, vol. 23, no. 5, pages 311–321, 2016. (Cited on page 115.)

[Hallen & Donald 2017] Mark A Hallen and Bruce R Donald. *CATS (Coordinates of Atoms by Taylor Series): protein design with backbone flexibility in all locally feasible directions.* Bioinformatics, vol. 33, no. 14, pages i5–i12, 2017. (Cited on page 67.)

[Hallen & Donald 2019] Mark A Hallen and Bruce R Donald. *Protein design by provable algorithms.* Communications of the ACM, vol. 62, no. 10, pages 76–84, 2019. (Cited on pages 67, 73 and 115.)

[Hallen *et al.* 2013] 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, vol. 81, no. 1, pages 18–39, 2013. (Cited on page 67.)

[Hallen *et al.* 2017] Mark A Hallen, Jonathan D Jou and Bruce R Donald. *LUTE (Local unpruned tuple expansion): Accurate continuously flexible protein design with general energy functions and rigid Rotamer-Like efficiency.* Journal of Computational Biology, vol. 24, no. 6, pages 536–546, 2017. (Cited on page 67.)

[Hallen *et al.* 2018] Mark A Hallen, Jeffrey W Martin, Adegoke Ojewole, Jonathan D Jou, Anna U Lowegard, Marcel S Frenkel, Pablo Gainza, Hunter M Nisonoff, Aditya Mukund, Siyu Wang*et al. OSPREY 3.0: Open-source protein redesign for you, with powerful new features.* Journal of computational chemistry, vol. 39, no. 30, pages 2494–2507, 2018. (Cited on page 67.)

[Harbury *et al.* 1998] Pehr B Harbury, Joseph J Plecs, Bruce Tidor, Tom Alber and Peter S Kim. *High-resolution protein design with backbone freedom.* Science, vol. 282, no. 5393, pages 1462–1467, 1998. (Cited on page 72.)

[Hart *et al.* 1968] Peter E Hart, Nils J Nilsson and Bertram Raphael. *A formal basis for the heuristic determination of minimum cost paths.* IEEE transactions on Systems Science and Cybernetics, vol. 4, no. 2, pages 100–107, 1968. (Cited on page 28.)

[Hasle *et al.* 1996] G Hasle, R Haut, B Johansen and T Ølberg. *Well activity scheduling-an application of constraint reasoning.* Artificial Intelligence in the Petroleum Industry: Symbolic and Computational Applications II, pages 209–228, 1996. (Cited on page 6.)

[Havranek & Harbury 2003] James J Havranek and Pehr B Harbury. *Automated design of specificity in molecular recognition.* Nature structural biology, vol. 10, no. 1, pages 45–52, 2003. (Cited on page 114.)

[Hebrard *et al.* 2005] Emmanuel Hebrard, Brahim Hnich, Barry O'Sullivan and Toby Walsh. *Finding diverse and similar solutions in constraint programming.* In Proceedings of AAAI 2005, volume 5, pages 372–377, 2005. (Cited on pages 81 and 93.)

[Hebrard *et al.* 2007] Emmanuel Hebrard, Barry O'Sullivan and Toby Walsh. *Distance Constraints in Constraint Satisfaction.* In IJCAI, volume 2007, pages 106–111, 2007. (Cited on page 81.)

[Henikoff & Henikoff 1992] Steven Henikoff and Jorja G Henikoff. *Amino acid substitution matrices from protein blocks.* Proceedings of the National Academy of Sciences, vol. 89, no. 22, pages 10915–10919, 1992. (Cited on pages 68 and 77.)

[Henzler-Wildman *et al.* 2007] Katherine A Henzler-Wildman, Ming Lei, Vu Thai, S Jordan Kerns, Martin Karplus and Dorothee Kern. *A hierarchy of timescales in protein dynamics is linked to enzyme catalysis.* Nature, vol. 450, no. 7171, pages 913–916, 2007. (Cited on page 59.)

[Heras *et al.* 2008] Federico Heras, Javier Larrosa and Albert Oliveras. *MiniMaxSAT: An efficient weighted Max-SAT solver*. Journal of Artificial Intelligence Research, vol. 31, pages 1–32, 2008. (Cited on page 71.)

[Hsia *et al.* 2016] Yang Hsia, Jacob B Bale, Shane Gonen, Dan Shi, William Sheffler, Kimberly K Fong, Una Nattermann, Chunfu Xu, Po-Ssu Huang, Rashmi Ravichandran*et al. Design of a hyperstable 60-subunit protein icosahedron*. Nature, vol. 535, no. 7610, pages 136–139, 2016. (Cited on page 72.)

[Hurley *et al.* 2016] Barry Hurley, Barry O'Sullivan, David Allouche, George Katsirelos, Thomas Schiex, Matthias Zytnicki and Simon De Givry. *Multi-language evaluation of exact solvers in graphical model discrete optimization*. Constraints, vol. 21, no. 3, pages 413–434, 2016. (Cited on pages 140 and 148.)

[Jégou *et al.* 2017] Philippe Jégou, Hélène Kanso and Cyril Terrioux. *Adaptive and Opportunistic Exploitation of Tree-decompositions for Weighted CSPs*. In 2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI), pages 366–373. IEEE, 2017. (Cited on page 32.)

[Jiang *et al.* 2008] 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, vol. 319, no. 5868, pages 1387–1391, 2008. (Cited on page 72.)

[Jones 1994] David T Jones. *De novo protein design using pairwise potentials and a genetic algorithm*. Protein Science, vol. 3, no. 4, pages 567–574, 1994. (Cited on page 66.)

[Joyce 2007] Gerald F Joyce. *Forty years of in vitro evolution*. Angewandte Chemie International Edition, vol. 46, no. 34, pages 6420–6436, 2007. (Cited on page 61.)

[Kaiser *et al.* 2007] Chris A Kaiser, Monty Krieger, Harvey Lodish and Arnold Berk. *Molecular Cell Biology.*, 2007. (Cited on page 54.)

[Karimi & Shen 2018] Mostafa Karimi and Yang Shen. *iCFN: an efficient exact algorithm for multistate protein design*. Bioinformatics, vol. 34, no. 17, pages i811–i820, 2018. (Cited on pages 115, 120, 123, 129 and 154.)

[Karmarkar 1984] Narendra Karmarkar. *A new polynomial-time algorithm for linear programming*. In Proceedings of the sixteenth annual ACM symposium on Theory of computing, pages 302–311, 1984. (Cited on page 43.)

[Kindermann 1980] Ross Kindermann. *Markov random fields and their applications*. American mathematical society, 1980. (Cited on pages 19 and 22.)

[Kirillov *et al.* 2015] Alexander Kirillov, Bogdan Savchynskyy, Dmitrij Schlesinger, Dmitry Vetrov and Carsten Rother. *Inferring M-best diverse labelings in a single one*. In Proceedings of the IEEE International Conference on Computer Vision, pages 1814–1822, 2015. (Cited on page 81.)

[Klir & Yuan 1995] George Klir and Bo Yuan. Fuzzy sets and fuzzy logic, volume 4. Prentice hall New Jersey, 1995. (Cited on page 21.)

[Koller & Friedman 2009] Daphne Koller and Nir Friedman. Probabilistic graphical models: principles and techniques. MIT press, 2009. (Cited on page 29.)

[Kolmogorov 2006] Vladimir Kolmogorov. *Convergent tree-reweighted message passing for energy minimization*. IEEE transactions on pattern analysis and machine intelligence, vol. 28, no. 10, pages 1568–1583, 2006. (Cited on pages 46 and 141.)

[Koval' & Schlesinger 1976] VK Koval' and Mykhailo Ivanovich Schlesinger. *Two-dimensional programming in image analysis problems*. Avtomatika i Telemekhanika, no. 8, pages 149–168, 1976. (Cited on page 33.)

[Kryshtafovych *et al.* 2019] Andriy Kryshtafovych, Torsten Schwede, Maya Topf, Krzysztof Fidelis and John Moult. *Critical assessment of methods of protein structure prediction (CASP)—Round XIII*. Proteins: Structure, Function, and Bioinformatics, vol. 87, no. 12, pages 1011–1020, 2019. (Cited on page 60.)

[Kschischang *et al.* 2001] Frank R Kschischang, Brendan J Frey and H-A Loeliger. *Factor graphs and the sum-product algorithm*. IEEE Transactions on information theory, vol. 47, no. 2, pages 498–519, 2001. (Cited on page 29.)

[Kuhlman *et al.* 2003] Brian Kuhlman, Gautam Dantas, Gregory C Ireton, Gabriele Varani, Barry L Stoddard and David Baker. *Design of a*

novel globular protein fold with atomic-level accuracy. science, vol. 302, no. 5649, pages 1364–1368, 2003. (Cited on page 72.)

[Larrosa & Dechter 2000] Javier Larrosa and Rina Dechter. *On the dual representation of non-binary semiring-based CSPs.* In CP'2000 workshop on soft constraints, 2000. (Cited on page 90.)

[Larrosa & Schiex 2003] Javier Larrosa and Thomas Schiex. *In the quest of the best form of local consistency for weighted CSP.* In IJCAI, volume 3, pages 239–244, 2003. (Cited on pages 36, 38, 39 and 41.)

[Larrosa & Schiex 2004] Javier Larrosa and Thomas Schiex. *Solving weighted CSP by maintaining arc consistency.* Artificial Intelligence, vol. 159, no. 1-2, pages 1–26, 2004. (Cited on page 36.)

[Lauriere 1978] Jena-Lonis Lauriere. *A language and a program for stating and solving combinatorial problems.* Artificial intelligence, vol. 10, no. 1, pages 29–127, 1978. (Cited on page 6.)

[Leach & Lemon 1998] Andrew R Leach and Andrew P Lemon. *Exploring the conformational space of protein side chains using dead-end elimination and the A* algorithm.* Proteins: Structure, Function, and Bioinformatics, vol. 33, no. 2, pages 227–239, 1998. (Cited on page 67.)

[Leaver-Fay *et al.* 2011a] Andrew Leaver-Fay, Ron Jacak, P Benjamin Stranges and Brian Kuhlman. *A generic program for multistate protein design.* PloS one, vol. 6, no. 7, page e20937, 2011. (Cited on page 114.)

[Leaver-Fay *et al.* 2011b] 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*et al. ROSETTA3: an object-oriented software suite for the simulation and design of macromolecules.* In Methods in enzymology, volume 487, pages 545–574. Elsevier, 2011. (Cited on pages 64, 65 and 66.)

[Lecoutre *et al.* 2006] Christophe Lecoutre, Lakhdar Sais, Sébastien Tabary and Vincent Vidal. *Last conflict based reasoning.* In ECAI, pages 133–137, 2006. (Cited on page 15.)

[Lecoutre *et al.* 2009] Christophe Lecoutre, Lakhdar Saïs, Sébastien Tabary and Vincent Vidal. *Reasoning from last conflict (s) in constraint programming.* Artificial Intelligence, vol. 173, no. 18, pages 1592–1614, 2009. (Cited on page 50.)

[Lecoutre *et al.* 2012] Christophe Lecoutre, Olivier Roussel and Djamel E Dehani. *WCSP integration of soft neighborhood substitutability.* In International Conference on Principles and Practice of Constraint Programming, pages 406–421. Springer, 2012. (Cited on page 50.)

[Lee & Leung 2009] Jimmy HM Lee and Ka Lun Leung. *Towards efficient consistency enforcement for global constraints in weighted constraint satisfaction.* In Twenty-First International Joint Conference on Artificial Intelligence, 2009. (Cited on pages 46 and 48.)

[Lee & Leung 2012] Jimmy Ho-Man Lee and Ka Lun Leung. *Consistency techniques for flow-based projection-safe global cost functions in weighted constraint satisfaction.* Journal of Artificial Intelligence Research, vol. 43, pages 257–292, 2012. (Cited on pages 46, 48, 88 and 145.)

[Levin 1973] Leonid Anatolevich Levin. *Universal sequential search problems.* Problemy peredachi informatsii, vol. 9, no. 3, pages 115–116, 1973. (Cited on page 16.)

[Levinthal 1968] Cyrus Levinthal. *Are there pathways for protein folding?* Journal de chimie physique, vol. 65, pages 44–45, 1968. (Cited on page 59.)

[Li 2009] Stan Z Li. Markov random field modeling in image analysis. Springer Science & Business Media, 2009. (Cited on page 22.)

[Linz 2006] Peter Linz. An introduction to formal languages and automata. Jones & Bartlett Learning, 2006. (Cited on page 47.)

[Löffler *et al.* 2017] Patrick Löffler, Samuel Schmitz, Enrico Hupfeld, Reinhard Sterner and Rainer Merkl. *Rosetta: MSF: a modular framework for multi-state computational protein design.* PLoS computational biology, vol. 13, no. 6, page e1005600, 2017. (Cited on page 114.)

[Looger & Hellinga 2001] Loren L Looger and Homme W Hellinga. *Generalized dead-end elimination algorithms make large-scale protein side-chain structure prediction tractable: implications for protein design and structural genomics.* Journal of molecular biology, vol. 307, no. 1, pages 429–445, 2001. (Cited on page 66.)

[Lovell *et al.* 2000] Simon C Lovell, J Michael Word, Jane S Richardson and David C Richardson. *The penultimate rotamer library.* Proteins: Structure, Function, and Bioinformatics, vol. 40, no. 3, pages 389–408, 2000. (Cited on page 63.)

[Mackworth 1977] Alan K Mackworth. *Consistency in networks of relations.* Artificial intelligence, vol. 8, no. 1, pages 99–118, 1977. (Cited on page 13.)

[Marcos *et al.* 2017] Enrique Marcos, Benjamin Basanta, Tamuka M Chidyausiku, Yuefeng Tang, Gustav Oberdorfer, Gaohua Liu, GVT Swapna, Rongjin Guan, Daniel-Adriano Silva, Jiayi Dou*et al. Principles for designing proteins with cavities formed by curved β sheets.* Science, vol. 355, no. 6321, pages 201–206, 2017. (Cited on page 69.)

[Marinescu & Dechter 2006] Radu Marinescu and Rina Dechter. *AND/OR Graph Search for Genetic Linkage Analysis.* In Workshop on Heuristic Search, Memory Based Heuristics and Their Applications, AAAI, volume 6, 2006. (Cited on pages 6 and 71.)

[Marquis *et al.* 2020] P Marquis, H Parde and O Papini. *A Guided Tour of Artificial Intelligence Research.* 2020. (Cited on pages 6, 19, 22 and 50.)

[Marriott *et al.* 2008] Kim Marriott, Nicholas Nethercote, Reza Rafeh, Peter J Stuckey, Maria Garcia De La Banda and Mark Wallace. *The design of the Zinc modelling language.* Constraints, vol. 13, no. 3, pages 229–267, 2008. (Cited on page 71.)

[Meseguer *et al.* 2006] Pedro Meseguer, Francesca Rossi and Thomas Schiex. *Soft constraints.* In Foundations of Artificial Intelligence, volume 2, pages 281–328. Elsevier, 2006. (Cited on pages 19 and 22.)

[Mladenović & Hansen 1997] Nenad Mladenović and Pierre Hansen. *Variable neighborhood search.* Computers & operations research, vol. 24, no. 11, pages 1097–1100, 1997. (Cited on page 50.)

[Mohr & Henderson 1986] Roger Mohr and Thomas C Henderson. *Arc and path consistency revisited.* Artificial intelligence, vol. 28, no. 2, pages 225–233, 1986. (Cited on page 13.)

[Montanari 1974] Ugo Montanari. *Networks of constraints: Fundamental properties and applications to picture processing.* Information sciences, vol. 7, pages 95–132, 1974. (Cited on page 6.)

[Moskewicz *et al.* 2001] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang and Sharad Malik. *Chaff: Engineering an efficient SAT solver.* In Proceedings of the 38th annual Design Automation Conference, pages 530–535, 2001. (Cited on page 15.)

[Murzin *et al.* 1995] Alexey G Murzin, Steven E Brenner, Tim Hubbard and Cyrus Chothia. *SCOP: a structural classification of proteins database for the investigation of sequences and structures.* Journal of molecular biology, vol. 247, no. 4, pages 536–540, 1995. (Cited on page 59.)

[Negron & Keating 2013] Christopher Negron and Amy E Keating. *Multi-state protein design using CLEVER and CLASSY.* In Methods in enzymology, volume 523, pages 171–190. Elsevier, 2013. (Cited on page 114.)

[Negron & Keating 2014] Christopher Negron and Amy E Keating. *A set of computationally designed orthogonal antiparallel homodimers that expands the synthetic coiled-coil toolkit.* Journal of the American Chemical Society, vol. 136, no. 47, pages 16544–16556, 2014. (Cited on page 114.)

[Noguchi *et al.* 2019] Hiroki Noguchi, Christine Addy, David Simoncini, Staf Wouters, Bram Mylemans, Luc Van Meervelt, Thomas Schiex, Kam YJ Zhang, Jeremy RH Tame and Arnout RD Voet. *Computational design of symmetrical eight-bladed β-propeller proteins.* IUCrJ, vol. 6, no. 1, pages 46–55, 2019. (Cited on page 72.)

[Ollikainen & Kortemme 2013] Noah Ollikainen and Tanja Kortemme. *Computational protein design quantifies structural constraints on amino acid covariation.* PLoS Comput Biol, vol. 9, no. 11, page e1003313, 2013. (Cited on pages 69, 99 and 148.)

[Otten *et al.* 2012] Lars Otten, Alexander Ihler, Kalev Kask and Rina Dechter. *Winning the pascal 2011 map challenge with enhanced and/or branch-and-bound.* In IN NIPS WORKSHOP DISCML. Citeseer, 2012. (Cited on page 71.)

[Ouali *et al.* 2020] Abdelkader Ouali, David Allouche, Simon de Givry, Samir Loudni, Yahia Lebbah, Lakhdar Loukil and Patrice Boizumault. *Variable neighborhood search for graphical model energy minimization.* Artificial Intelligence, vol. 278, page 103194, 2020. (Cited on page 50.)

[Papadimitriou 1994] Christos H. Papadimitriou. Computational complexity. Addison-Wesley, 1994. (Cited on page 15.)

[Park *et al.* 2016] Hahnbeom Park, Philip Bradley, Per Greisen Jr, Yuan Liu, Vikram Khipple Mulligan, David E Kim, David Baker and Frank DiMaio. *Simultaneous optimization of biomolecular energy functions on features from small molecules and macromolecules.* Journal of

chemical theory and computation, vol. 12, no. 12, pages 6201–6212, 2016. (Cited on pages 100, 131 and 148.)

[Pearl 1988] Judea Pearl. Probabilistic reasoning in intelligent systems: Networks of plausible inference. Morgan Kaufmann, 1988. (Cited on pages 23, 30 and 32.)

[Perrett 1991] M Perrett. *Using constraint logic programming techniques in container port planning.* ICL technical journal, vol. 7, no. 3, pages 537–545, 1991. (Cited on page 6.)

[Pesant 2004] Gilles Pesant. *A regular language membership constraint for finite sequences of variables.* In International conference on principles and practice of constraint programming, pages 482–495. Springer, 2004. (Cited on pages 84, 85, 87 and 145.)

[Petit & Trapp 2015] Thierry Petit and Andrew C Trapp. *Finding diverse solutions of high quality to constraint optimization problems.* In Twenty-Fourth International Joint Conference on Artificial Intelligence, 2015. (Cited on page 81.)

[Pierce & Winfree 2002] Niles A Pierce and Erik Winfree. *Protein design is NP-hard.* Protein engineering, vol. 15, no. 10, pages 779–782, 2002. (Cited on pages 65, 115 and 151.)

[Pierce *et al.* 2000] Niles A Pierce, Jan A Spriet, Johan Desmet and Stephen L Mayo. *Conformational splitting: A more powerful criterion for dead-end elimination.* Journal of computational chemistry, vol. 21, no. 11, pages 999–1009, 2000. (Cited on page 66.)

[Pisinger & Ropke 2010] David Pisinger and Stefan Ropke. *Large neighborhood search.* In Handbook of metaheuristics, pages 399–419. Springer, 2010. (Cited on page 50.)

[Pohl 1970] Ira Pohl. *Heuristic search viewed as path finding in a graph.* Artificial intelligence, vol. 1, no. 3-4, pages 193–204, 1970. (Cited on pages 28 and 105.)

[Prasad *et al.* 2014] Adarsh Prasad, Stefanie Jegelka and Dhruv Batra. *Submodular meets structured: Finding diverse subsets in exponentially-large structured item sets.* In Advances in Neural Information Processing Systems, pages 2645–2653, 2014. (Cited on page 81.)

[Roberts *et al.* 2012] Kyle E Roberts, Patrick R Cushing, Prisca Boisguerin, Dean R Madden and Bruce R Donald. *Computational design of a*

PDZ domain peptide inhibitor that rescues CFTR activity. PLoS Comput Biol, vol. 8, no. 4, page e1002477, 2012. (Cited on page 72.)

[Romero & Arnold 2009] Philip A Romero and Frances H Arnold. *Exploring protein fitness landscapes by directed evolution.* Nature reviews Molecular cell biology, vol. 10, no. 12, pages 866–876, 2009. (Cited on page 61.)

[Rosenfeld *et al.* 1976] Azriel Rosenfeld, Robert A Hummel and Steven W Zucker. *Scene labeling by relaxation operations.* IEEE Transactions on Systems, Man, and Cybernetics, no. 6, pages 420–433, 1976. (Cited on pages 19 and 24.)

[Rossi *et al.* 2006] F. Rossi, P. van Beek and T. Walsh, editors. Handbook of constraint programming. Elsevier, 2006. (Cited on page 141.)

[Röthlisberger *et al.* 2008] Daniela Röthlisberger, Olga Khersonsky, Andrew M Wollacott, Lin Jiang, Jason DeChancie, Jamie Betker, Jasmine L Gallaher, Eric A Althoff, Alexandre Zanghellini, Orly Dym*et al. Kemp elimination catalysts by computational enzyme design.* Nature, vol. 453, no. 7192, pages 190–195, 2008. (Cited on page 72.)

[Ruffini *et al.* 2019a] Manon Ruffini, Jelena Vucinic, Simon de Givry, George Katsirelos, Sophie Barbe and Thomas Schiex. *Guaranteed Diversity & Quality for the Weighted CSP.* In 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), pages 18–25. IEEE, 2019. (Cited on pages 3, 76, 139 and 144.)

[Ruffini *et al.* 2019b] Manon Ruffini, Jelena Vucinic, Simon de Givry, Georges Katsirelos, Sophie Barbe and Thomas Schiex. *Qualité et diversité garanties dans les réseaux de fonctions de coût.* In Journées Francophones de Programmation par Contraintes, pages 143–150, 2019. (Cited on pages 3, 139 and 144.)

[Samish 2017] Ilan Samish. Computational protein design. Springer, 2017. (Cited on pages 63, 64 and 73.)

[Schiex *et al.* 1995] Thomas Schiex, Helene Fargier, Gerard Verfaillie*et al. Valued constraint satisfaction problems: Hard and easy problems.* IJCAI (1), vol. 95, pages 631–639, 1995. (Cited on pages 19 and 22.)

[Schiex 1992] Thomas Schiex. *Possibilistic constraint satisfaction problems or "How to handle soft constraints?".* In Uncertainty in Artificial Intelligence, pages 268–275. Elsevier, 1992. (Cited on page 19.)

[Schiex 2000] Thomas Schiex. *Arc consistency for soft constraints.* In International Conference on Principles and Practice of Constraint Programming, pages 411–425. Springer, 2000. (Cited on pages 33, 34 and 141.)

[Schlesinger 2007] Dmitrij Schlesinger. *Exact solution of permuted submodular minsum problems.* In International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition, pages 28–38. Springer, 2007. (Cited on page 46.)

[Senior *et al.* 2020] Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, Alex Bridgland*et al. Improved protein structure prediction using potentials from deep learning.* Nature, vol. 577, no. 7792, pages 706–710, 2020. (Cited on page 61.)

[Shafer & Shenoy 1988] G Shafer and PP Shenoy. *Local Computations in Hypertrees (Working Paper 201).* School of Business, University of Kansas, Lawrence, KS, USA, 1988. (Cited on page 29.)

[Shapiro & Haralick 1981] Linda G Shapiro and Robert M Haralick. *Structural descriptions and inexact matching.* IEEE transactions on pattern analysis and machine intelligence, no. 5, pages 504–519, 1981. (Cited on page 19.)

[Shapovalov & Dunbrack Jr 2011] 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, vol. 19, no. 6, pages 844–858, 2011. (Cited on pages 63, 100, 121, 131 and 148.)

[Shaw 1998] Paul Shaw. *Using constraint programming and local search methods to solve vehicle routing problems.* In International conference on principles and practice of constraint programming, pages 417–431. Springer, 1998. (Cited on page 50.)

[Siegel *et al.* 2010] Justin B Siegel, Alexandre Zanghellini, Helena M Lovick, Gert Kiss, Abigail R Lambert, Jennifer L St Clair, Jasmine L Gallaher, Donald Hilvert, Michael H Gelb, Barry L Stoddard*et al. Computational design of an enzyme catalyst for a stereoselective bimolecular Diels-Alder reaction.* Science, vol. 329, no. 5989, pages 309–313, 2010. (Cited on page 72.)

[Silberstein *et al.* 2013] Mark Silberstein, Omer Weissbrod, Lars Otten, Anna Tzemach, Andrei Anisenia, Oren Shtark, Dvir Tuberg, Eddie Galfrin, Irena Gannon, Adel Shalata*et al. A system for exact*

and approximate genetic linkage analysis of SNP data in large pedigrees. Bioinformatics, vol. 29, no. 2, pages 197–205, 2013. (Cited on page 71.)

[Simoncini *et al.* 2015] David Simoncini, David Allouche, Simon de Givry, Céline Delmas, Sophie Barbe and Thomas Schiex. *Guaranteed discrete energy optimization on large protein design problems.* Journal of chemical theory and computation, vol. 11, no. 12, pages 5980–5989, 2015. (Cited on pages 71, 99, 100, 108 and 148.)

[Sontag *et al.* 2012] David Sontag, Talya Meltzer, Amir Globerson, Tommi S Jaakkola and Yair Weiss. *Tightening LP relaxations for MAP using message passing.* arXiv preprint arXiv:1206.3288, 2012. (Cited on page 71.)

[Traoré *et al.* 2013] 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, vol. 29, no. 17, pages 2129–2136, 2013. (Cited on pages 69 and 71.)

[Traoré *et al.* 2016] Seydou Traoré, Kyle E Roberts, David Allouche, Bruce R Donald, Isabelle André, Thomas Schiex and Sophie Barbe. *Fast search algorithms for computational protein design.* Journal of computational chemistry, vol. 37, no. 12, pages 1048–1058, 2016. (Cited on pages 69, 70 and 120.)

[Traoré *et al.* 2017] Seydou Traoré, David Allouche, Isabelle André, Thomas Schiex and Sophie Barbe. *Deterministic Search Methods for Computational Protein Design.* In Computational Protein Design, pages 107–123. Springer, 2017. (Cited on page 69.)

[Trösser *et al.* 2020] Fulya Trösser, Simon de Givry and George Katsirelos. *Relaxation-Aware Heuristics for Exact Optimization in Graphical Models.* In Emmanuel Hebrard and Nysret Musliu, editors, Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pages 475–491, Cham, 2020. Springer International Publishing. (Cited on page 50.)

[Valiant 1979] Leslie G Valiant. *The complexity of computing the permanent.* Theoretical computer science, vol. 8, no. 2, pages 189–201, 1979. (Cited on pages 18 and 23.)

[Van Laarhoven & Aarts 1987] Peter JM Van Laarhoven and Emile HL Aarts. *Simulated annealing.* In Simulated annealing: Theory and applications, pages 7–15. Springer, 1987. (Cited on page 65.)

[Verfaillie *et al.* 1996] Gérard Verfaillie, Michel Lemaître and Thomas Schiex. *Russian doll search for solving constraint optimization problems.* In AAAI/IAAI, Vol. 1, pages 181–187, 1996. (Cited on page 28.)

[Verges *et al.* 2015] Alizée Verges, Emmanuelle Cambon, Sophie Barbe, Stéphane Salamone, Yann Le Guen, Claire Moulis, Laurence A Mulard, Magali Remaud-Siméon and Isabelle André. *Computer-aided engineering of a transglycosylase for the glucosylation of an unnatural disaccharide of relevance for bacterial antigen synthesis.* ACS Catalysis, vol. 5, no. 2, pages 1186–1198, 2015. (Cited on page 72.)

[Vucinic *et al.* 2020] Jelena Vucinic, David Simoncini, Manon Ruffini, Sophie Barbe and Thomas Schiex. *Positive multistate protein design.* Bioinformatics, vol. 36, no. 1, pages 122–130, 2020. (Cited on pages 3, 112, 114, 120, 123 and 139.)

[Wainwright *et al.* 2002] Martin Wainwright, Tommi Jaakkola and Alan Willsky. *Tree-based reparameterization analysis of belief propagation and related algorithms for approximate inference on graphs with cycles.* In Proceedings IEEE International Symposium on Information Theory,, page 113. IEEE, 2002. (Cited on page 33.)

[Waltz 1975] David Waltz. *Understanding line drawings of scenes with shadows.* In The psychology of computer vision. Citeseer, 1975. (Cited on page 6.)

[Yanover *et al.* 2007] Chen Yanover, Menachem Fromer and Julia M Shifman. *Dead-end elimination for multistate protein design.* Journal of computational chemistry, vol. 28, no. 13, pages 2122–2129, 2007. (Cited on page 115.)

[Yue & Dill 1992] Kaizhi Yue and Ken A Dill. *Inverse protein folding problem: designing polymer sequences.* Proceedings of the National Academy of Sciences, vol. 89, no. 9, pages 4163–4167, 1992. (Cited on page 63.)

[Zadeh 1999] Lotfi A Zadeh. *Fuzzy logic= computing with words.* In Computing with Words in Information/Intelligent Systems 1, pages 3–23. Springer, 1999. (Cited on page 21.)

[Zhang & Yap 2001] Yuanlin Zhang and Roland HC Yap. *Making AC-3 an optimal algorithm.* In IJCAI, volume 1, pages 316–321, 2001. (Cited on page 13.)

[Zytnicki *et al.* 2006] Matthias Zytnicki, Christine Gaspin and Thomas Schiex. *A new local consistency for weighted CSP dedicated to long domains.* In Proceedings of the 2006 ACM symposium on Applied computing, pages 394–398, 2006. (Cited on page 39.)

Title: Models and Algorithms for Computational Protein Design

Abstract: Weighted constraint programming concisely represents and answers queries on a function of many discrete variables, expressed as a combination of simple functions. This framework can express a wide range of real-world problems, including image labelling, scheduling, resource allocation, etc.
Computational protein design aims at automatically predicting a new sequence of amino acids that will fold into a protein with desired properties and functionalities. This problem can be expressed as a weighted constraint satisfaction problem that minimizes the sequence pairwise energy function on the target rigid backbone. This model being an approximation of the intractable real design problem, the optimum sequence might not answer the biologist needs.
This thesis first contribution is the generation of a set of diverse, good quality solutions, instead of the single minimum energy conformation. The guaranteed diversity and quality of the predictions improve the likelihood of finding a working protein. To that end, a diversity encoding cost function was developed, along with an iterative greedy strategy for the production of diverse sets of good quality sequences. Application to CPD instances showed that predicted sequence quality is improved thanks to diversity.
Besides, the rigid backbone assumption ignores protein flexibility. Multi-state design considers several backbone states simultaneously, therefore retrieving flexibility information. The next step is the inclusion of both desired and undesired states that enables the enhancement of a protein specificity, while preserving its stability. This negative design is harder than the NP-complete positive design problem. It requires a second optimization level. The second part of this work presents the adaptation of the branch-and-bound search and lower bound computation to bilevel optimization.
Keywords: combinatorial optimization, weighted constraint network, local consistency, diversity, bilevel optimization, protein design

---

Titre: Models and Algorithms for Computational Protein Design

Résumé: La programmation par contraintes pondérée permet la représentation concise et la résolution de requêtes sur une fonction définie sur un grand nombre de variables discrètes, exprimée comme une combinaison de fonctions simples. Ce cadre correspond à une large palette de problèmes concrets, comme l'annotation d'images, la planification, l'allocation de ressources, etc.
Le design computationnel de protéines vise à prédire automatiquement des séquences d'acides aminés, qui se replient en de nouvelles protéines aux propriétés et fonctionnalités voulues. Ce problème s'exprime comme un problème de satisfaction de contraintes pondéré, dont le but est la minimisation d'une fonction d'énergie binaire sur la séquence pliée sur un squelette rigide donné. Ce modèle n'est qu'une approximation du problème de design réel, qui est insoluble, et la séquence optimale ne répond pas forcément aux besoins des biologistes.
La première contribution de cette thèse est la génération d'un ensemble de solutions, à la fois diverses et de bonne qualité, à la place de la seule conformation d'énergie minimale. La diversité et la qualité des prédictions sont garanties, et la probabilité de trouver une protéine qui fonctionne est améliorée. Pour cela, j'ai développé une fonction de coût qui encode la diversité, ainsi qu'une stratégie gloutonne pour la production d'un ensemble de séquences diverses de basse énergie. L'application à des instances de design montre que la qualité des séquences prédites est renforcée grâce à la diversité.
Par ailleurs, l'hypothèse de rigidité du backbone néglige la flexibilité de la protéine. Le design multi-états prend en compte plusieurs états du squelette, retrouvant ainsi de l'information de flexibilité. L'étape qui suit est la considération d'états désirés et non-désirés, ce qui permet d'améliorer la spécificité d'une protéine, sans perte de stabilité. Ce design négatif est plus complexe que le design positif, qui est déjà NP-complet: il requiert un second niveau d'optimisation. La seconde partie de mes travaux porte sur l'adaptation de la recherche en branch-and-bound et du calcul du minorant, pour l'optimisation bi-niveau.
Mots-clés: optimisation combinatoire, réseau de contraintes pondérées, cohérences locales, diversité, optimisation bi-niveau, design de protéines