THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée et soutenue le *18/12/2017* **par :**

CLÉMENT VIRICEL

Contributions au développement d'outils computationnels de design de protéines : méthodes et algorithmes de comptage avec garantie

JURY

FRÉDÉRIC CAZALS	Directeur de recherche INRIA	Rapporteur
CHRISTOPHE LECOUTRE	Professeur université d'Artois	Rapporteur
MARTIN COOPER	Professeur université de Toulouse	Examinateur
RAPHAËL GUEROIS	Directeur de recherche CEA	Examinateur
ISABELLE ANDRÉ	Directrice de recherche CNRS	Membre invité
THOMAS SCHIEX	Directeur de recherche INRA	Directeur de thèse
SOPHIE BARBE	Chargée de recherche INRA	Directrice de thèse

École doctorale et spécialité :

MITT : Domaine Mathématiques : Mathématiques appliquées

Unité de Recherche :

Mathématiques et Informatique Appliquées de Toulouse

Directeur(s) de Thèse :

Thomas Schiex et *Sophie Barbe*

Rapporteurs :

Christophe Lecoutre et *Frédéric Cazals*

Remerciements

Je voudrais d'abord remercier l'Institut National de la Recherche Agronomique et la région Occitanie pour m'avoir financé pendant les trois années de cette thèse. Merci pour la confiance accordée et pour avoir rendu possible cette extraordinaire expérience.

Je souhaite aussi exprimer mes remerciements à mes directeurs de thèse Thomas et Sophie pour m'avoir guidés tout du long de cette expérience. Ils ont su faire preuve de patience à mon égard et ils m'ont appris énormément de choses. Pour cela, je leur suis infiniment reconnaissant.

Je tiens à remercier mes deux équipes, le MIA et le LISBP qui m'ont toujours aider quand j'en avais, ou pas, besoin. Merci à Léonard de n'avoir été personne, je suis content de t'aider à devenir quelqu'un. Merci à B2OPS, Popo, Marlich, Clav, Moumou, Angie et Alex de m'avoir supporté, moi et mes "quelques" excentricités, ce qui, avouons-le, n'est pas mince à faire. Merci à Xavier, grâce à toi, je sais construire un tonneau, faire de la mozzarella (en fait c'était Polo ça), m'occuper d'un astéroïde, ce qu'il y a dans les profondeurs océaniques, comment fonctionne un marteau pilon et j'en passe. Merci à Paul, de s'être souvenu de moi et de m'avoir accepter en tant que colocataire. Merci aux fous rires qu'on s'est tapé tous les trois, à la mal bouffe qu'on a black-listée et à nos discussions (parfois) pleine de sens. merci de corriger mon orthographe en espérant qu'un jour je ne me tromperaiS plus.

Merci à mes amis, Flofli, Fanfan, Marguiroux, Cancan, Simaon, Soso, Val, Yo, pour être là pour moi, pour nos soirées, pour tout ce que l'on a vécu et vivra ensemble, pour être ma team, ma seconde famille. Merci à Loulou, d'être aussi... lui même ! Grâce à toi, ma vie est quand même vachement plus marrante. Merci aux PewPewPew, qui se reconnaîtrons, de m'avoir détourné de la thèse pendant nos innombrables débats et parties de gaming (bande de feeder). Merci à Martin Gavidia pour m'avoir donner un minimum de style en Hip-Hop. Merci à Marty et Olive pour toutes nos péripéties passées et futures ! Pour avoir vécu avec moi pendant des années, vous serez toujours mes colocs (ce qui c'est passé en Belgique, restera en Belgique...). Merci à Éliane, de m'avoir passée ses cours de Licence quand je n'y allais pas ! Merci à Laura. Heureusement que tu es là pour adoucir un peu mes fureurs et écouter mes problèmes.

Merci à ma famille, de me supporter quoi qu'il arrive, de me soutenir quoi que je dise ou fasse. Merci à mes frères, Firas et Norman. Vous me montrez ce que je dois faire, qui je dois être. Il y a tellement à dire pour vous deux, sans vous,

je ne serais sûrement pas arrivé jusqu'ici. Merci à Damien, ma moitié, on partage tellement de choses que je n'ai pas assez de mots. Je ne te lâcherai pas, jamais. Merci à mes parents, à mon frère, à ma sœur, à leurs moitiés et leurs enfants, à mes cousins, bref, merci à toute ma famille pour m'apporter autant d'amour.

Merci à Kekette. Sans toi, je serai perdu. Tu es celle qui a le plus comptée durant ces deux dernières années. Merci de m'avoir épaulé, rassuré, supporté, guéri et chéri.

Merci à tous ceux que j'ai oublié mais qui ont été là.

À mes parents, ma grand-mère et mon arrière grand mère,

Après presque 100 ans de vie, je me rend compte que les gens ne savent plus discuter.

Odorizzi Josette, 1916 - aujourd'hui

Table des matières

Introduction

> "Je rêve d'un jour où l'égoïsme ne régnera plus dans les sciences, où on s'associera pour étudier, au lieu d'envoyer aux académiciens des plis cachetés, on s'empressera de publier ses moindres observations pour peu qu'elles soient nouvelles, et on ajoutera : je ne sais pas le reste."
>
> *Évariste Galois (1811-1832)*

Les recherches menées en biologie moléculaire, biochimie et biologie structurale visent à mieux comprendre les mécanismes moléculaires du vivant. Parmi les molécules des organismes vivants, les protéines tiennent un rôle central. Le fonctionnement des protéines dépend de leur structure tridimensionnelle (3D) et de leurs interactions avec différents partenaires moléculaires. Grâce à des processus de sélection, la nature a façonné une étonnante gamme de protéines dotées de diverses fonctions : catalyse, signalisation, reconnaissance, régulation, compartimentation, réparation, *etc.* Cependant, en dépit de cette pléthore de fonctionnalités, la demande en protéines dotées de fonctions et de propriétés intéressantes pour de nouvelles applications en biotechnologie, biologie de synthèse, chimie verte, médecine et nanotechnologie ne cesse de croître.

Les technologies d'ingénierie de protéines offrent des possibilités pour la construction sur mesure de protéines pourvues de fonctions et de propriétés adaptées aux exigences applicatives. Les méthodes computationnelles sont de plus en plus au cœur des stratégies d'ingénierie de protéines pour prédire les modifications moléculaires (mutations) à apporter au sein des protéines afin de leur conférer les propriétés et/ou fonctions recherchées. De telles stratégies raisonnées d'ingénierie de protéines visent à réduire les efforts expérimentaux et les coûts associés tout en augmentant la probabilité de concevoir la protéine d'intérêt. En particulier, des méthodes mettant à profit les liens étroits existants entre la fonction et la structure d'une protéine ont été développées pour pré-filtrer les combinaisons de mutations les plus pertinentes vis à vis des propriétés recherchées et ainsi réduire les tests expérimentaux à un petit nombre de protéines. Malgré les progrès réalisés dans le domaine du *Design Computationnel de Protéines* [1–7], de nouveaux algorithmes, méthodes et outils computationnels sont nécessaires pour évaluer plus précisément l'impact des mutations sur la propriété recherchée afin d'améliorer ainsi la fiabilité des prédictions vis à vis des modifications moléculaires à apporter au sein des protéines. De telles avancées sont essentielles pour accélérer la conception de protéines ou d'assemblages (macro)moléculaires adaptés à leur utilisation dans des procédés durables et compétitifs d'intérêt pour tout un spectre de secteurs (mé-

dical, cosmétique, environnemental, bio-énergie, agroalimentaire, pharmaceutique, *etc.*). En particulier, des méthodes capables de prédire et pré-filtrer les mutations les plus prometteuses pour optimiser l'affinité de liaison d'une protéine avec un partenaire moléculaire donné (autre protéine, petite molécule, *etc.*) sont d'intérêt majeur pour diverses applications. Ces approches devraient également permettre d'améliorer notre compréhension des mécanismes de reconnaissance moléculaire.

Ma thèse se situe au carrefour entre la biologie structurale computationnelle, les mathématiques et l'informatique et porte sur le développement de nouvelles méthodes computationnelles afin d'estimer l'affinité de liaison intermoléculaire pour guider la conception de protéines à affinité améliorée vis à vis d'un ligand d'intérêt. En particulier, les méthodes computationnelles développées durant cette thèse se basent sur des outils d'optimisations pour trouver la conformation d'énergie minimale ainsi que de nouveaux algorithmes de comptage (ou d'intégration discrète) pour calculer la fonction de partition de systèmes macromoléculaires.

La première partie regroupe une introduction et un état de l'art sur les modèles graphiques, la modélisation et le design computationnel des protéines. Le chapitre 1 introduit les modèles graphiques, un formalisme utilisé pour la modélisation de nombreux problèmes concrets. Il y est question d'optimisation discrète et de calcul de la fonction de partition, et nous présentons différentes méthodes visant à résoudre ces problèmes. Le chapitre 2 dépeint un état de l'art des différentes méthodes de calcul de la fonction de partition, regroupées selon le type de garantie qu'elles fournissent (sans garantie, garantie asymptotique, probabiliste ou déterministe). Le chapitre 3 est une introduction générale sur la structure des protéines et les concepts de base du design computationnel de protéines ainsi qu'une présentation de méthodes estimant l'affinité de liaison entre une protéine et un ligand et plus particulièrement l'impact de mutations sur cette affinité de liaison.

La seconde partie présente les contributions de cette thèse pour l'algorithmique de comptage dans les modèles graphiques et le calcul de la fonction de partition de modèles graphiques issus d'interaction protéine-protéine. L'application de ces méthodes pour la prédiction de l'affinité de liaison de complexes protéine-protéine et de l'impact de mutations sur cette affinité est décrite dans cette partie.

Le chapitre 4 décrit trois algorithmes de calcul de la fonction de partition. Le premier algorithme, Z^*_ε, est un algorithme fournissant la fonction de partition avec une garantie ε fixée. Il repose sur l'élagage des quantités négligeables de potentiels durant la recherche en profondeur. Z^*_ε a été comparé à d'autres méthodes du domaine dans [8,9]. Le second algorithme, `#HBFS`, est une adaptation d'une méthode de recherche arborescente développée pour l'optimisation, `HBFS` [10], combinant deux stratégies de recherche *Depth First Search* et *Best First Search*. Il fournit la fonction de partition avec une garantie ε à n'importe quel moment de la re-

cherche (algorithme dit *anytime*) et croissante au cours du temps. Il s'est montré plus performant que son prédécesseur Z_ε^*. Le dernier, `BTDZ`, est un algorithme exact s'appuyant sur une décomposition arborescente pour identifier les parties conditionnellement indépendantes du modèle graphique. Il mémorise et réutilise les contributions à la fonction de partition de ses parties conditionnellement indépendantes, économisant ainsi du temps de calcul sur les modèles graphiques possédant une petite largeur d'arbre. `BTDZ` s'est révélé efficace pour calculer la fonction de partition sur des instances issues d'interactions de protéines appelées "superhélices".

Le chapitre 5 comporte la description de notre outil de design computationnel de complexe de protéines, estimant l'affinité de liaison de systèmes macromoléculaires. Notre outil est entièrement automatisé de la modélisation du système à l'estimation de l'affinité de liaison de larges librairies de mutants. Il comprend une première méthode, EasyE, basée sur l'algorithme d'optimisation `HBFS` [10] et une seconde basée, JayZ, sur notre algorithme de comptage Z_ε^* (détaillé dans le chapitre 4), toutes deux appliquées à l'estimation de l'affinité de liaison protéine-protéine. Ensuite, cette partie se focalise sur la mise en œuvre de ces méthodes sur un jeu de donnée de mutants de complexes protéine-protéine pour lesquels des données expérimentales d'affinité de liaison sont disponibles. Les performances de nos méthodes sont comparées à celles de méthodes traditionnellement utilisées en design de protéines. Un article a été soumis à *Bioinformatics* et est actuellement en révision.

Tous les algorithmes et méthodes développés pendant la thèse sont disponibles en *open source*. Enfin le chapitre 6 conclut et présente les perspectives de nos travaux.

Publications

Cost Function Network-based Design of Protein-Protein Interactions : predicting changes in binding affinity
C Viricel, S De Givry, T Schiex, S. Barbe
Bioinformatics
Date de publication : (en révision)

Guaranteed weighted counting for affinity computation : Beyond determinism and structure
C Viricel, D Simoncini, S Barbe, T Schiex
International Conference on Principles and Practice of Constraint Programming p733-750
Springer International Publishing
Date de publication : 2016

Approximate counting with deterministic guarantees for affinity computation
C Viricel, D Simoncini, D Allouche, S De Givry, S Barbe, T Schiex
Modelling, Computation and Optimization in Information Systems and Management Sciences, p165-176
Springer, Cham
Date de publication : 2015

Communications orales

Simoncini D, **Viricel C**, Traoré S, Allouche D, André I, Schiex T, Barbe S.
New methods for Computational Protein Design.
20eme congrès du Groupe de Graphisme et de Modélisation Moléculaire, GGMM 2017,
Reims, France, May 9-11, 2017.

Viricel C, Simoncini D, Barbe S, Schiex T.
Guaranteed Weighted Counting for Affinity Computation : Beyond Determinism and Structure.
22nd International Conference on Principles and Practice of Constraint Programming,
Toulouse, France. 5-9 September 2016.

Viricel C, Simoncini D, Allouche D, de Givry S, Barbe S, Schiex T.
Approximate Counting with Deterministic Guarantees for Binding Affinity Com-

putation.
5 th workshop on Algorithmic issues for Inference in Graphical Models (AIGM), Paris, France, Sept 28, 2015.

Viricel C, Simoncini D, Allouche D, de Givry S, Barbe S, Schiex T.
Approximate Counting with Deterministic Guarantees for Affinity Computation.
11th workshop on Constraint Based Methods for Bioinformatics held in conjunction with CP 2015,
Cork, Ireland, August 31, 2015.

Allouche D, **Viricel C**, Traoré S, Simoncini D, de Givry S, Katsirelos G, André I, Schiex T, Barbe S.
Computational Enzyme Design through deterministic optimization and counting.
3DSIG : Structural Bioinformatics and Computational Biophysics,
Dublin, Ireland, July 10-11, 2015.

Viricel C, Simoncini D, Allouche D, de Givry S, Barbe S, Schiex T.
Approximate Counting with Deterministic Guarantees for Affinity Computation.
Modelling, Computation and Optimization in Information Systems and Management Sciences, MCO 2015,
Metz, France, May 11-13, 2015.

Communications posters

Simoncini D, **Viricel C**, Charpentier A, Traoré S, de Givry S, Cortés J, André I, Allouche D, Schiex T, Barbe S.
Computational Protein Design to accelerate the conception of fine-tuned biocatalysts.
Enzyme Engineering XXIV Conference,
Toulouse, France, 24-28 September 2017.

Allouche D, **Viricel C**, Traoré S, Simoncini D, de Givry S, Katsirelos G, André I, Schiex T, Barbe S.
Computational enzyme design through deterministic search and counting methods.
3DSIG : Structural Bioinformatics and Computational Biophysics,
Dublin, Ireland, July 10-11, 2015.

Viricel C, Traoré S, Simoncini D, Allouche D, de Givry, André I, Schiex T, Barbe S.
Computational enzyme design through deterministic search and counting methods.
Groupe de Graphisme et de Modélisation Moléculaire, GGMM 2015,
Sète, France, May 25-28, 2015.

Première partie

Modèles graphiques et protéines

Chapitre 1

Les modèles graphiques

> "Without mathematics, there's nothing you can do. Everything around you is mathematics. Everything around you is numbers."
>
> *Shakuntala Devi (1929-2013)*

1.1 Formalismes et définitions

Selon Kevin Murphy, les modèles graphiques sont le fruit du mariage entre la théorie des probabilités et la théorie des graphes [11]. En effet, un modèle graphique [12–14] est un vecteur (les vecteurs seront notés en gras) de variables aléatoires $\mathbf{X} = (X_1, \cdots, X_n)$ reliées entre elles par un ensemble $\mathscr{F}$ de fonctions locales f portant chacune sur un vecteur de variables, et dont la structure d'interaction peut se représenter sous la forme d'un graphe.

Dans un modèle graphique, chacune des n variables X_i (les variables seront notées en majuscules) pourra prendre des valeurs x_i (les valeurs seront notées en minuscules). Une valeur x_i est *affectée* à une variable X_i lorsque $X_i = x_i$. L'ensemble des valeurs possibles de X_i est appelé son domaine et il est désigné par d_i. Une affectation complète $\mathbf{X} = (X_1 = x_1, \cdots, X_n = x_n)$ est notée $\mathbf{x} = (x_1, \cdots, x_n)$. Une affection partielle du vecteur $\mathbf{X}$ est notée $\mathbf{x}_q = (x_1, \cdots, x_q)$ avec $q < n$.

Une fonction $f \in \mathscr{F}$ fait correspondre un vecteur de valeurs issues des d_i à un élément d'un ensemble $\mathbb{E}$ muni des opérations $\bigotimes$ et $\bigoplus$ d'élément neutre respectif 1 et 0. La **portée** S (*scope*) d'une fonction f_S est un sous-ensemble d'indices inclus dans $\{1, \cdots, n\}$ et définit l'ensemble des variables sur lequel f_S porte. L'**arité** de

f est le cardinal de la portée. Par exemple, une fonction f_{ij} d'arité 2 (aussi appelée fonction binaire) est définie sur $d_i \times d_j$. L'argument de la fonction f_S noté $\mathbf{x}[\mathbf{S}]$ est la restriction (ou projection) du vecteur $\mathbf{x}$ sur S. Ainsi, si $S = \{1, 2\}$ alors $\mathbf{x}[\mathbf{S}] = (x_1, x_2)$. Un modèle graphique est formellement défini par :

Définition 1.1.1. Un modèle graphique *est un tuple* $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \mathscr{F}, \bigotimes \rangle$ *où :*

- $\boldsymbol{X} = (X_1, \ldots, X_n)$ *est un vecteur de* n *variables.*
- $\boldsymbol{D} = (d_1, \ldots, d_n)$ *est le vecteur des* n *domaines respectifs.*
- $\mathscr{F}$ *est l'ensemble des fonctions locales* f_S *chacune ayant une portée* $S \subseteq \{1, \cdots, n\}$ *et définies par* $f_S : D_S \mapsto \mathbb{E}$ *où* $D_S = \prod_{i \in S} d_i$ *dénote le produit cartésien de tous les* d_i *pour* $i \in S$ *et* $\mathbb{E}$ *un ensemble totalement ordonné muni des opérations* $\bigotimes$ *et* $\bigoplus$ *définissant un semi-anneau d'éléments neutres respectifs* 1 *et* 0.
- *L'opérateur de combinaison* $\bigotimes$ *permet de combiner les fonctions locales de* $\mathscr{F}$ *en une fonction jointe*

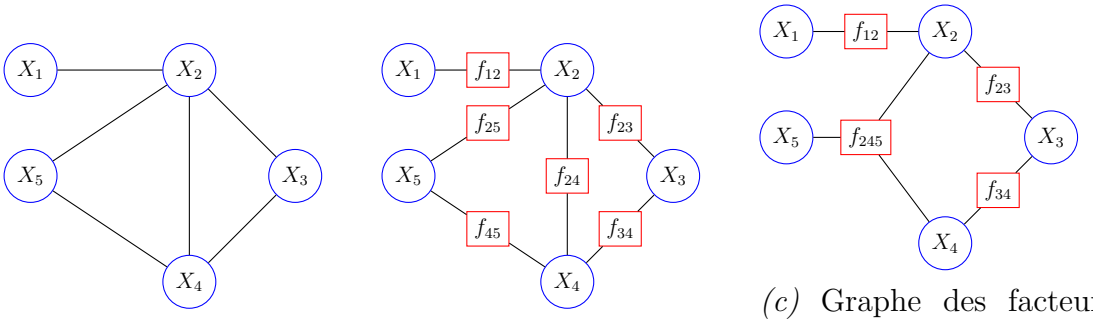
$$F(\boldsymbol{x}) = \bigotimes_{f_S \in \mathscr{F}} f_S(\boldsymbol{x}[\mathbf{S}])$$

pour toute affectation $\boldsymbol{x} \in D_X$.

Le maximum sur la taille des domaines est noté $d = \max_i |d_i|$ *et le nombre de fonctions d'un modèle graphique est noté* $e = |\mathscr{F}|$.

Un modèle graphique peut se représenter sous la forme d'un graphe $G = (V, E)$ avec un ensemble de sommets V et un ensemble d'arêtes E (figure 1.1a). Un sommet est associé à une variable de $\mathbf{X}$ et une arête existe entre deux sommets si et seulement si il existe une fonction f_S dont la portée S contient les indices correspondants aux variables. Le nombre d'arêtes partant d'un sommet s'appelle son degré. Usuellement, les fonctions unaires ne sont pas représentées dans le graphe mais elles peuvent être représentées par une boucle si nécessaire. Le graphe d'un modèle graphique permet de rendre l'interaction entre les variables plus intuitive.

Il existe une représentation plus précise de la structure d'un modèle graphique appelée **graphe des facteurs** [15] (Voir figures 1.1b, 1.1c). Le graphe des facteurs d'un modèle graphique est un graphe bipartie (X, Y, E) où X est un ensemble de sommets représentant les variables, Y un ensemble de sommets représentant les fonctions et E est un ensemble d'arêtes. Une arête est définie entre un sommet variable et un sommet fonction si et seulement si la fonction implique la dite variable. Le graphe des facteurs permet de visualiser sans ambiguïté les portées des fonctions ce que ne permet pas toujours la représentation sous forme de graphe simple (si ce n'est dans le cas d'un modèle graphique avec une portée maximale de deux variables).



(a) Modèle graphique de cinq variables $(X_1, X_2, X_3, X_4, X_5)$.

(b) Graphe des facteurs avec six fonctions binaires $f_{12}, f_{23}, f_{34}, f_{24}, f_{25}, f_{45}$

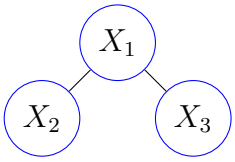
(c) Graphe des facteurs avec une fonction ternaire f_{245} et trois fonctions binaires f_{12}, f_{23}, f_{34}

Fig. 1.1 – Différents graphes G représentant un modèle graphique de 5 variables. Les variables sont les cercles bleus. Pour les graphes des facteurs, les fonctions sont encadrées en rouge.

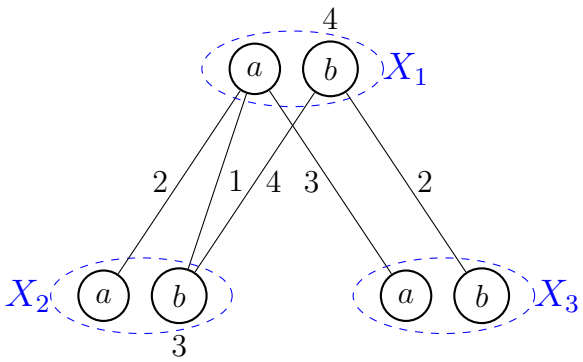
Dans la figure 1.1a, le modèle graphique a cinq variables $\mathbf{X} = (X_1, X_2, X_3, X_4, X_5)$. L'ensemble $\mathscr{F}$ pourra correspondre soit à $\mathscr{F} = \{f_{12}, f_{23}, f_{34}, f_{24}, f_{25}, f_{45}\}$ (cf. figure 1.1b) soit à $\mathscr{F} = \{f_{12}, f_{23}, f_{34}, f_{245}\}$ (figure 1.1c).

Le graphe d'un modèle graphique ne laisse transparaître que les variables et l'existence d'une interaction entre variables ; il ne contient pas l'information sur la forme de l'interaction entre les valeurs des variables. La micro-structure d'un modèle graphique est un "zoom" sur le graphe où les sommets sont les valeurs possibles prises par les variables. Une arête entre deux valeurs est définie si et seulement si il existe une fonction impliquant ces deux valeurs (figure 1.2b). Pour un souci de clarté, nous ne mettrons pas d'exemple de fonction d'arité supérieure à 2. Les grandeurs retournées par les fonctions binaires sont placées sur les arêtes et celles retournées par les fonctions unaires près des valeurs. Par convention, dans cette thèse nous ne représentons pas les arêtes associées à l'élément neutre de $(\mathbb{E}, \bigotimes)$.

Les modèles graphiques couvrent une variété de modèles mathématiques qui dépendent de l'ensemble de définition des fonctions et de l'opérateur de combinaison. Les fonctions peuvent être définies par des disjonctions de variables, ou de leur négation, définies sur un ensemble booléen $\mathbb{B}$ et combinées via l'opérateur logique $\bigotimes = \wedge$ (modèle SAT) ou par des fonctions booléennes arbitraires (modèle CSP) mais peut aussi utiliser des fonctions entières positives combinées par $\bigotimes = \sum$ (réseau de fonctions de coût) ou encore des fonctions réelles positives combinées avec le produit $\bigotimes = \prod$ (réseau bayésien/champ de Markov).



(a) Graphe d'un modèle graphique de trois variables X_1, X_2, X_3



(b) Micro structure d'un modèle graphique de trois variables X_1, X_2, X_3 toutes trois de domaine $\{a, b\}$. L'absence d'arête indique ici une valeur égale à l'élément neutre. Ce sera tout le temps le cas par la suite.

Fig. 1.2 – Modèle graphique et sa micro-structure.

Sur cet ensemble de modèles graphiques, plusieurs types de problème peuvent être considérés : problème de satisfaction d'une formule propositionnelle (SAT), problème de satisfaction de contraintes (CSP), identification du maximum d'*a posteriori* pour les champs de Markov (MAP-MRF), recherche de la solution de coût minimum pour un réseau de fonctions de coût (WCSP). Tous ces problèmes sont des problèmes d'optimisation consistant à minimiser la fonction jointe $F(x)$. Les champs de Markov et les réseaux de fonctions de coûts seront utilisés tout au long de cette thèse et seront donc définis dans ce chapitre.

Champ de Markov

Un champ de Markov [14, 16–18] définit une fonction potentielle sur un vecteur de variables aléatoires comme le produit de fonctions potentielles locales. Une fonction potentielle ϕ d'un champ de Markov peut être n'importe quelle fonction réelle non-négative. La valeur retournée par une fonction potentielle est appelée un potentiel.

Définition 1.1.2. *Un* champ de Markov (*Markov Random Field* MRF) *est un modèle graphique* $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \Phi, \prod \rangle$ *où :*

- $\boldsymbol{X} = (X_1, \ldots, X_n)$ *est un vecteur de* n *variables aléatoires.*
- $\boldsymbol{D} = (d_1, \ldots, d_n)$ *est un vecteur de* n *domaines respectifs.*
- Φ *est un ensemble de fonctions potentielles locales* ϕ_S *ayant chacune une portée* $S \subseteq \{1, \cdots, n\}$ *et définies par* $\phi_S : D_S \mapsto \mathbb{R}^+ \cup \{\infty\}$ *où* D_S *dénote le produit cartésien de tous les* d_i *pour* $i \in S$
- *L'opérateur de combinaison est le produit* $\prod$*, il définit une fonction potentielle jointe :*

$$P(\boldsymbol{x}) = \prod_{\phi_S \in \Phi} \phi_S(\boldsymbol{x}[\mathbf{S}])$$

Définition 1.1.3 (Distribution de probabilité associée)**.** *Après normalisation, un champ de Markov* $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \Phi, \prod \rangle$ *définit une distribution de probabilité jointe* p *:*

$$p(\boldsymbol{x}) = \frac{1}{Z} P(\boldsymbol{x}) = \frac{1}{Z} \prod_{\phi_S \in \Phi} \phi_S(\boldsymbol{x}[\mathbf{S}])$$

$\mathbf{Z}$*, la* ***constante de normalisation****, aussi appelée* ***fonction de partition****, est la somme des potentiels de la fonction jointe* P *sur toutes les affectations* $\boldsymbol{x} \in D_X$*.*

$$Z = \sum_{\boldsymbol{x} \in D_X} P(\boldsymbol{x}) = \sum_{\boldsymbol{x} \in D_X} \prod_{\phi_S \in \Phi} \phi_S(\boldsymbol{x}[\mathbf{S}])$$

Réseau de fonctions de coût

Les réseaux de fonctions de coût [19–21] sont des modèles graphiques représentés sous forme d'un réseau de variables (non aléatoires) connectées entre elles par des fonctions de coût. Les réseaux de fonctions de coût sont très utilisés en intelligence artificielle. Le principe est assez simple : pour chaque affectation de variables, il faudra payer des coûts correspondant aux affectations. Une fonction de coût d'un réseau de fonction de coût peut être n'importe quelle fonction entière non-négative*. La valeur retournée par une fonction de coût c est appelée un coût.

Définition 1.1.4. *Un* réseau de fonctions de coût (*Cost Function Network* CFN) *est un modèle graphique* $\langle \boldsymbol{X}, \boldsymbol{D}, \mathscr{C}, \sum^{\top} \rangle$ *où :*

- $\boldsymbol{X} = (X_1, \ldots, X_n)$ *est un vecteur de* n *variables.*
- $\boldsymbol{D} = (d_1, \ldots, d_n)$ *est un vecteur de* n *domaines respectifs.*
- $\mathscr{C}$ *est un ensemble de fonctions de coût* c_S *chacune ayant une portée* $S \subseteq \{1, \cdots, n\}$ *et définies par* $c_S : D_S \mapsto \{0, ..., \top\}$ *où* D_S *dénote le produit cartésien de tout les* d_i *pour* $i \in S$
- *L'opérateur de combinaison est la somme* $\sum^{\top}$ *bornée par* $\top$ *représentant un coût interdit i.e* $\forall a, b \in \{1, ..., \top\}, a +^{\top} b = \min(\top, a + b)$. *Il définit une fonction de coût jointe :*

$$C(\boldsymbol{x}) = \sum_{c_S \in \mathscr{C}}^{\top} c_S(\boldsymbol{x}[\mathbf{S}])$$

Un réseau de fonctions de coûts définit un problème de satisfaction de contraintes pondéré ou *Weighted Constraint Satisfaction Problem* (WCSP) qui consiste à identifier une affectation complète des variables dont le coût est minimum.

Un opérateur $-^{\top}$ est défini selon la relation suivante :

$$\forall a, b \in \{1, ..., \top\}, a \geq b, a -^{\top} b = \begin{cases} \top \text{ si } a = \top \\ a - b \text{ sinon} \end{cases}$$

Dans la suite du manuscrit, afin d'éviter une surcharge d'écriture, les opérateurs $\sum^{\top}, +^{\top}$ et $-^{\top}$ seront remplacés respectivement par $\sum, +$ et $-$.

On notera que les coûts d'un CFN sont des entiers positifs. De ce fait, le coût d'une affectation croît au fur et à mesure que l'on considère de nouvelles fonctions de coût. Soit $c_{\varnothing}$ la fonction de coût d'arité nulle (de portée vide, une constante). Du fait de la monotonie de la fonction de coûts globale, $c_{\varnothing}$ est un minorant du

∗. Certaines variantes utilisent des valeurs rationnelles, voir [22].

coût $C(\mathbf{x})$ de n'importe quelle affectation $\mathbf{x}$. En effet, le coût d'une affectation se réécrit en :

$$C(\mathbf{x}) = c_\varnothing + \sum_{c_S \in \mathscr{C} \setminus \{\varnothing\}} c_S(\mathbf{x}[\mathbf{S}])$$

Et donc,

$$\forall \mathbf{x} \in D_X, c_\varnothing \leq C(\mathbf{x})$$

Lien entre réseaux de fonctions de coût et champs de Markov

D'après les définitions 1.1.3 et 1.1.4, il est assez facile de reconnaître qu'un CFN et un MRF se ressemblent beaucoup. En utilisant une fonction composée de l'opposé du logarithme des potentiels, il sera possible de transformer l'opérateur produit en somme et les potentiels en coûts. Il est bien sur impossible de trouver une bijection de $\mathbb{R}^+$ dans $\mathbb{N}^+$ sans que Hilbert ne se retourne dans sa tombe. La fonction utilisée sera donc dépendante d'une précision p qui servira à arrondir les réels en entiers (représentation en virgule fixe).

$$\begin{aligned} c_S(\mathbf{x}[\mathbf{S}]) &= \lfloor -\log\left(\frac{\phi_S(\mathbf{x}[\mathbf{S}])}{\max\limits_{\mathbf{y} \in D_S} \phi_S\left(\mathbf{y}[\mathbf{S}]\right)}\right) \times 10^p \rfloor \\ &= \lfloor \left(E_S(\mathbf{x}[\mathbf{S}]) - \min_{\mathbf{y} \in D_S} E_S\left(\mathbf{y}[\mathbf{S}]\right)\right) \times 10^p \rfloor \end{aligned} \tag{1.1}$$

Si un potentiel est égal à 0 alors le coût est fixé à $\top$. Le terme $E(\mathbf{x}) = -\log(\phi(\mathbf{x}))$ est couramment appelé l'**énergie**. Cette correspondance entre CFN et MRF nous permettra d'utiliser les algorithmes existants pour traiter chaque type de modèle graphique. Durant la suite de la thèse, on se permettra donc d'utiliser l'un ou l'autre, sachant qu'ils sont équivalents à la précision p près.

Plus haut, nous avons brièvement énoncé qu'un modèle graphique permet de modéliser un système. Un système est très souvent accompagné d'un problème bien défini. Les problèmes associés aux modèles graphiques sont appelés des **problèmes d'inférence**. Dans la prochaine partie, nous allons définir deux problèmes d'inférence très étudiés : l'**optimisation** et le **comptage**. Nous allons aussi définir les moyens les plus couramment utilisés pour résoudre ce genre de problème.

1.2 Recherche arborescente

Dans un **problème d'optimisation**, il est question de **minimiser ou maximiser** une fonction jointe. Pour un MRF, l'objectif du calcul du maximum

d'*a posteriori* (MAP) (ou `Max-Prod` car $\bigoplus = \max$ et $\bigotimes = \prod$) est de trouver l'affectation $\mathbf{x}^*$ telle que la fonction potentielle (ou la probabilité) jointe $P(\mathbf{x}^*)$ (ou $p(\mathbf{x}^*)$) soit maximale. C'est à dire que l'objectif est de trouver l'affectation $\mathbf{x}^*$ telle que :

$$\mathbf{x}^* = \underset{\mathbf{x} \in D_X}{\operatorname{argmax}} P(\mathbf{x}) = \underset{\mathbf{x} \in D_X}{\operatorname{argmax}} p(\mathbf{x}) = \underset{\mathbf{x} \in D_X}{\operatorname{argmax}} \prod_{\phi_S \in \Phi} \phi_S(\mathbf{x}[\mathbf{S}])$$

De manière équivalente, le problème d'optimisation sur un CFN est de trouver l'affectation $\mathbf{x}^*$ telle que le coût global $C(\mathbf{x})$ soit minimal (aussi connu sous le nom de problème `Min-Sum` car $\bigoplus = \min$ et $\bigotimes = \sum$).

$$\mathbf{x}^* = \underset{\mathbf{x} \in D_X}{\operatorname{argmin}} C(\mathbf{x}) = \underset{\mathbf{x} \in D_X}{\operatorname{argmin}} \sum_{c_S \in \mathscr{C}} c_S(\mathbf{x}[\mathbf{S}])$$

De manière générale, ces problèmes d'optimisation sont **intraitables**, c'est à dire qu'aucun **algorithme déterministe connu ne peut résoudre systématiquement** le problème d'optimisation **en un temps polynomial** en la taille de l'entrée. Ces problèmes sont **NP-difficiles** [23] † et plus précisément décision NP-complets. Ils sont centraux dans de nombreux domaines tels que l'intelligence artificielle [24,25], la vision sur ordinateur [26,27], la planification [28] ou dans notre cas, l'ingénierie de protéines [2,4–7].

Cependant, notre problème d'intérêt est de calculer la fonction de partition, c'est à dire de sommer la valeur de la fonction jointe sur toutes les affectations possibles, il est autrement nommé `Sum-Prod` car $\bigoplus = \sum$ et $\bigotimes = \prod$.

$$Z = \sum_{\mathbf{x} \in D_X} \prod_{\phi_S \in \Phi} \phi_S(\mathbf{x}[\mathbf{S}])$$

Ce problème fait partie de la classe de complexité #P [29], plus haute que NP, qui rassemble des problèmes **de comptage**. Il est complet pour cette classe.

Une possibilité pour résoudre un problème d'optimisation ou de comptage est de représenter les affectations des variables par **un arbre de recherche** (Figure 1.3). Dans un arbre de recherche, chaque nœud correspond à une variable X_i et ses fils correspondent chacun à un choix de valeur x_i pour l'affectation de la variable X_i. La racine de l'arbre correspond à l'état initial du modèle graphique lorsqu'encore aucune variable n'a été affectée. Il est courant d'ordonner la progression de la recherche arborescente avec des heuristiques de choix de variables [30,31].

Un **sous-arbre** est un arbre enraciné en un nœud interne de l'arbre. Une **feuille de l'arbre** est atteinte lorsque toutes les variables ont été affectées. En théorie, les problèmes d'optimisation et de comptage peuvent être résolus de façon brutale.

†. Les classifications sont faites par la théorie de la complexité.

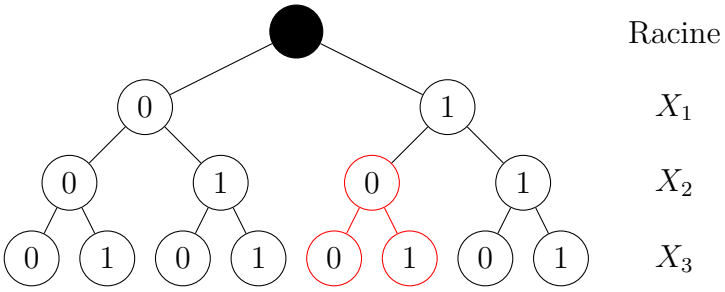


Fig. 1.3 – Arbre de recherche de trois variables X_1, X_2, X_3 de domaine booléen $d_i = \{0, 1\}$. Chaque niveau i de l'arbre correspond à l'affectation d'une variable X_i. Chaque branchement à partir d'un nœud correspond à une affectation de valeur à une variable. Un sous-arbre enraciné au nœud $(x_1 = 1, x_2 = 0)$ est représenté en rouge.

Pour cela, il suffit de développer tout l'arbre afin d'atteindre toutes les feuilles puis d'extraire la solution optimale ou la somme des potentiels de l'énorme espace combinatoire.

Un algorithme de recherche arborescente exploite deux ressources définissant deux types de complexités : le **temps** et l'**espace**. Par exemple, pour un modèle graphique de n variables, explorer de manière arbitraire l'arbre de recherche a une complexité en espace de $O(d^n)$. En pratique, les algorithmes développés permettent de réduire (ou d'équilibrer) ces deux complexités. Il existe plusieurs stratégies pour explorer un arbre de recherche [32]. Dans cette section nous verrons deux méthodes fondamentales d'exploration : la recherche en profondeur d'abord (*Depth First Search* `DFS`) et la recherche en meilleur d'abord (*Best First Search* `BFS`).

Recherche en profondeur d'abord

La recherche en profondeur consiste à d'abord s'enfoncer dans l'arbre puis à remonter (**backtrack**) afin d'explorer les autres branches, les fils d'un nœud sont explorés avant ses frères (figure 1.4). La complexité de `DFS` en temps est de $O(d^n)$ et celle en espace de $O(n)$. Cette méthode d'exploration est peu coûteuse en mémoire et permet d'explorer tout l'arbre de recherche mais elle reste coûteuse en temps. L'exploration de l'arbre peut être améliorée en utilisant des mécanismes additionnels permettant de couper les branches de l'arbre dont on peut prouver qu'elles ne mènent pas à une solution. Il s'agit de l'idée principale de l'algorithme de **séparation-évaluation** (*Branch and Bound* `B&B`). Dans sa version en profondeur

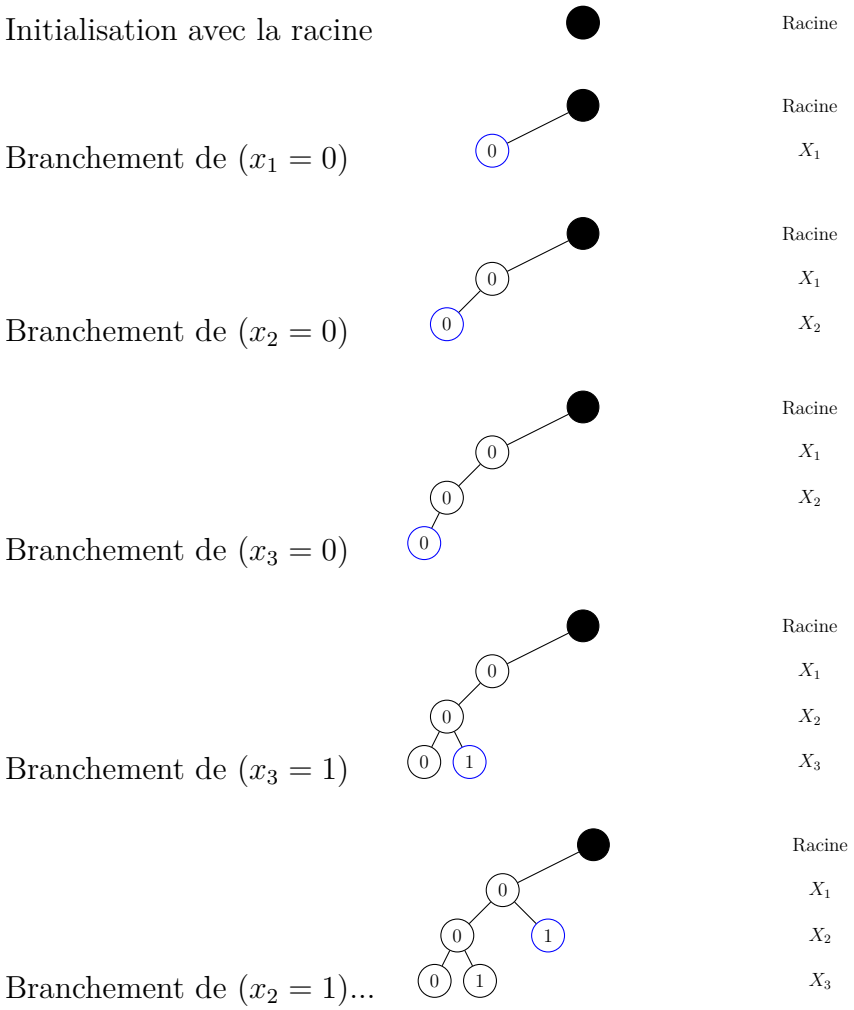


Fig. 1.4 – Recherche de type `DFS`. Chaque étape correspond au branchement d'un nœud (en bleu) et donc à une affectation de valeur à une variable. Les backtracks sont implicites.

d'abord, l'idée est de maintenir le coût k de la meilleure affectation rencontrée jusqu'ici et de le comparer à une estimation h qui minore les coûts de toutes les feuilles présentes dans un sous-arbre. A la racine, le coût minimal k est fixé à $\top$. Supposons que durant la recherche en profondeur, un nœud n ayant pour affectation partielle $\mathbf{x_q}$ soit tel que $h(x_q) > k$. Alors il ne sera pas nécessaire d'explorer le sous-arbre enraciné au nœud n car il est certain que le coût de chacune des feuilles du sous-arbre n'est pas plus petit que k. De plus, si `DFS` s'enfonce dans une branche et qu'une feuille est atteinte alors nécessairement le coût de l'affectation correspondant à la feuille est meilleur que k. Par conséquent, k est remplacé par le coût de la nouvelle feuille. L'algorithme s'arrête lorsque toutes les feuilles ont été explorées ou élaguées (Voir algorithme 1.1).

Algorithme 1.1 : Algorithme de parcours d'arbre `DFS-B&B` pour un problème `Min-Sum` d'un CFN.

```
1  X ← ∅
2  k ← ⊤
3  Fonction DFS-BB(x, X)
4      si X = ∅ alors
5          k ← C(x);
6      sinon
7          Prendre une variable X_i ∈ X;
8          pour x_i ∈ d_i faire
9              x ← x ∪ {x_i};
10             si (h(x) < k) alors
11                 DFS-BB(x, X − {X_i});
```

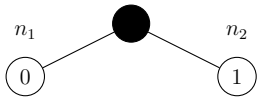
Recherche en meilleur d'abord

Le `BFS` explore l'arbre **en meilleur d'abord**. Il utilise une structure appelée liste de nœuds ouverts ou *open* pour stocker les nœuds de l'arbre restant à explorer ainsi qu'une heuristique lui indiquant quel nœud choisir pour continuer la recherche. Un nœud n_i est retiré de la liste *open* afin de le développer et d'ajouter tous ses fils ($Child(n_i)$) dans la liste de nœuds ouverts. Au départ de l'exploration, la liste de nœuds ouverts ne contient que la racine. La complexité en espace et en temps de `BFS` est en $O(d^n)$.

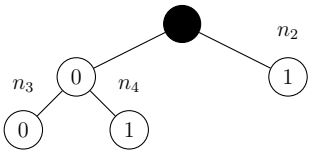
L'avantage de cette méthode est que le choix du nœud à développer peut-être fait de sorte que la première solution retournée soit la solution optimale. Pour

Initialisation à la racine dans la liste : $open = [root]$

Branchement des fils de la racine, ajout de ces fils dans la liste :
$open = [n_1, n_2]$



Branchement des fils de n_1, ajout de ces fils dans la liste :
$open = [n_2, n_3, n_4]$



Branchement des fils de n_2, ajout de ces fils dans la liste :
$open = [n_3, n_4, n_5, n_6]$

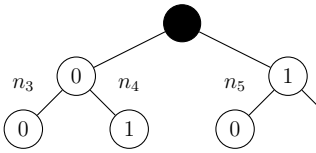


Fig. 1.5 – Recherche de type `BFS`. Chaque étape correspond au choix d'un nœud dans la liste de nœuds ouverts.

cela, l'heuristique doit être admissible, c'est à dire qu'elle doit être minorante. L'algorithme A^* est un algorithme de recherche sur un graphe de type `BFS` qui assure de tomber sur la solution optimale en premier. Il a été développé initialement par Hart et al. [33] en 1968 pour résoudre une tâche de planification représentable via un graphe. Il sera ici appliqué sur le cas plus simple d'un arbre de recherche.

A^* développe un nœud n_i en fonction d'une heuristique $h(n_i)$ qui indiquera la qualité de ce nœud. Cette fonction $h(n_i)$ est la somme de deux autres fonctions $f(n_i)$ et $g(n_i)$ qui représentent respectivement le coût déjà dépensé pour arriver au nœud et une estimation optimiste sur le coût additionnel minimum permettant d'arriver jusqu'à une feuille descendant de n_i. Dans ce cas, la première feuille rencontrée sera assurément la meilleure solution $\mathbf{x}^*$ possible, avec un coût minimal (voir algorithme 1.2).

Algorithme 1.2 : Algorithme de parcours d'arbre A^*.

```
1  O ← (root) ;
2  Fonction A*(O)
3      tant que O ≠ ∅ faire
4          n ← argmin_{n_i ∈ O} h(n_i) ;
5          O ← O − {n} ;
6          si n est une feuille alors
7              retourner x* ;
8          sinon
9              pour n' ∈ Child(n) faire
10                 O ← O ∪ {n'} ;
```

Les algorithmes 1.1 et 1.2 sont des versions minimalistes d'algorithme de résolution du problème d'optimisation `Min-Sum` (avec équivalence pour `Max-Prod` moyennant le changement de signe). En ce qui concerne le comptage et le calcul de la fonction de partition, il est possible d'adapter ces deux algorithmes. Par exemple pour `DFS`-B&B, il suffirait de ne pas élaguer de branche de l'arbre (c'est à dire de mettre k égal à $\top$) et de simplement additionner les potentiels dans une variable globale. Mais une telle stratégie reste naïve car elle revient à calculer/compter de façon brutale. Nous verrons plus tard dans le chapitre 2 différentes méthodes pour résoudre le problème d'intérêt de cette thèse : le calcul de la fonction de partition.

Nous venons de voir quelques manières de résoudre un problème d'optimisation défini par un modèle graphique. Nous allons passer aux prérequis nécessaires pour comprendre les techniques de résolutions utilisées tout au long du manuscrit. Nous

allons tout d'abord nous intéresser à une technique de transformation d'un réseau de fonction de coût en un autre réseau équivalent, mais simplifié, puis à une technique de décomposition de graphe d'un modèle graphique en sous-parties conditionnellement indépendantes.

1.3 Transformations préservant l'équivalence

Les réseaux de fonctions de coûts utilisent des transferts de coûts locaux entre fonctions appelés transformations préservant l'équivalence (*Equivalence Preserving Transformation* EPT) pour simplifier le modèle graphique et instaurer des propriétés locales désirables sur les fonctions de coûts. Ces propriétés s'appellent **les cohérences locales** (voir section 1.4). Pour qu'une opération soit une EPT, les transferts de coût doivent être faits de sorte à ce que le coût d'une affectation complète reste inchangé.

Définition 1.3.1 (Équivalence)**.** *Deux CFNs* $\mathcal{M}_1 = \langle \boldsymbol{X}, \boldsymbol{D}, \mathscr{C}_1, \sum \rangle$ *et* $\mathcal{M}_2 = \langle \boldsymbol{X}, \boldsymbol{D}, \mathscr{C}_2, \sum \rangle$ *sont* équivalents *si toute affectation a le même coût dans* $\mathcal{M}_1$ *et* $\mathcal{M}_2$ *:*

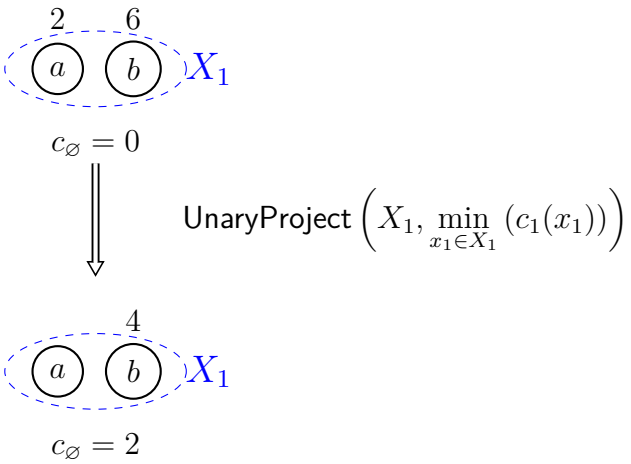
$$\forall \boldsymbol{x} \in D_X, \qquad C_{\mathcal{M}_1}(\boldsymbol{x}) = C_{\mathcal{M}_2}(\boldsymbol{x})$$

Les EPTs sur les CFN se divisent en deux catégories. **La projection** (Project) soustrait une quantité α aux fonctions de coûts d'arité n et l'ajoute à une autre d'arité $n-1$ (voir algorithme 1.3 et 1.4). **L'extension** (Extend) fait l'opération inverse (cela revient à faire une projection de valeur $-\alpha$). Ces deux EPTs peuvent être appliquées sous la condition qu'aucun coût négatif n'apparaisse.

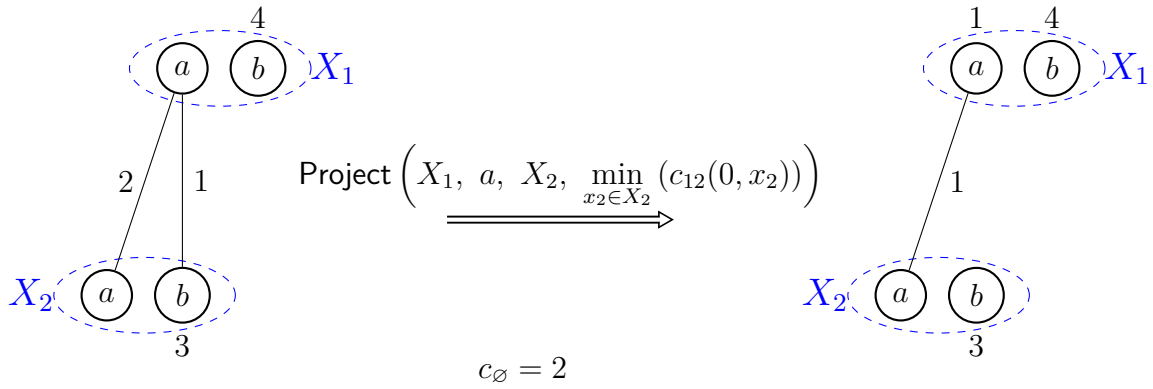
Algorithme 1.3 : Algorithme de projection unaire d'un entier α vers $c_\varnothing$.

```
1 Fonction UnaryProject(X_i, α)
2     c_∅ ← c_∅ + α ;
3     pour x_i ∈ d_i faire
4         c_i(x_i) ← c_i(x_i) − α ;
```

Comme nous le verrons dans la section suivante, il est courant de projeter le minimum des coûts unaires d'une variable sur $c_\varnothing$ (Figure 1.6a) ou le minimum des coûts de fonctions binaires sur une fonction unaire. Ces exemples sont représentés dans la figure 1.6b. Ces EPT sont habituellement utilisées dans les réseaux de



(a) Exemple de projection unaire. Variable X_1, avec un domaine $\mathbf{d_1} = \{a, b\}$, avec comme coût $c_1(a) = 2$ et $c_1(b) = 6$. Le minimum des coûts unaires $\min_{x_1 \in \mathbf{d_1}} c_1(x_1) = 2$ est projeté sur le coût $c_\varnothing$. Le nouveau modèle graphique a $c_\varnothing = 2$, $c_1(a) = 0$ et $c_1(b) = 4$.



(b) Exemple de projection binaire. Variable X_1, avec un domaine $\mathbf{d_1} = \{a, b\}$, Variable X_2, avec un domaine $\mathbf{d_2} = \{a, b\}$ et coût binaire $c_{12}(a, a) = 2$ et $c_{12}(a, b) = 1$. Le minimum $\min_{x_2 \in \mathbf{d_2}} c_{12}(a, x_2) = 1$ est projeté sur $c_1(a)$.

Fig. 1.6 – Exemple d'opérations préservant l'équivalence.

Algorithme 1.4 : Algorithme de projection binaire d'un entier α de c_{ij} vers une fonction unaire c_i.

```
1 Fonction Project(X_i, x_i, X_j, α)
2     c_i(x_i) ← c_i(x_i) + α ;
3     pour x_j ∈ d_j faire
4         c_ij(x_i, x_j) ← c_ij(x_i, x_j) − α ;
```

fonctions de coût car elles font apparaître du **déterminisme**, c'est à dire qu'elles transforment au moins un coût d'une fonction unaire/binaire en un coût nul et qu'elles peuvent aussi faire apparaître des coûts intolérables ($\top$). Enfin, et surtout, elles sont capables de faire croître $c_\varnothing$.

1.4 Cohérences locales

Les EPTs permettent de reformuler un CFN tout en faisant croître le minorant $c_\varnothing$. Il est possible de caractériser le résultat de l'application de ces EPTs au travers de propriétés que devront satisfaire les fonctions de coûts du modèle graphique suite à ces applications. On parle de propriétés de cohérences locales [22, 34–39]. Il existe plusieurs niveaux de cohérences locales en commençant par la plus rudimentaire, s'appliquant uniquement sur les fonctions de coûts unaires, **la cohérence de nœud**.

Cohérence de nœud

La cohérence de nœud (*Node Consistency* NC) garantit que pour toute variable, au moins un des coûts de la fonction unaire est égal à 0 et qu'aucun ne dépasse $\top$.

Définition 1.4.1 (Cohérence de nœud [34])**.** *Un CFN* $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \mathscr{C}, \sum \rangle$ *est* nœud cohérent *si pour toute variable* $X_i \in \boldsymbol{x}$*,*

1. *Pour toute valeur* $x_i \in d_i$*,* $c_i(x_i) + c_\varnothing < \top$
2. *Il existe* $x_i \in d_i$ *tel que* $c_i(x_i) = 0$

Un moyen de garantir efficacement la cohérence de nœud est de projeter pour chaque variable le minimum des coûts unaires sur $c_\varnothing$ (comme présenté dans la section précédente) et d'effacer les valeurs dont le coût unaire dépasse $\top$ (figure 1.7 et algorithme 1.5).

Algorithme 1.5 : Algorithme assurant la cohérence de nœud d'un CFN $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathscr{C}, \sum \rangle$.

```
1 Fonction NC(X)
2     pour X_i ∈ X faire
3         UnaryProject(X_i, min_{x_i ∈ d_i} c_i(x_i)) ;
4         pour x_i dans d_i faire
5             si c_i(x_i) + c_∅ ≥ ⊤ alors
6                 Supprimer x_i de d_i ;
```

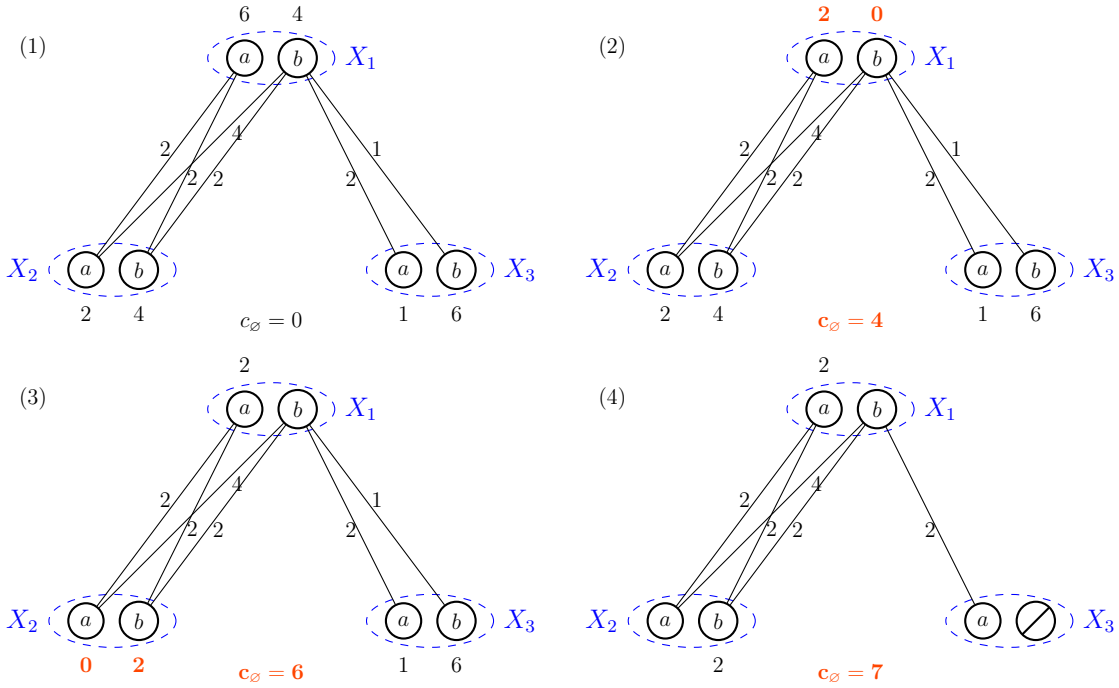


Fig. 1.7 – Établissement de la cohérence de nœud sur un CFN de trois variables avec $\top = 12$.
(1) Problème original avec trois variables (X_1, X_2, X_3) de domaine $\{a, b\}$. Les coûts unaires sont affichés à côté des sommets et les coûts binaires le long des arêtes. En l'absence d'arête, le coût est égal à 0. Initialement $c_\varnothing = 0$.
(2) Première projection de $\min_{x_1 \in d_1} c_1(x_1) = 4$ sur $c_\varnothing$.
(3) Seconde projection de $\min_{x_2 \in d_2} c_2(x_2) = 2$ sur $c_\varnothing$.
(4) Détection d'un coût supérieur à $\top$: $c_\varnothing + c_3(b) \geq \top$.
On supprime la valeur b du domaine de d_3. Et on projette $c_3(a)$ dans $c_\varnothing$

La cohérence de nœud fait apparaître du déterminisme seulement sur les fonctions de coûts unaires. Or dans la figure 1.7, on peut apercevoir que certaines combinaisons de coûts sont supérieures à $\top$, par exemple $c_\varnothing + c_1(a) + c_{12}(a,b) + c_2(b) \geq \top$. Il existe une cohérence locale permettant de faire apparaître du déterminisme dans les fonctions de coûts au moins binaires d'un CFN, **la cohérence d'arc généralisée**.

Cohérence d'arc

La cohérence d'arc généralisée est une propriété qui garantit, en sus de la cohérence de nœud, que pour chaque valeur d'une variable impliquée dans une fonction de coût d'arité au moins 2, il est possible d'étendre cette valeur en une affectation de la fonction de coût nul. Cela revient à dire qu'aucune projection de fonction de coût d'arité supérieure ou égale à 2 ne peut modifier les fonctions de coût unaires. Lorsque le CFN ne possède que des fonctions au plus binaires, la cohérence d'arc généralisée est plus connue sous le nom de cohérence d'arc (*Arc Consistency* AC).

Définition 1.4.2. Cohérence d'arc [35] *Un CFN (binaire)* $\langle X, D, C, \sum \rangle$ *est* arc cohérent *(AC) si il est NC et que pour toute variable* $X_i \in \boldsymbol{x}$*, pour toute fonction* $c_{ij} \in \mathscr{C}$ *et pour toute valeur* $x_i \in d_i$*, il existe* $x_j \in d_j$ *tel que* $c_{ij}(x_i, x_j) = 0$ [‡].

La méthode transformant un réseau en son équivalent arc-cohérent est similaire à celle qui établit la cohérence de nœud. Pour chaque valeur et fonction de coût binaire, il faut projeter le minimum du coût de la fonction binaire dans les fonctions de coûts unaires puis appliquer l'algorithme garantissant NC et répéter cela jusqu'à stabilisation du modèle graphique vers un point fixe (voir figure 1.8). Cependant, le point fixe n'est pas unique et dépend de l'ordre de traitement des variables. L'algorithme `AC*` assure qu'un CFN est arc cohérent, il est décrit dans Larrosa et al. [34] et il est montré dans [40] que la complexité de `AC*` est de $O(ed)$ en espace et de $O(n^2 d^3)$ en temps.

La cohérence d'arc permettra d'aspirer une quantité de coût provenant de fonctions binaires dans le minorant $c_\varnothing$ tout en instaurant du déterminisme. L'accroissement de $c_\varnothing$ est un point essentiel pour résoudre les problèmes d'optimisation [5, 6]. Il existe des cohérences locales plus puissantes que NC et AC, en terme d'accroissement de $c_\varnothing$, appelées FDAC [34], EDAC [36] et VAC [37, 39]. Elles ne seront pas expliquées dans cette thèse mais elles seront utilisées par la suite, car $c_\varnothing$ sera capital dans le calcul de nos majorants sur la fonction de partition Z (voir section 4.1).

‡. Une définition alternative plus forte et élégante a été introduite dans [35], mais elle est peu utilisée en pratique.

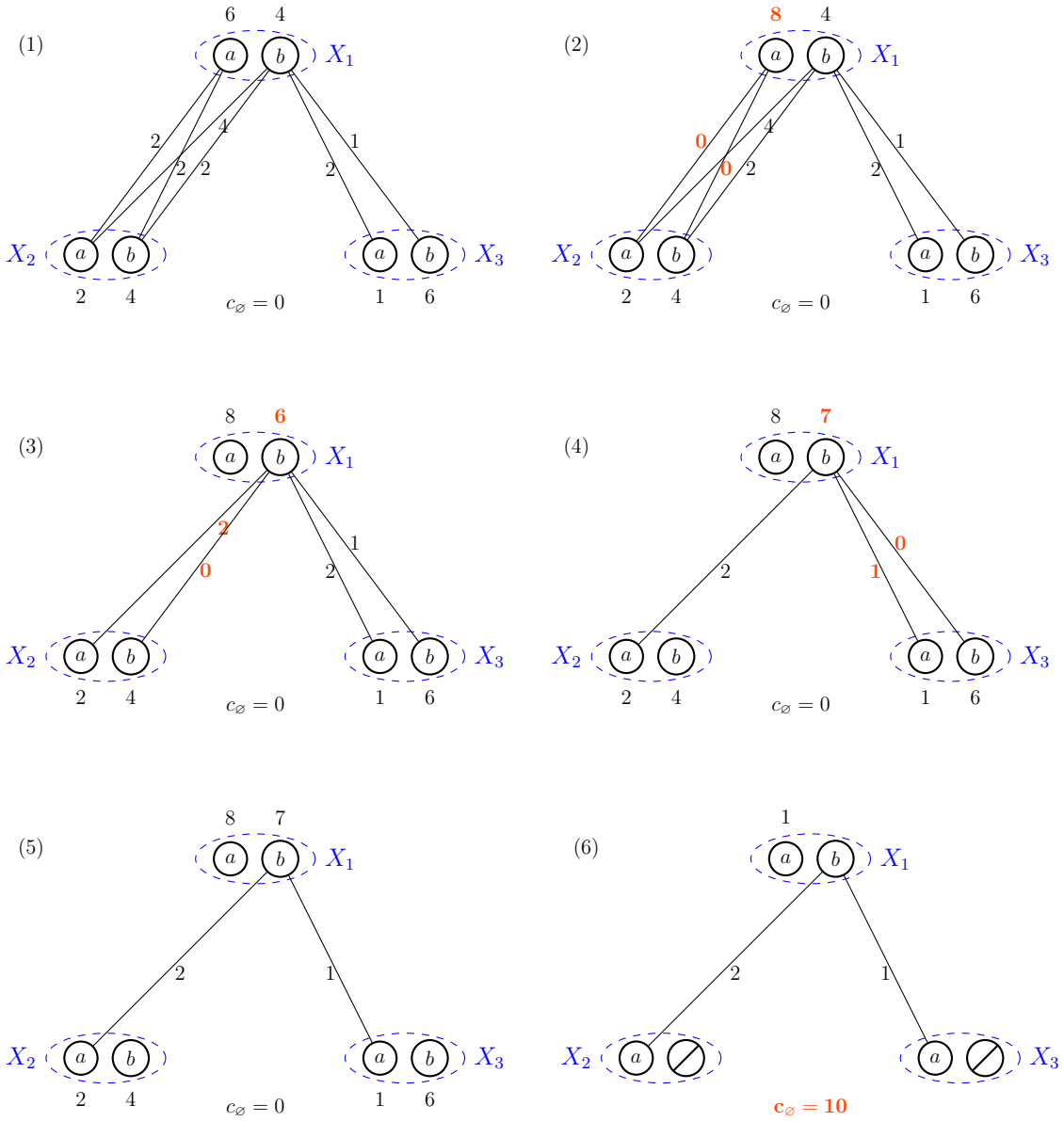


Fig. 1.8 – Etablissement de la cohérence d'arc sur un CFN de 3 variables avec $\top = 12$.
(1) Problème original avec trois variables (X_1, X_2, X_3) (en bleu) et de domaine $\{a, b\}$. Initialement $c_\varnothing = 0$.
(2) La variable X_1 est rendue arc cohérente. Première projection de $\min_{x_2 \in X_2} c_{12}(a, x_2) = 2$ sur $c_1(a)$.
(3) Seconde projection de $\min_{x_2 \in X_2} c_{12}(b, x_2) = 2$ sur $c_1(b)$.
(4) Troisième projection de $\min_{x_3 \in X_3} c_{13}(b, x_3) = 1$ sur $c_1(b)$.
(5) Le graphe est déjà AC pour toutes les variables et fonctions.
(6) Etablir NC.

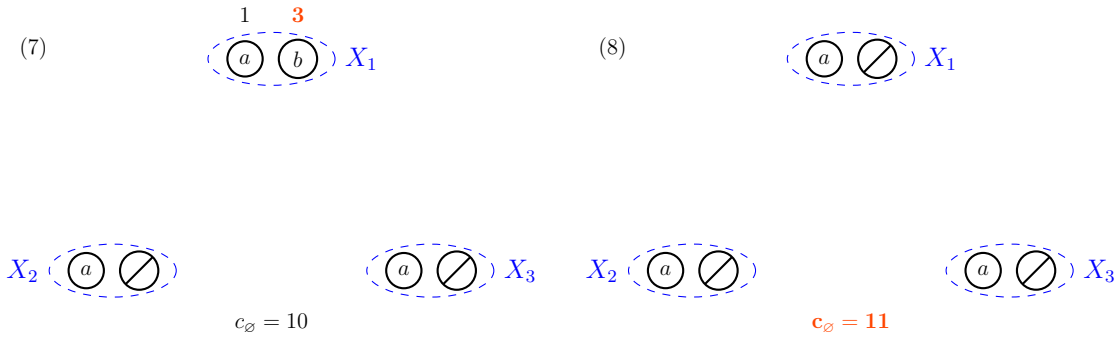


Fig. 1.8 – Fin de l'établissement de la cohérence d'arc.
(7) Dernières projections de $c_{12}(b, a)$ et $c_{13}(b, a)$ vers $c_1(b)$.
(8) Etablir NC.

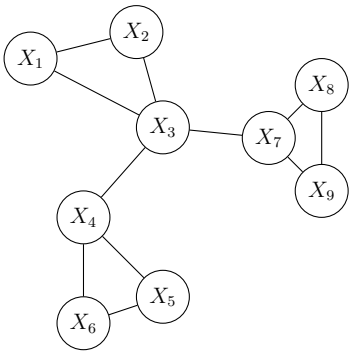
Les cohérences locales exploitent la répartition des coûts sur le modèle graphique. Nous allons maintenant nous intéresser à un outil permettant d'exploiter les parties conditionnellement indépendantes d'un modèle graphique en passant par une décomposition arborescente de son graphe.

1.5 Décomposition arborescente

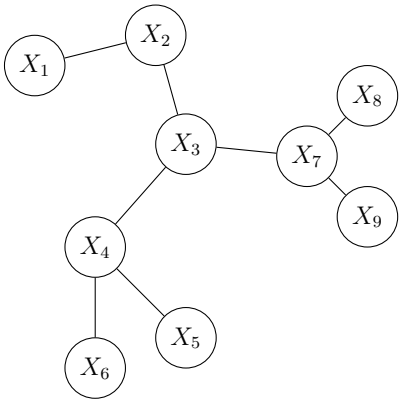
Comme énoncé plus haut, un modèle graphique peut être représenté sous la forme d'un graphe $G = (V, E)$. Un graphe peut avoir différentes structures. Par exemple dans la figure 1.9b, le graphe est un arbre [§] car il est connexe et ne contient pas de cycle. La structure d'un graphe est très importante pour résoudre certains types de problèmes. Nous verrons plus tard dans le chapitre 2 qu'un graphe en structure d'arbre simplifie grandement le calcul de la fonction de partition.

A première vue, le graphe représenté sur la figure 1.9a ne possède rien de particulier, cependant sa structure lui donne en fait quelque chose de très intéressant. En effet, si la variable X_3 est affectée eg. son domaine est réduit à une valeur par une cohérence locale) et si les fonctions l'impliquant sont projetées sur les variables voisines, alors le graphe résultant se décompose en trois sous-graphes indépendants (figure 1.10). Ces trois sous-graphes sont **conditionnellement indépendants** par rapport à la variable X_3. Ainsi, un problème d'optimisation sur ce modèle graphique pourra être résolu de façon brutale avec une complexité en temps de $O(d^9)$, mais si la variable X_3 est affectée en premier, la résolution pourra

§. Ne pas confondre avec l'arbre de recherche



(a) Graphe connexe arbitraire possédant 9 sommets.



(b) Graphe connexe sans cycle possédant 9 sommets : arbre.

Fig. 1.9 – Deux graphes.

se faire en $O\left(d\left(d^2 + d^3 + d^3\right)\right)$ en exploitant l'apparition des trois sous-problèmes indépendants.

En général, un graphe peut toujours être décomposé en sous-ensembles de sommets appelés **clusters** (figure 1.11a). Une **décomposition arborescente** d'un graphe est définie par un arbre de clusters bien organisé, articulé autour de variables formant des **séparateurs** (figure 1.11b).

Définition 1.5.1 (Décomposition arborescente [41])**.** *Soit $G = (V, E)$ un graphe. Une* **décomposition arborescente** *de G est un couple $(\mathbf{C}, T)$ tel que $\mathbf{C}$ est* ***un ensemble de clusters*** *et T est* ***un ensemble d'arêtes*** *connectant les clusters en arbre. Un cluster C_i est un ensemble de sommets $C_i \subset V$. Une arête connectant deux clusters définit un ensemble de sommets appelé* ***séparateur*** *et noté $\mathbf{sep(C_i, C_j)} = C_i \cap C_j$. L'ensemble $\mathbf{C} = \{C_1, ..., C_m\}$ doit vérifier les conditions suivantes :*

1. *L'ensemble des clusters doit contenir toutes les sommets :*

$$\bigcup_{C_e \in \mathbf{C}} C_e = V$$

2. *L'ensemble des clusters doit contenir toutes les arêtes :*

$$\forall (i,j) \in E,\ \exists\ C_e \in C \mid \{i,j\} \subset C_e$$

3. *Si un sommet i apparaît dans deux clusters C_s et C_t, il doit appartenir aux clusters C_e apparaissant sur l'unique chemin allant de C_s à C_t dans (C, T).*

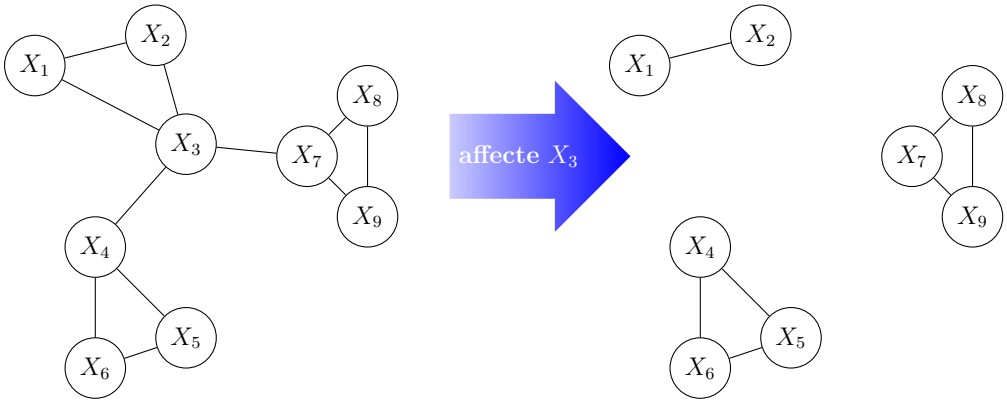
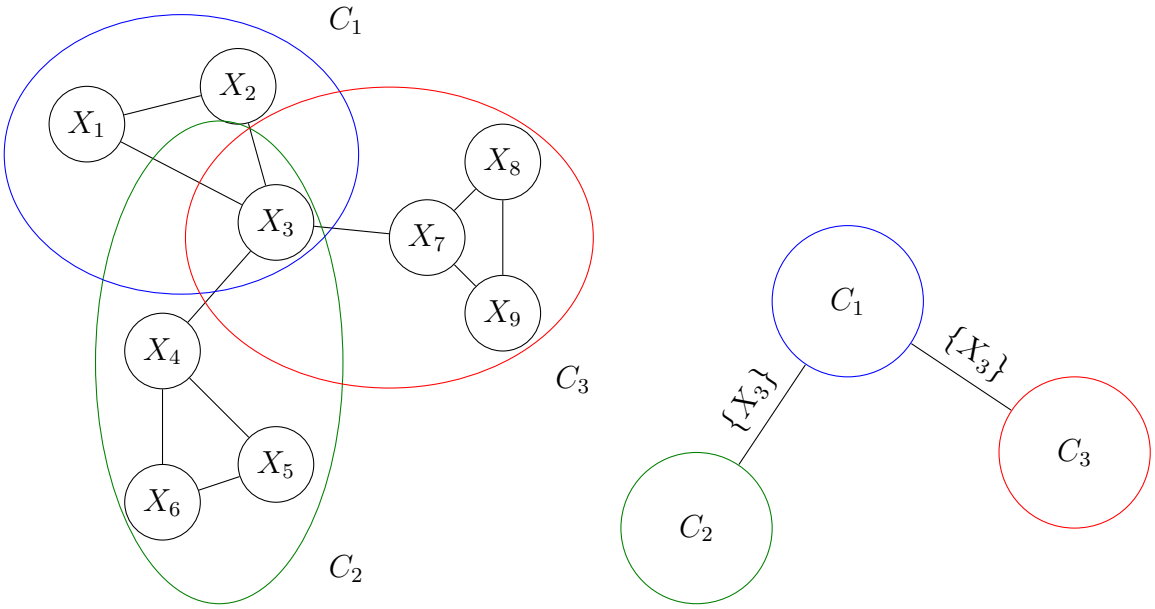


Fig. 1.10 – (Gauche) Graphe quelconque possédant 9 sommets. (Droite) Même graphe où la variable X_3 a été affectée/enlevée

Un exemple de décomposition arborescente est donné dans la figure 1.11. Les séparateurs des clusters peuvent contenir une ou plusieurs variables. Lorsque c'est le graphe d'un modèle graphique qui est décomposé et qu'un cluster est totalement affecté (c'est à dire que toutes les variables correspondant aux sommets du cluster ont été affectées) alors le reste des clusters forment un ou plusieurs sous-problèmes indépendants. Un sous-problème est donc dépendant de l'affectation d'un séparateur. Par exemple dans la figure 1.11, si les variables du cluster C_1 sont affectées (donc a *fortiori* la variable X_3) alors le cluster C_2 devient indépendant du cluster C_3 sachant l'affectation donnée à X_3. La notion de décomposition arborescente est exploitée dans les algorithmes de programmation dynamique [42] mais aussi dans les recherches arborescentes de type `AND/OR` [43] et `BTD` [44].

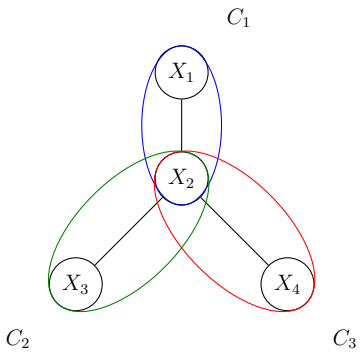
La largeur d'une décomposition arborescente est la taille du plus gros cluster moins un. **La largeur d'arbre ou largeur arborescente** d'un graphe est le minimum des largeurs d'arbres sur l'ensemble des décompositions arborescentes du graphe. Il a été montré que résoudre un problème d'optimisation sur un modèle graphique dont le graphe a une largeur d'arbre r a une complexité en temps de $O(cd^{r+1})$ avec c le nombre de clusters. Une décomposition arborescente alliée à une recherche `DFS` peut donc être plus performante qu'un simple `DFS` [44]. La figure 1.12 montre que la recherche ne se passe plus de la même façon. En effet, une affectation complète prendra la forme d'un sous-arbre (figure 1.12c) et non plus d'une simple branche (figure 1.12b). Cependent, trouver une décomposition arborescente de largeur minimale est un problème NP-difficile [45]. Des heuristiques de décompositions sont fréquemment utilisées.



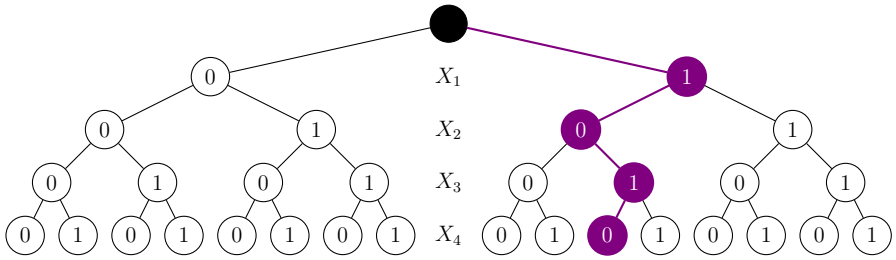
(a) Le cluster $C_1 = \{X_1, X_2, X_3\}$ est en bleu, le cluster $C_2 = \{X_3, X_4, X_5, X_6\}$ est en vert, le cluster $C_3 = \{X_3, X_7, X_8, X_9\}$ est en rouge.

(b) Décomposition arborescente représenté par un arbre de cluster. Le séparateur des clusters est représenté sur l'arête connectant les clusters.

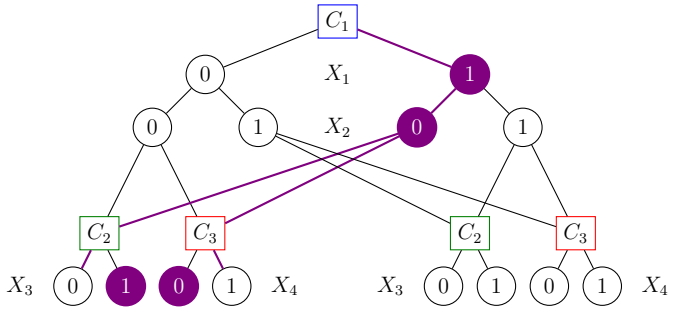
Fig. 1.11 – Décomposition arborescente d'un graphe représentant un modèle graphique de 9 variables.



(a) Décomposition arborescente d'un graphe représentant un modèle graphique de 4 variables.



(b) Le chemin en violet montre l'affectation $(X_1 = 1,\ X_2 = 0,\ X_3 = 1,\ X_4 = 0)$.



(c) Le chemin en violet montre la même affectation $(X_1 = 1,\ X_2 = 0,\ X_3 = 1,\ X_4 = 0)$.

Fig. 1.12 – Différence entre une affectation dans un arbre de recherche simple et dans un arbre de recherche avec une décomposition arborescente.

L'algorithme *Backtrack Tree Decomposition* (`BTD`) combine l'utilisation d'une décomposition arborescente avec un algorithme de type DFS et l'établissement de cohérences locales. Il a permis de résoudre des problèmes d'optimisation intraitables jusqu'ici [46, 47]. Nous verrons plus tard dans la section 4.3, un algorithme basé sur `BTD` que nous avons développé pour calculer la fonction de partition.

Dans ce chapitre, nous avons introduit les modèles graphiques utiles à la modélisation de notre problème d'intérêt ainsi que les notions nécessaires à la résolution d'un problème d'optimisation sur un modèle graphique. Néanmoins, notre objectif final requiert non pas de résoudre un problème d'optimisation mais de calculer la fonction de partition.

Chapitre 2

Principaux algorithmes de calcul de fonction de partition

"You want to know how to rhyme,
You better learn how to add,
It's mathematics."

Mos Def

Pour donner tout leur sens probabiliste aux champs de Markov, il est souvent crucial de connaître la **constante de normalisation Z** qui permet de transformer la fonction potentielle jointe en une distribution de probabilité jointe. En mécanique statistique, la constante de normalisation, aussi appelé **fonction de partition**, permet de décrire certaines propriétés statistiques d'un système thermodynamique à l'équilibre, ce que nous verrons plus en profondeur dans le chapitre 3 section 3.3. Pour rappel, la fonction de partition est :

Définition 2.0.1 (Constante de normalisation ou fonction de partition)**.** *Soit* $(\boldsymbol{X}, \boldsymbol{D}, \Phi, \prod)$ *un MRF.* ***La constante de normalisation*** Z *est la somme des potentiels de la fonction jointe sur toutes les affectations* $\boldsymbol{x} \in \mathbf{D_X}$ *:*

$$Z = \sum_{\boldsymbol{x} \in D_X} \prod_{\phi_S \in \Phi} \phi_S(\boldsymbol{x}[\mathbf{S}]) \tag{2.1}$$

Quand il sera nécessaire et dans un souci de clarté, nous abrégerons parfois l'équation 2.1 en :

$$Z = \sum_{\boldsymbol{x}} \prod_{S} \phi_S(\boldsymbol{x})$$

La fonction de partition n'est donc qu'une "simple" somme de potentiels et son calcul un problème de comptage (*counting*). Ce calcul de Z, classé comme problème #P-complet par la théorie de la complexité [29], reste un défi puisqu'il nécessite de calculer la somme d'un nombre exponentiel de termes. Comme montré dans [29,48], un seul appel à un #P-oracle serait suffisant pour résoudre n'importe quel problème dans la hiérarchie polynomiale de Meyer-Stockmeyer. Par conséquent, la complexité de ces problèmes est au-delà de toutes les couches de complexité d'une hiérarchie dont le premier niveau est la, déjà redoutable, classe NP. Pourtant, le calcul de Z est central dans de nombreux domaines : en statistique pour l'estimation de paramètres, dans le traitement des réseaux Bayésiens pour prendre en compte des observations, en biologie computationnelle (comme nous le verrons dans le chapitre 3) et dans bien d'autres domaines encore.

Pour ces raisons, au fil des années, plusieurs approches ont été développées pour calculer, approximer ou encadrer la fonction de partition. Notamment, il existe plusieurs catégories d'algorithmes essayant de trouver un équilibre entre exactitude et temps de calcul (figure 2.1). Les premiers calculent la fonction de partition de manière **exacte** mais prennent en général beaucoup de temps. Ils s'appuient sur la **structure** et/ou le **déterminisme** du modèle graphique, ce qui en font des algorithmes résolvant uniquement certains problèmes ciblés. La seconde catégorie contient les algorithmes "*Probably Approximately Correct*" (PAC) fournissant des garanties **probabilistes à distance finie**. Cela signifie que l'estimation retournée par un algorithme PAC a une probabilité contrôlée de sortir d'un niveau de qualité lui aussi contrôlé. La troisième catégorie ne fournit que des garanties **asymptotiques**. Enfin, la dernière catégorie ne fournit **aucune garantie** quelle qu'elle soit. Dans la prochaine section, nous allons passer en revue différentes méthodes pour calculer la constante de normalisation.

2.1 Méthodes de calcul exactes

Les méthodes exactes calculent la constante de normalisation avec une garantie absolue. Pour tenter d'éviter d'avoir à ajouter tous les termes de la somme 2.1, ils exploitent deux idées. D'une part, ils exploitent le déterminisme, afin de détecter (ou faire apparaître) des potentiels nuls ou égaux à 1 dans le but d'éliminer les termes ne contribuant pas à la fonction de partition. D'autre part, ils exploitent l'indépendance conditionnelle pouvant être déterminée au niveau structurel du modèle graphique [59] comme le fait la décomposition arborescente. Cela permet de garder en mémoire les parties conditionnellement indépendantes du modèle graphique pour ne pas avoir à les recalculer, ce qui épargne du temps de calcul au prix d'une consommation de mémoire accrue. La même idée est exploitée par la

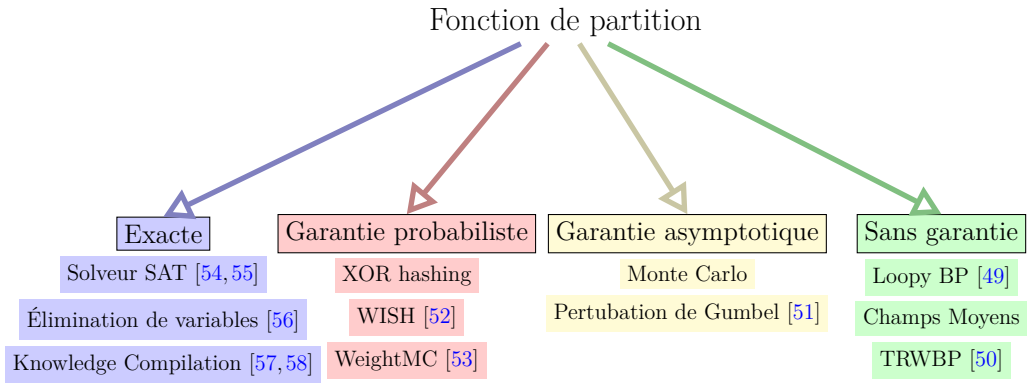


Fig. 2.1 – Différentes catégories de méthodes pour calculer Z.

compilation de connaissance (*knowledge compilation*) qui peut compiler un modèle graphique dans un "langage" particulier pour lequel la tâche de comptage devient plus facile [57, 58].

Dans cette section, nous verrons en détail le calcul de Z basé sur l'élimination de variables `Sum-Prod` [56] puis nous survolerons les méthodes basées sur les solvers SAT [55] et la compilation de connaissance [57, 58].

2.1.1 Élimination de variables

L'algorithme d'élimination de variable appelé aussi *bucket elimination algorithm* [42, 56, 60]) est un algorithme de programmation dynamique permettant de résoudre exactement les tâches d'inférence telles que MAP (`Max-Prod`), ou l'estimation de distribution marginale sur un sous-ensemble de variables (`Sum-Prod`). Si le sous-ensemble de variables est vide alors la marginale devient la constante de normalisation.

Lemme 1 (voir [45])**.** *Soit un champ de Markov* $(\boldsymbol{X}, \boldsymbol{D}, \Phi, \prod)$*. Soit* X_i*, une variable à éliminer. Soit* $\boldsymbol{X}_{-i}$ *le vecteur de variables* $\boldsymbol{X} \backslash X_i$*. Soit* Φ_i *le sous-ensemble de fonctions de* Φ *tel que* $\phi_S \in \Phi_i$ *si et seulement si* $i \in S$ *et* Φ_{-i} *son complémentaire dans* Φ*. Alors la fonction de partition peut se réécrire sous la forme suivante :*

$$Z = \sum_{\boldsymbol{x}_{-i}} \left[\left(\prod_{\phi_S \in \Phi_{-i}} \phi_S(\boldsymbol{x}_{-i}[S]) \right) \left(\sum_{x_i} \prod_{\phi_S \in \Phi_i} \phi_S(\boldsymbol{x}_i[S]) \right) \right] \tag{2.2}$$

Dans cette section, nous nous intéresserons à l'élimination de variables pour le calcul de constante de normalisation, appelé aussi "*Sum-Prod variable elimination*" (`Sum-Prod VE`). Pour l'algorithme `Sum-Prod VE`, il faut premièrement déterminer un ordre d'élimination de variables, puis éliminer les variables une par une comme le montre la figure 2.2. Illustrons l'élimination de variables par un exemple simple avec le champ de Markov présenté dans la figure 2.2a. Z est défini par la quantité suivante :

$$Z = \sum_{x_1,x_2,x_3,x_4,x_5} \phi_{12}(x_1,x_2)\phi_{23}(x_2,x_3)\phi_{34}(x_3,x_4)\phi_{245}(x_2,x_4,x_5)$$

Soit l'ordre d'élimination suivant $(X_1, X_2, X_5, X_3, X_4)$, alors :

$$Z = \sum_{x_2,x_3,x_4,x_5} \phi_{23}(x_2,x_3)\phi_{34}(x_3,x_4)\phi_{245}(x_2,x_4,x_5) \underbrace{\sum_{x_1} \phi_{12}(x_1,x_2)}_{m_2(x_2)} \tag{2.3}$$

$$= \sum_{x_3,x_4,x_5} \phi_{34}(x_3,x_4) \underbrace{\sum_{x_2} m_2(x_2)\phi_{23}(x_2,x_3)\phi_{245}(x_2,x_4,x_5)}_{m_{345}(x_3,x_4,x_5)} \tag{2.4}$$

$$= \sum_{x_3,x_4} \phi_{34}(x_3,x_4) \underbrace{\sum_{x_5} m_{345}(x_3,x_4,x_5)}_{m_{34}(x_3,x_4)} \tag{2.5}$$

$$= \sum_{x_4} \underbrace{\sum_{x_3} m_{34}(x_3,x_4)\phi_{34}(x_3,x_4)}_{m_4(x_4)} \tag{2.6}$$

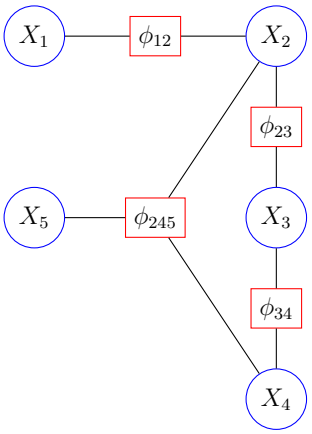
$$= \sum_{x_4} m_4(x_4) \tag{2.7}$$

$$= m_\emptyset \tag{2.8}$$

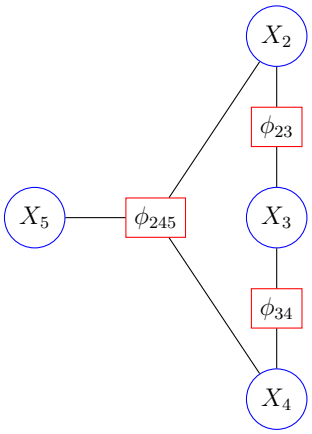
$$\tag{2.9}$$

Lors de l'élimination d'une variable X_i, l'ensemble des variables du modèle graphique ne contient plus la variable X_i, et s'il existe une fonction connectant les variables voisines de X_i alors ces voisines sont connectées entre elles par une fonction. En pratique, si la fonction ϕ_S existe déjà alors le message m_S crée par l'élimination de variables est simplement combiné à ϕ_S. Sinon, le message est représenté par une nouvelle fonction $\phi_S = m_S$. Par exemple, dans la figure 2.2, une fonction $\phi_{345} = m_{345}(x_3, x_4, x_5)$ est créée (en vert sur la figure 2.2c). La tâche prenant le plus de temps lors d'une élimination de variable est le calcul du message car il demande un produit de toutes les fonctions impliquant la variable, sommé sur toutes les combinaisons de ses valeurs.

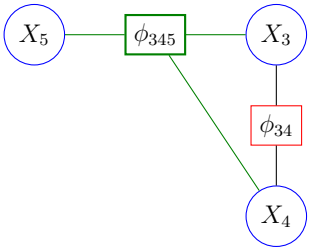
Soit r l'arité maximale d'une fonction créée par l'élimination de variables alors la complexité en espace de l'élimination de variables pour le calcul de la fonction



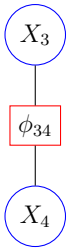
(a) Graphe du modèle graphique d'origine



(b) La variable X_1 est éliminée, un message $m_2(x_2)$ est envoyé à ϕ_2



(c) La variable X_2 est éliminée, un message $m_{345}(x_3, x_4, x_5)$ qui n'existait pas est créé et représenté par la fonction $\phi_{345} = m_2(x_3, x_4, x_5)$



(d) La variable X_5 est éliminée, un message $m_{34}(x_3, x_4)$ est envoyé à ϕ_{34}

Fig. 2.2 – Champ de Markov de cinq variables. L'ordre d'élimination est $(X_1, X_2, X_5, X_3, X_4)$.

de partition est de $O(d^r)$ et celle en temps est de $O(d^{r+1})$ [45]. La complexité de l'élimination de variables est donc dépendante de l'ordre d'élimination. Dans notre exemple, il aurait été plus sage de choisir l'ordre d'élimination X_1, X_5, X_3, X_4, X_2. Malheureusement, trouver l'ordre optimal d'élimination et donc le r le plus petit est un problème classé NP-difficile (qui revient à déterminer la largeur d'arbre du graphe [45]). Cependant, pour des graphes avec une structure particulière, l'élimination de variables peut s'avérer très efficace. Par exemple, lorsque le graphe possède une structure d'arbre, l'élimination de variables permet de calculer Z en temps linéaire en la taille de l'entrée en utilisant un algorithme récursif de programmation dynamique consistant à toujours éliminer une feuille (variable de degré 1) en priorité.

2.1.2 Solveurs SAT et compilation de connaissance

Les solveurs SAT vérifient l'existence d'une solution (aussi appelée un modèle) d'une formule propositionnelle en forme normale conjonctive. Une forme normale conjonctive est une expression logique qui est une conjonction (ET : $\wedge$) de disjonctions (OU : $\vee$) de littéraux (variable booléenne A ou sa négation $\neg A$). Un problème SAT est donc un modèle graphique (cf. définition 1.1.1) où les variables sont booléennes et les fonctions sont exprimées par des disjonctions de littéraux, combinées avec l'opérateur $\wedge$. Le problème SAT est le premier problème à avoir été classé NP-complet en 1971 par Stephen Cook, père fondateur de la théorie de la complexité. Dans les années 90, les solveurs SAT font des progrès faramineux. Beaucoup de compagnies utilisent les solveurs SAT pour la vérification de matériel ou de logiciel informatique [61].

Certains des solveurs SAT ont été adaptés pour résoudre le problème de comptage de modèles, #SAT [54] lui aussi #P complet, puis le comptage #SAT pondéré, où des poids sont associés aux littéraux. Le problème de comptage #SAT pondéré s'apparente au calcul de la fonction de partition. En effet, lorsque tous les littéraux sont affectés, les poids des littéraux affectés à 1 sont multipliés entre eux pour obtenir le poids total du modèle. Enfin, il suffit d'additionner les poids de tous les modèles du problème SAT. Pour avoir un compte pondéré égal à la fonction de partition, le champ de Markov doit donc être encodé en problème SAT [57, 62].

Pour exemple, voici la description de l'encodage `ENC1` proposé dans [57]. Chaque valeur $x_i \in d_i$ de chacune des variables $X_i \in \mathbf{X}$ est représentée par un littéral d_{X_i,x_i}. Ce littéral est vrai si et seulement si la valeur x_i est affectée à la variable X_i. L'encodage de *At Most One* (au plus une valeur est utilisée) est défini par les clauses $(\neg d_{X_i,x_i} \vee \neg d_{X_i,x_j})$ pour tout $X_i \in \mathbf{X}$ et $i < j,\ x_i, x_j \in d_i$ et *At Least One* (au moins une valeur est utilisée) est encodé par la clause $(\bigvee_{x_i} d_{X_i,x_i})$. Ces clauses

assurent que l'encodage ne permet qu'une seule affectation de valeur pour chaque variable.

Chaque fonction potentiel ϕ_S et affectation $\mathbf{x}[\mathbf{S}] \in D_S$ est représentée par le littéral $p_{S,\mathbf{x}[\mathbf{S}]}$. Le poids associé est le potentiel non nul $\phi_S(\mathbf{x}[\mathbf{S}])$ et celui de la négation $\neg p_{S,\mathbf{x}[\mathbf{S}]}$ est pondéré par 1 (élément neutre de la multiplication). Pour chaque variable $X_i \in \mathbf{X}$, on ajoute aussi la clause $(d_{X_i,\mathbf{x}[\mathbf{i}]} \vee \neg p_{S,\mathbf{x}[\mathbf{S}]})$. Ces clauses impliquent que si l'affectation $\mathbf{x}[\mathbf{S}]$ est utilisée alors les valeurs $\mathbf{x}[\mathbf{i}]$ de $\mathbf{x}[\mathbf{S}]$ doivent être aussi utilisées. Enfin, pour chaque valeur $x_i \in d_i$ de chaque chaque variable X_i, les clauses $(\neg d_{X_i,x_i} \vee \bigvee_{\mathbf{x}[\mathbf{S}] \in D_S, \mathbf{x}[\mathbf{i}]=x_i} p_{S,\mathbf{x}[\mathbf{S}]})$ assurent que si la valeur x_i est utilisée alors une des affections $\mathbf{x}[\mathbf{S}]$ telle que $\mathbf{x}[\mathbf{i}] = x_i, \phi_S(\mathbf{x}[\mathbf{S}]) > 0$ est utilisée.

Le solveur #SAT pondéré exact le plus connu à ce jour reste Cachet [55] prenant en argument un mélange de clauses et de littéraux pondérés. Il est basé sur le solveur SAT Zchaff [63, 64] et il utilise le *caching* de formule [65–67] afin de mémoriser certaines quantités pour ne pas avoir à les recalculer (comme le permet une décomposition arborescente). Il utilise aussi le *clause learning* [68–70] et son *caching* s'appuie sur une *dynamic component analysis* [67, 71, 72] qui actualise la décomposition exploitée pendant la recherche.

D'autres algorithmes se basent sur la compilation de connaissance [73, 74]. La compilation de connaissance traduit un modèle graphique en un langage sur lequel la tâche de comptage ou de sommation devient plus facile. Il existe une variété de langages différents adaptés pour certaines tâches. Dans cette thèse, nous nous comparons à deux outils basés sur la compilation de connaissance : ACE [57] et miniC2D [58]. Les méthodes basées sur la compilation de connaissance dépendent beaucoup de leur langage cible qui peut aboutir à des représentations très vite volumineuses. Il y a de plus dans miniC2D une limite technique en nombre de littéraux, ce qui restreint son utilisation. Dans la section suivante, nous allons présenter des méthodes calculant la fonction de partition à partir d'un échantillonnage de la distribution.

2.2 Méthodes d'échantillonnage stochastiques

Les méthodes d'échantillonnage stochastiques sont basées sur le calcul d'une moyenne empirique d'un échantillon d'affectations tirées aléatoirement. Théoriquement, les méthodes d'échantillonnage stochastiques permettent d'approcher la valeur exacte de Z en un temps infini. Les méthodes les plus connues sont celles basées sur l'échantillonnage de **Monte Carlo par Chaîne de Markov (MCMC)** [75–78]. Dans cette section, nous nous intéresserons à MCMC ainsi

qu'à une autre méthode assez proche, **l'échantillonnage de Gumbel** développé par Tamir Hazan [51].

Tout d'abord, si un modèle graphique possède un total de N affectations complètes alors la fonction de partition (définition 2.0.1) peut s'écrire :

$$Z = N \times \mathbb{E}_{\mathbf{x}} \left[P(\mathbf{x}) \right] \tag{2.10}$$

Les méthodes MCMC permettent de tirer un échantillon d'une distribution de probabilité dont la constante de normalisation est inconnue. Un ensemble de méthodes est disponible afin d'avoir cet échantillon [76] dont l'échantillonnage de Gibbs et de Metropolis-Hasting pour en citer quelques unes. Soit $\mathbf{x}^i,\ i \in \{1, \cdots, M\}$ les M échantillons d'affectations fournis par une méthode d'échantillonnage alors le côté droit de l'équation 2.10 peut être estimé comme :

$$\hat{\mathbb{E}}_{\mathbf{x}} \left[P(\mathbf{x}) \right] = \frac{1}{M} \sum_{i=1}^{M} \phi(\mathbf{x}^i)$$

Lorsque $M \to \infty$, alors l'équation 2.10 tend vers la valeur exacte de Z. Cependant, les méthodes MCMC peuvent prendre un temps considérable pour atteindre la distribution de probabilité stationnaire de la chaîne de Markov construite. De plus, il n'y a aucune méthode sûre pour savoir si cette mesure stationnaire est atteinte. Cela a pour effet une estimation de la fonction de partition qui n'est garantie qu'asymptotiquement.

Echantillonnage par perturbation de Gumbel

La méthode d'**échantillonnage par perturbation de Gumbel** repose sur une perturbation des énergies $E(\mathbf{x}[\mathbf{S}]) = \log\left(-\phi(\mathbf{x}[\mathbf{S}])\right)$ du modèle graphique par une distribution de Gumbel $\gamma(\mathbf{x}[\mathbf{S}])$. C'est à dire que toutes les fonctions potentielles du modèle graphique vont être perturbées par des valeurs tirées aléatoirement d'une distribution de Gumbel. Cette perturbation est suivie d'une phase d'optimisation sur le modèle graphique perturbé afin d'avoir l'affectation $\mathbf{x}^*$ dont le potentiel joint est maximal. L'opération est répétée M fois puis une moyenne empirique est calculée afin d'avoir la valeur approchée de Z. L'intérêt de cette méthode est de transformer un problème #P complet en plusieurs problèmes d'optimisations NP-difficiles.

Théorème 2.2.1 ([51]). *Soit $(\boldsymbol{X}, \boldsymbol{D}, \Phi, \prod)$ un MRF et Z sa constante de normalisation. Soit $(\gamma(\boldsymbol{x}))_{\boldsymbol{x} \in \boldsymbol{X}}$ une collection de réalisations indépendantes et identiquement distribuées suivant une loi de Gumbel $\mathcal{G}(-c, 1)$ de fonction de répartition*

$F_\gamma(t) = \exp(-\exp(-(t+c)))$ avec c la constante d'Euler. Alors on peut montrer que $\max_{\boldsymbol{x}\in\boldsymbol{X}}(-E(\boldsymbol{x})+\gamma(\boldsymbol{x}))$, avec $E(\boldsymbol{x}) = -\log P(\boldsymbol{x})$, suit une loi de Gumbel et son espérance est égale au logarithme de la fonction de partition Z du modèle graphique associé.

$$\log(Z) = \mathbb{E}_\gamma\left[\max_{\boldsymbol{x}\in\boldsymbol{X}}(-E(\boldsymbol{x})+\gamma(\boldsymbol{x}))\right] \tag{2.11}$$

Preuve 2.2.1.1. *[51] Posons $W = \max_{\boldsymbol{x}\in\boldsymbol{X}}(-E(\boldsymbol{x})+\gamma(\boldsymbol{x}))$. Calculons la fonction de partition $F_W(t)$ de W.*

Du fait de l'indépendance des lois $\gamma(\boldsymbol{x})$, il en découle que :

$$\begin{aligned}
F_W(t) &= \prod_{\boldsymbol{x}\in\boldsymbol{X}} F_\gamma(t+E(\boldsymbol{x})) \\
&= \prod_{\boldsymbol{x}\in\boldsymbol{X}} \exp(-\exp(-(t+c+E(\boldsymbol{x})))) \\
&= \prod_{\boldsymbol{x}\in\boldsymbol{X}} \exp(-\exp(-(t+c))\exp(-E(\boldsymbol{x}))) \\
&= \exp\left(-\exp(-(t+c))\sum_{\boldsymbol{x}\in X}\exp(-E(\boldsymbol{x}))\right) \\
&= \exp(-\exp(-(t+c))\,Z) \\
&= \exp(-\exp(-(t+c-\log(Z))))
\end{aligned}$$

Sachant que l'espérance d'une loi de Gumbel $\mathcal{G}(\mu,\beta)$ de fonction de répartition : $F(t) = \exp\left(-\exp\left(-(\frac{t-\mu}{\beta})\right)\right)$ est $\mu+\beta c$, alors W suit une loi de Gumbel $\mathcal{G}(-c+\log(Z),1)$ d'espérance $\log(Z)$ □

Le théorème 2.2.1 fournit une approximation de $\log(Z)$ basé sur un échantillonnage, en remplaçant l'espérance par son estimateur (sans biais) empirique. Nous avons testé l'approximation donnée par le théorème 2.2.1 sur un benchmark d'instances tirées d'une modélisation d'interaction de protéines (voir section 3.4 et 4.1 pour la traduction en modèle graphique) et il en est ressorti que l'approximation ne convergeait pas même au bout de 10 000 exécutions, cette approche a donc été abandonnée.

2.3 Méthode sans garantie : Loopy Belief Propagation

Les méthodes sans garantie sont très rapides mais elles n'assurent aucune garantie de qualité sur l'estimation obtenue. Il n'y a donc aucun moyen de savoir

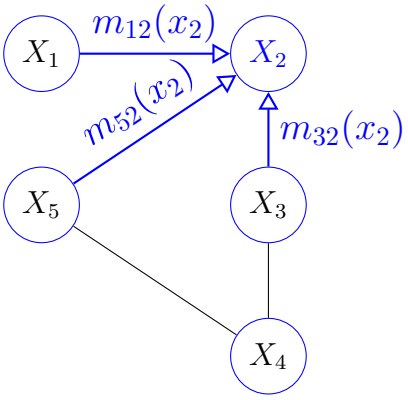


Fig. 2.3 – Modèle graphique de 5 variables. Les variables (X_1, X_3, X_5) envoient leurs messages $m_{i2}(x_2),\ i \in [1, 3, 5]$ à la variable X_2.

à quelle distance de la valeur exacte se trouve l'estimation de Z, ni si elle est inférieure ou supérieure. Pour plus de clarté, nous nous restreindrons aux modèles graphiques binaires mais tous les résultats présentés peuvent se généraliser.

Loopy Belief Propagation (LBP) est un algorithme de type "message passing" initialement proposé par [79] en 1982 pour estimer les marginales d'un réseau bayésien et à été grandement étudié depuis [80,81]. Il est en particulier utilisé dans les algorithmes de *turbo decoding* utilisés en téléphonie mobile [82]. Le principe réside dans le fait qu'une variable peut recevoir des "messages" informatifs de ses voisins (Figure 2.3). Ces messages apportent une information sur la quantité de potentiels que leurs variables apportent à Z. Soit $m_{ij}(x_j)$ le message provenant de i et allant vers j. Soit $\mathcal{N}(i)$, l'ensemble des voisins de la variable X_i connectés par une fonction potentielle ϕ_{ij}. Il est habituel d'initialiser tous les messages à 1 pour ensuite itérer sur chaque variable puis répéter le processus. La mise à jour de ces messages suit l'équation suivante :

$$m_{ij}^{(t+1)}(x_j) = \sum_{x_i} \phi_{ij}(x_i, x_j)\phi_i(x_i) \prod_{\substack{k \in \mathcal{N}(i) \\ k \neq j}} m_{ki}^{(t)}(x_i)$$

Il est conseillé de normaliser les messages $\sum_{x_j} m_{ij}(x_j) = 1$ afin d'éviter les problèmes d'**overflow** ou **underflow** numériques (dépassement de capacité des nombres flottants lorsque t augmente). Sur chacune des variables, LBP calcule une estimation b appelée *belief* de la distribution de probabilité p à partir des messages

échangés avec l'équation suivante :

$$b_i(x_i) \propto \phi_i(x_i) \prod_{j \in \mathcal{N}(i)} m_{ij}(x_i)$$

$$b_{ij}(x_i, x_j) \propto \phi_{ij}(x_i, x_j) \prod_{\substack{k \in \mathcal{N}(j) \\ k \neq i}} m_{jk}(x_j) \prod_{\substack{k \in \mathcal{N}(i) \\ k \neq j}} m_{ik}(x_i)$$

normalisés par :

$$\sum_{x_i} b_i(x_i) = 1$$

$$\sum_{x_i, x_j} b_{ij}(x_i, x_j) = 1$$

Lorsque les conversations ont convergé ou qu'une limite de temps est écoulée, $\log(Z)$ est estimé [83] à partir de :

$$\begin{aligned}\log(\hat{Z}) = &\sum_{\phi_i \in \boldsymbol{\Phi}} \sum_{x_i} b_i(x_i) \log \phi_i(x_i) + \sum_{\phi_{ij} \in \boldsymbol{\Phi}} \sum_{x_i, x_j} b_{ij}(x_i, x_j) \log \phi_{ij}(x_i, x_j) \\ &- \sum_{\phi_{ij} \in \boldsymbol{\Phi}} \sum_{x_i, x_j} b_{ij}(x_i, x_j) \log b_{ij}(x_i, x_j) + \sum_{\phi_i \in \boldsymbol{\Phi}} (d_i - 1) \sum_{x_i} b_i(x_i) \log b_i(x_i)\end{aligned}$$

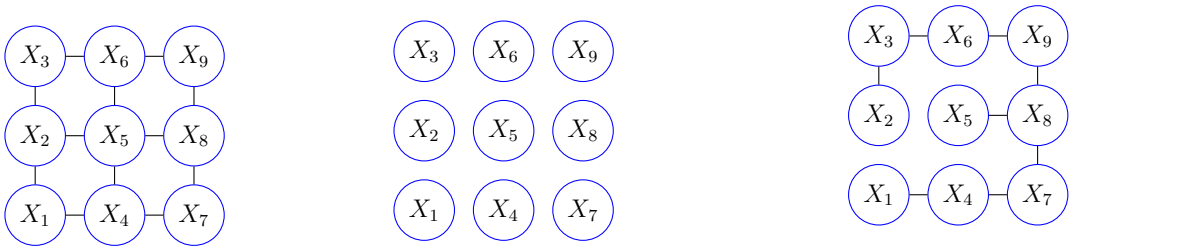
Les algorithmes de *message passing* sont itératifs et dans le cas général, il n'y a aucune preuve qu'ils convergent vers Z. Néanmoins, si le modèle graphique possède un graphe en structure d'arbre alors LBP fournit la valeur exacte de Z car son exécution est équivalente à celle d'un algorithme de `Sum-Prod VE`.

2.4 Majorant et minorant de la fonction de partition

Certains algorithmes ne fournissent pas de garantie quantitative sur la distance les séparant de Z. Toutefois, ils assurent que la valeur obtenue soit un minorant ou un majorant. De plus, en combinant un minorant et un majorant, il devient possible d'obtenir un encadrement de Z, ce qui fournit une garantie quantitative sur Z.

2.4.1 Théorie des champs moyens

La théorie des champs moyens [84] est une théorie permettant d'approximer une distribution $p(\mathbf{x}) = \frac{1}{Z} P(\mathbf{x}) = \frac{1}{Z} \exp(-E(\mathbf{x}))$ par une autre supposée entière-



$p(\mathbf{x}) \propto \prod_{\mathbf{x}} \phi_S(\mathbf{x})$ $\quad$ $q(\mathbf{x}) \propto \prod_{i} q_i(x_i)$ $\quad$ $q(\mathbf{x}) \propto \prod_{i,j\in\mathcal{T}}^{n} q_i(x_i)q_j(x_j)q_{ij}(x_i)$

(a) Graphe du modèle graphique

(b) Champ moyens naïf

(c) Champ moyen structuré sur un arbre $\mathcal{T}$

Fig. 2.4 – Approximation des champs moyens

ment factorisable et simple à analyser. Pour plus de clarté, nous nous restreindrons **aux modèles graphiques binaires** mais tous les résultats présentés peuvent se généraliser. Soit $\mathscr{E}$ l'ensemble des fonctions d'énergie $E_S = -\log(\phi_S)$. Dans ce cas, $p(\mathbf{x})$ s'écrit :

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{\phi_i \in \Phi} \phi_i(x_i) \prod_{\phi_{ij} \in \Phi} \phi_{ij}(x_i, x_j) = \frac{1}{Z} \exp\left(- \sum_{E_i \in \mathscr{E}} E_i(x_i) - \sum_{E_{ij} \in \mathscr{E}} E_{ij}(x_i, x_j) \right)$$

Les champs moyens naïfs (Figure 2.4b) approximent la distribution $p(\mathbf{x})$ par une distribution $q(\mathbf{x})$ en supposant q indépendante en les variables X_i (c'est à dire $q(\mathbf{x}) = \prod q_i(x_i)$) tandis que **les champs moyens structurés** (Figure 2.4c) l'approximent par une distribution q factorisée sous forme d'arbre. L'objectif est d'utiliser la théorie des champs moyens naïf pour estimer la fonction de partition du modèle graphique. Dans ce qui suit, toutes les étapes sont démontrées afin d'obtenir un minorant sur $\log(Z)$. On rappel que dans les champs moyens naïf, $q(\mathbf{x}) = \prod q_i(x_i)$ et on suppose aussi que pour toute variable X_i, $\sum_{x_i} q_i(x_i) = 1$.

Tout d'abord, soit $KL(q||p)$ la divergence de Kullback-Leibler (KL) de p par rapport à q définie comme suit :

$$
\begin{aligned}
KL(q||p) &= \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} \\
&= \sum_{\mathbf{x}} q(\mathbf{x}) \log q(\mathbf{x}) - \sum_{\mathbf{x}} q(\mathbf{x}) \log(p(\mathbf{x})) \\
&= \sum_{\mathbf{x}} q(\mathbf{x}) \log q(\mathbf{x}) - \sum_{\mathbf{x}} q(\mathbf{x}) \log \left(\frac{1}{Z} \exp\left(-E(x)\right) \right) \\
&= \underbrace{\sum_{\mathbf{x}} q(\mathbf{x}) \log q(x)}_{\text{Entropie négative } -\mathcal{S}(q)} + \underbrace{\sum_{\mathbf{x}} q(\mathbf{x}) E(\mathbf{x})}_{\text{Enthalpie } \mathcal{H}(q)} + \log(Z)
\end{aligned} \tag{2.12}
$$

Communément, $\mathcal{S}(q)$ est appelé **l'entropie** de q et $\mathcal{H}(q)$ **l'enthalpie** de E sous q. En injectant la définition des distributions $p(\mathbf{x})$ et de $q(\mathbf{x})$ pour un modèle graphique binaire dans 2.12, il en découle que :

$$
\begin{aligned}
KL(q||p) = &\sum_{E_i \in \mathscr{E}} \left[\sum_{x_i} q_i(x_i) E_i(x_i) - \mathcal{S}_i(q_i) \right] \\
&+ \sum_{E_{ij} \in \mathscr{E}} \sum_{x_i, x_j} q_i(x_i) q_j(x_j) E_{ij}(x_i, x_j) + \log(Z)
\end{aligned} \tag{2.13}
$$

Avec $\mathcal{S}_i(q_i) = -\sum_{x_i} q_i(x_i) \log q_i(x_i)$.

Preuve 2.4.0.1. *Commençons par démontrer que l'entropie $S(q)$ se décompose en somme d'entropies locales $\mathcal{S}_i$ c'est à dire que $\mathcal{S}(q) = \sum_{E_i \in \mathscr{E}} \mathcal{S}_i(q_i)$.*

$$
\begin{aligned}
\mathcal{S}(q) &= -\sum_{\boldsymbol{x}} q(\boldsymbol{x}) \log q(\boldsymbol{x}) \\
&= -\sum_{\boldsymbol{x}} q(\boldsymbol{x}) \log \prod_{E_i \in \mathscr{E}} q_i(x_i) \\
&= -\sum_{\boldsymbol{x}} q(\boldsymbol{x}) \sum_{E_i \in \mathscr{E}} \log q_i(x_i) \\
&= -\sum_{\boldsymbol{x}} \sum_{E_i \in \mathscr{E}} q(\boldsymbol{x}) \log q_i(x_i) \\
&= -\sum_{E_i \in \mathscr{E}} \sum_{\boldsymbol{x}} q(\boldsymbol{x}) \log q_i(x_i) \qquad \textit{(les sommes finies peuvent être échangées)} \\
&= -\sum_{E_i \in \mathscr{E}} \sum_{\boldsymbol{x}} q_i(x_i) \log q_i(x_i) \prod_{\substack{E_j \in \mathscr{E} \\ j \neq i}} q_j(x_j) \\
&= -\sum_{E_i \in \mathscr{E}} \sum_{x_i} q_i(x_i) \log q_i(x_i) \underbrace{\sum_{x_{-i}} \prod_{\substack{E_j \in \mathscr{E} \\ j \neq i}} q_j(x_j)}_{=1} \qquad \textit{avec } x_{-i} = (x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n) \\
&= \sum_{E_i \in \mathscr{E}} \mathcal{S}_i(q_i)
\end{aligned}
$$

Ensuite de manière analogue :

$$\begin{aligned}\sum_{\boldsymbol{x}} q(\boldsymbol{x})E(\boldsymbol{x}) &= \sum_{\boldsymbol{x}} q(\boldsymbol{x}) \left(\sum_{E_i \in \mathscr{E}} E_i(x_i) + \sum_{E_{ij} \in \mathscr{E}} E_{ij}(x_i, x_j) \right) \\ &= \sum_{\boldsymbol{x}} \sum_{E_i \in \mathscr{E}} q(\boldsymbol{x}) E_i(x_i) + \sum_{\boldsymbol{x}} \sum_{E_{ij} \in \mathscr{E}} q(\boldsymbol{x}) E_{ij}(x_i, x_j) \\ &= \sum_{E_i \in \mathscr{E}} \sum_{x_i} q_i(x_i) E_i(x_i) + \sum_{E_{ij} \in \mathscr{E}} \sum_{x_i, x_j} q_i(x_i) q_j(x_j) E_{ij}(x_i, x_j)\end{aligned}$$

Il ne reste plus qu'à additionner les termes avec $\log(Z)$. □

La divergence de KL est une divergence, toujours positive, et donc d'après l'équation 2.13, il découle que :

$$\log(Z) \geq \sum_{E_i \in \mathscr{E}} \left[\mathcal{S}_i(q_i) - \sum_{x_i} q_i(x_i) E_i(x_i) \right] - \sum_{E_{ij} \in \mathscr{E}} \sum_{x_i, x_j} q_i(x_i) q_j(x_j) E_{ij}(x_i, x_j) \quad (2.14)$$

Une maximisation du côté droit de l'équation 2.14 (il y a trivialement l'égalité si $q = p$) fournira le minorant le plus proche de $\log(Z)$. On pose le problème variationnel suivant :

$$\max_q f(q) = \max_q \sum_{E_i \in \mathscr{E}} \left[\mathcal{S}_i(q_i) - \sum_{x_i} q_i(x_i) E_i(x_i) \right] - \sum_{E_{ij} \in \mathscr{E}} \sum_{x_i, x_j} q_i(x_i) q_j(x_j) E_{ij}(x_i, x_j)$$

sous la contrainte d'égalité :

$$\sum_{x_i} q_i(x_i) = 1, \qquad \forall i \in \{1, \cdots n\}$$

Afin **d'optimiser sous contrainte** la fonction $f(q)$, il est courant d'utiliser **une relaxation Lagrangienne**. La relaxation Lagrangienne est une méthode permettant de trouver les points stationnaires (maximum, minimum, point selle) d'une fonction dérivable (ici f) d'une ou plusieurs inconnues (ici q), sous certaines contraintes (ici $\sum_{x_i} q_i(x_i) = 1, \forall i \in \{1, \cdots n\}$). Dans un Lagrangien, la contrainte doit s'annuler lorsqu'elle est vérifiée. Soit $g(q) = \sum_{x_i} q_i(x_i) - 1$ la contrainte relaxée et $f(q)$ la fonction à optimiser, alors le Lagrangien de f sous la contrainte g s'écrit $\mathcal{L}(q, \lambda) = f(q) + \lambda g(q)$ avec $\lambda = (\lambda_i)$ le multiplicateur de Lagrange, il en découle que :

$$\begin{aligned}\max_{q,\lambda} \mathcal{L}(q, \lambda) = \max_{q,\lambda} & \sum_{E_i \in \mathscr{E}} \left[\mathcal{S}_i(q_i) - \sum_{x_i} q_i(x_i) E_i(x_i) \right] - \sum_{E_{ij} \in \mathscr{E}} \sum_{x_i, x_j} q_i(x_i) q_j(x_j) E_{ij}(x_i, x_j) \\ & + \sum_{E_i \in \mathscr{E}} \lambda_i \left[\sum_{x_i} q_i(x_i) - 1 \right]\end{aligned}$$

Afin d'optimiser $\mathcal{L}(q, \lambda)$ en q et λ il faut calculer le gradient du Lagrangien $\nabla\mathcal{L}(q, \lambda)$ ce qui revient à calculer $\frac{d\mathcal{L}}{dq_i(x_i)}$ et $\frac{d\mathcal{L}}{d\lambda_i}$ puis s'intéresser aux cas où les dérivées s'annulent :

$$\frac{d\mathcal{L}}{dq_i(x_i)} = -\log(q_i(x_i)) - 1 - E_i(x_i) - \sum_{j\in\mathcal{N}(i)}\sum_{x_j} q_j(x_j)E_{ij}(x_i, x_j) + \lambda_i = 0$$

$$\Leftrightarrow q_i(x_i) = \exp\left(-E_i(x_i) - \sum_{j\in\mathcal{N}(i)}\sum_{x_j} q_j(x_j)E_{ij}(x_i, x_j)\right)\exp(\lambda_i - 1) \tag{2.15}$$

et

$$\frac{d\mathcal{L}}{d\lambda_i} = \sum_{x_i} q_i(x_i) - 1 = 0$$

$$\Leftrightarrow \sum_{x_i} q_i(x_i) = 1 \tag{2.16}$$

Finalement, en injectant 2.15 dans 2.16 il découle que :

$$\sum_{x_i} q_i(x_i) = \exp(\lambda_i - 1)\sum_{x_i}\exp\left(-E_i(x_i) - \sum_{j\in\mathcal{N}(i)}\sum_{x_j} q_j(x_j)E_{ij}(x_i, x_j)\right) = 1$$

$$\Leftrightarrow \exp(\lambda_i - 1) = \frac{1}{Z_i}$$

$$\text{avec } Z_i = \sum_{x_i}\exp\left(-E_i(x_i) - \sum_{j\in\mathcal{N}(i)}\sum_{x_j} q_j(x_j)E_{ij}(x_i, x_j)\right)$$

$$\text{Donc } q_i(x_i) = \frac{1}{Z_i}\exp\left(-E_i(x_i) - \sum_{j\in\mathcal{N}(i)}\sum_{x_j} q_j(x_j)E_{ij}(x_i, x_j)\right) \tag{2.17}$$

L'équation 2.17 n'étant pas convexe, afin d'optimiser approximativement la distribution q, il est possible d'utiliser une descente en coordonnées (optimisations indépendantes sur chacune des variables).

On rappelle que si $q = p$, alors l'approximation des champs moyens est exacte. Sinon la seule garantie est d'avoir un minorant sur $\log(Z)$ et donc sur Z. On peut aussi généraliser le procédé présenté ci dessus à des champs de Markov avec des fonctions de n'importe quelle arité. Pour cela, il suffira d'ajouter le terme d'arité supérieure dans l'équation 2.14 ainsi que dans la mise à jour de l'algorithme 2.1.

2.4.2 TRWBP et inégalité de Hölder

Afin d'obtenir **un majorant sur** Z, **Tree ReWeighted Belief Propagation** (TRWBP) [50, 85, 86] exploite **la convexité** de la forme exponentielle de

Algorithme 2.1 : Optimisation de la distribution des champs moyens naïf par une descente en coordonnées.

$t \leftarrow 0$;
Initialiser $q^{(t)}$;
tant que maximum d'itération non atteint **faire**
    $q_i^{(t+1)} \leftarrow \exp\left(-E_i(x_i) - \sum_{j \in \mathcal{N}(i)} \sum_{x_j} q_j(x_j) E_{ij}(x_i, x_j)\right)$;
    $Z_i \leftarrow \sum_{x_i} q_i^{t+1}(x_i)$;
    $q_i^{(t+1)} \leftarrow \frac{q_i^{(t+1)}}{Z_i}$;

$\log(Z)$, **Belief Propagation** (voir Section 2.3) et **une pondération par arbre couvrant**. Cette méthode est limitée par le fait que le majorant qu'elle définit est le résultat d'une maximisation finale. Si cette maximisation n'est pas parfaite, le résultat perd sa garantie de majorant. En pratique, son implémentation dans la librairie `libdai` [87] est numériquement très instable et fournit parfois des "majorants" plus faibles que la valeur exacte de Z.

En s'inspirant de cette méthode, Liu et al. [88] ont couplé l'**élimination de variables "par paquet"** (*weighted mini bucket* [89]) et l'**inégalité de Hölder** afin d'avoir un encadrement plus précis de $\log Z$ et ne souffrant pas de la même limitation. Soit $\forall i \in \{1, \cdots, N\},\ \omega_i > 0$ et f_i une collection de N fonctions positives alors l'inégalité de Hölder donne :

$$\left(\sum_{\mathbf{x}} \prod_{i=1}^{n} f_i(\mathbf{x})^{1/\sum_{i=1}^{n} \omega_i}\right)^{\sum_{i=1}^{n} \omega_i} \leq \prod_{i=1}^{n} \left(\sum_{\mathbf{x}} f_i(\mathbf{x})^{1/\omega_i}\right)^{\omega_i}$$

Si $\sum_{i=1}^{n} \omega_i = 1$, si N est le nombre de d'affectations complètes d'un modèle graphique et si les fonctions positives sont les potentiels alors :

$$Z \leq \prod_{i=1}^{N} \left(\sum_{\mathbf{x}} \phi_i(\mathbf{x})^{1/\omega_i}\right)^{\omega_i}$$

Contrairement à TRWBP, le majorant résulte d'une minimisation non convexe et le résultat fournira, quoiqu'il arrive, un majorant.

Chapitre 3

La modélisation et le design des protéines

"Essentially, all models are wrong, but some are useful."

George E. P. Box (1919-2013)

"*Protéine*" provient du nom grec ancien *prôtos* signifiant "premier". La ressemblance avec le terme *Protée* n'est pas anodine, Dieu de la mer pouvant prendre plusieurs formes ou apparences, mer où la vie est apparue en premier. L'étymologie du mot parle d'elle même, les protéines sont des macromolécules centrales dans d'innombrables processus biologiques. L'hémoglobine approvisionne nos cellules en oxygène, l'insuline régule notre taux de glucose, les anticorps nous protègent des infections et le collagène confère une armature à nos tissus et nos organes. Il existe une multitude de protéines toutes spécialisées pour accomplir une tâche spécifique [90]. Les protéines sont produites par des ribosomes traduisant le code d'une molécule d'ARN dit "messager" (portant l'information provenant de l'ADN) en un enchaînement de molécules appelées acides aminés, maintenues par des liaisons peptidiques. Ce polymère adopte une structure tridimensionnelle impliquant des interactions atomiques pour ainsi devenir une protéine fonctionnelle.

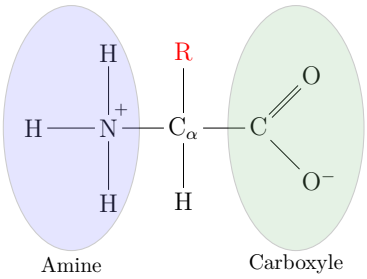
Il existe 20 acides aminés naturels différents dont l'association permet de former des enchaînements différents et par conséquent différentes protéines. Ces enchaînements d'acides aminés se replient en une forme tridimensionnelle (3D) particulière appelée la structure 3D d'une protéine. Ce repliement de la séquence en une structure particulière permet à la protéine d'assurer sa fonction. Ainsi, accéder à la structure des protéines est primordial pour comprendre la fonction des protéines.

Depuis la découverte de la structure 3D d'une protéine en 1951 par Linus Pauling, de plus en plus de structures de protéines ont été résolues, comme en atteste le nombre de structures déposées dans la banque de donnée de structures de protéines (*Protein Data Bank* : PDB). Cependant, la détermination de structures de protéines par des techniques expérimentales (Cristallographie aux rayons X ou Résonance Magnétique Nucléaire) reste encore difficile. Des méthodes de modélisation moléculaire apportent des alternatives pour construire des modèles de structures 3D de protéines, en se basant le plus souvent sur des structures connues de protéines.

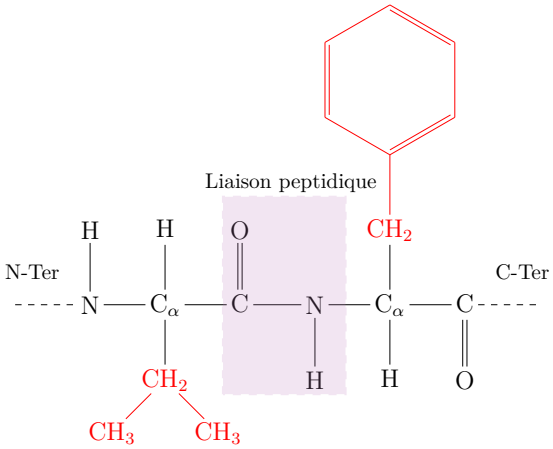
Les relations étroites entre la structure et la fonction des protéines sont très étudiées et ont permis de mieux comprendre, par exemple dans le domaine médical, les mécanismes moléculaires responsables de certaines maladies [91–93]. Ces relations structure-fonction sont également exploitées pour guider la conception de protéines pourvues de fonctions nouvelles ou optimisées notamment dans le cadre des biotechnologies pour tout un panel de domaines d'applications (médical, alimentaire, chimie, bioénergies, bio-raffinerie, cosmétique, *etc.*).

Les technologies d'ingénierie de protéines offrent aujourd'hui des possibilités pour concevoir des protéines dotées de fonctions et/ou de propriétés optimisées et nouvelles mais elles restent encore trop limitées pour répondre aux besoins croissants, particulièrement en enzymes hautement spécifiques. Le défi majeur pour le design de protéine réside dans la taille de l'espace des séquences (10^{13} séquences pour un simple peptide de 10 acides aminés en considérant les 20 types d'acides aminés naturels) dans lequel il faut trouver une combinaison de mutations qui conduit à la protéine ayant les propriétés recherchées. Cependant, la génération et l'exploration de la diversité de séquences par des approches expérimentales (évolution dirigée, mutagénèse dirigée, *etc.*) restent limitées à un petit nombre (10^4 à 10^8) par rapport à la taille de l'espace de recherche. Dès lors, des approches computationnelles sont nécessaires pour explorer et pré-filtrer l'espace des séquences et ainsi, réduire et cibler les acides aminés pertinents sur lesquels l'évolution de la protéine doit être dirigée. C'est dans ce contexte que se situe mon projet de thèse qui porte sur le développement d'algorithmes et de méthodes pour évaluer les combinaisons de mutations les plus bénéfiques à la propriété recherchée et ainsi guider au niveau expérimental la construction de protéines optimisées.

Dans ce chapitre, nous allons décrire la composition et la structure 3D d'une protéine ainsi que les concepts de bases du design computationnel de protéine (CPD) en se focalisant ensuite sur les méthodes d'évaluations de l'affinité de liaison protéine-ligand.



(a) Forme générale d'un acide aminé naturel. La partie variable est notée R (en rouge).



(b) Deux acides aminés, valine et phénylalanine avec leur chaîne latérale spécifique (en rouge) formant une liaison peptidique (encadré en violet)

Fig. 3.1 – Liaison peptidique entre deux acides aminés dans la structure primaire d'une protéine

3.1 De la séquence à la structure 3D

Les protéines sont constituées d'un enchaînement de molécules appelées **acides aminés** [94] reliés entre eux par une liaison covalente. Un acide aminé est constitué d'**une chaîne principale**, composée de deux atomes de carbone, d'un atome d'azote et de deux atomes d'oxygènes, sur laquelle est fixée **une chaîne secondaire** définissant **la partie variable** (figure 3.1a). Les acides aminés sont liés les uns aux autres par des liaisons **peptidiques**. La liaison peptidique résulte de l'amidification du groupe carboxyle (COO^-) de l'acide aminé $i-1$ par le groupe aminé (NH_3^+) de l'acide aminé i (avec libération d'une molécule d'eau). L'extrémité portant le groupe amine est appelée N-terminale, elle est située à gauche par convention alors que l'extrémité carboxyle C-terminale est située à droite.

L'enchaînement d'acides aminés est appelé la séquence ou la structure primaire. Le **squelette** de la protéine est l'enchaînement des chaînes principales des acides aminés alors que les chaînes secondaires sont désignées comme les **chaînes latérales** de la protéine.

Comme le montre la figure 3.1b, les acides aminés diffèrent selon leur chaîne la-

térale qui leur confère des propriétés physico-chimiques différentes. Il existe 20 acides aminés naturels différents (figure 3.2) que l'on peut classer selon qu'ils soient chargés positivement ou négativement (Arg, Asp, Glu, Lys, His), polaires (Asn, Gln, Ser, Thr, Tyr) ou hydrophobes (Ala, Cys, Gly, Ile, Leu, Met, Phe, Pro, Trp, Val). Selon le pH, les liaisons et la place qu'a un acide aminé dans la protéine, il peut changer de catégorie. Par exemple, la cystéine (C) peut être placée dans la catégorie des acides aminés polaires lorsqu'elle ne forme pas un pont disulfure. De même, le tryptophane (T) peut aussi être placé dans la catégorie des acides aminés polaires à cause de son groupe NH. La glycine (G) est un acide aminé difficile à classifier car sa chaîne latérale se limite à un seul hydrogène.

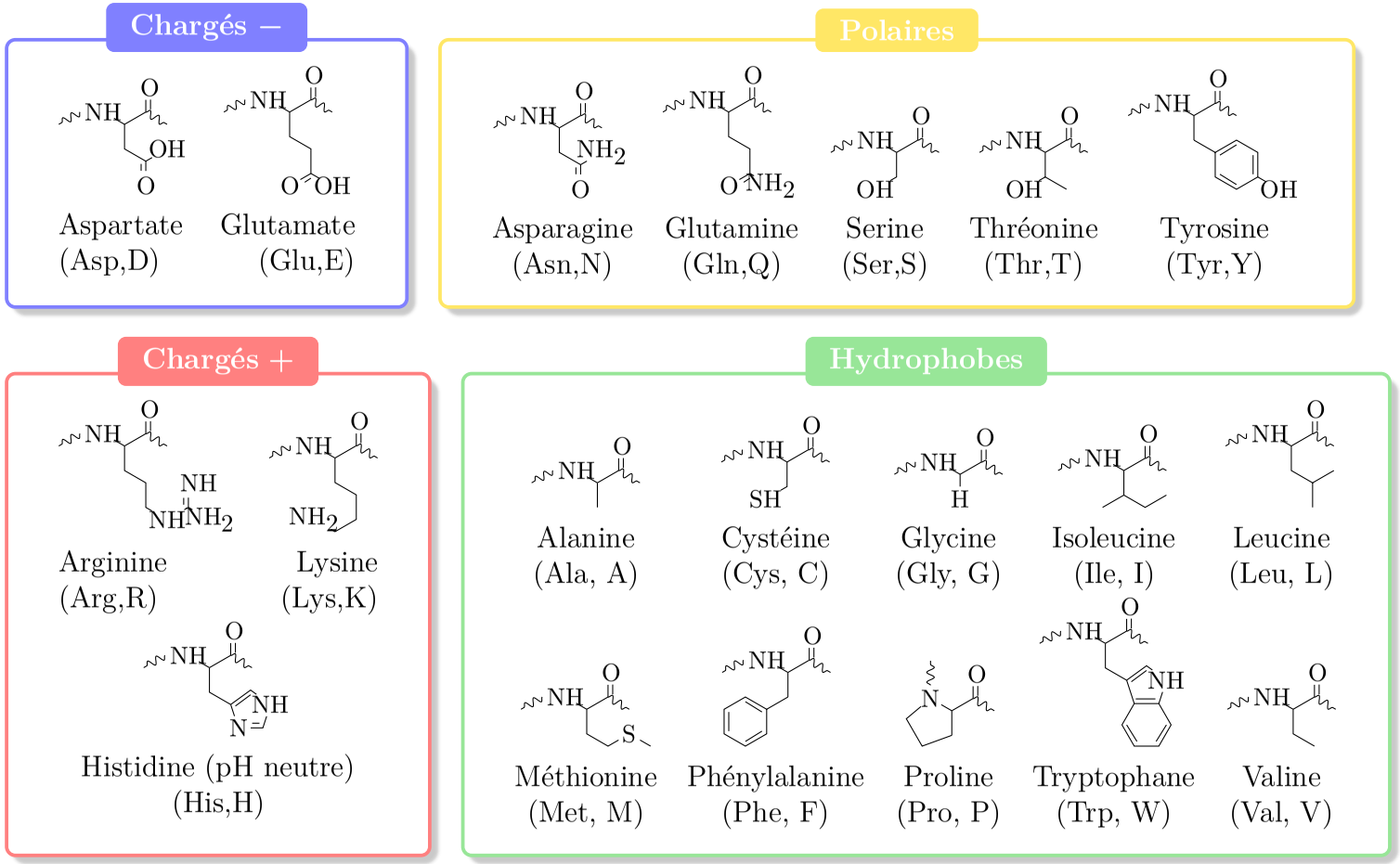


Fig. 3.2 – Les 20 acides aminés naturels.

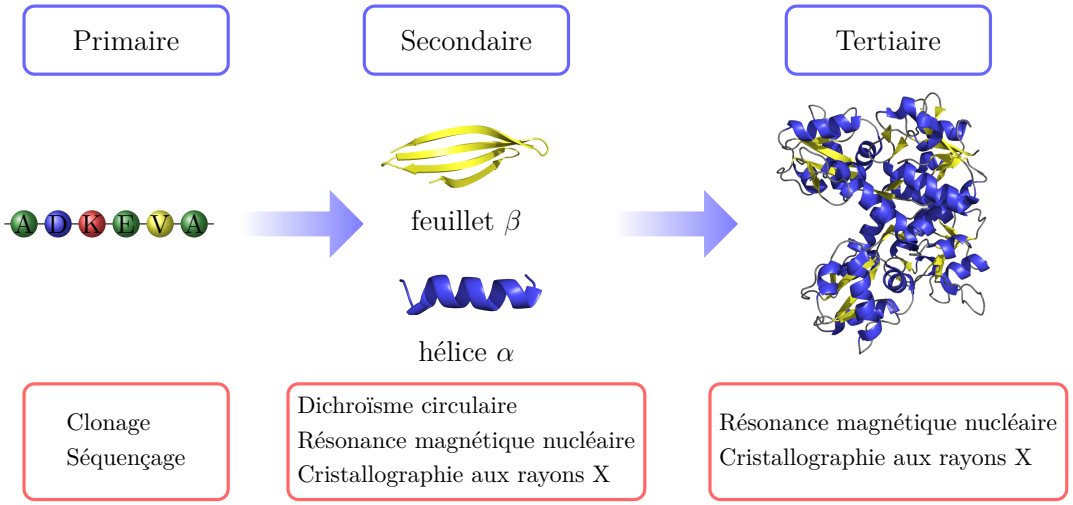


Fig. 3.3 – Structure primaire, secondaire et tertiaire des protéines.

Guidée par différentes interactions entre les acides aminés, la séquence se replie afin d'adopter une structure secondaire puis tertiaire (figure 3.3). La structure secondaire est un repliement local de la structure primaire provenant d'une stabilisation par des interactions atomiques, principalement par des liaisons hydrogènes entre les groupes CO et NH du squelette. Il y a deux structures secondaires principales, **les hélices α** et **les brins β** qui s'assemblent pour former des **feuillets β** (Figure 3.3).

La structure 3D d'une protéine est maintenue par des liaisons covalentes et non covalentes dictées par les types d'acides aminés. Les liaisons non covalentes regroupent les liaisons hydrogènes, de van der Walls et autres interactions électrostatiques et hydrophobes. Dans certaines protéines, des liaisons covalentes appelées ponts disulfures s'établissent entre les soufres de deux cystéines (stabilisant fortement la structure de la protéine). Les structures des protéines peuvent être résolues, notamment grâce à des techniques de cristallographie aux rayons X [95] ou de résonance magnétique nucléaire (*Nuclear Magnetic Resonance* RMN) [96]. Elles sont répertoriées dans la banque de structures de protéines (PDB) [97] (`http://www.rcsb.org/`) sous forme de fichier PDB contenant les coordonnées atomiques des protéines avec une précision (en Å) plus ou moins élevée.

À son commencement en 1976, la PDB contenait seulement 13 structures. À l'heure où cette thèse est rédigée, la PDB contient 132 536 structures. Toutefois, même si de plus en plus de structures sont résolues, il n'y a que 1 393 types de repliements différents répertoriés à ce jour dans la PDB. Il y a donc de nombreuses séquences

qui adoptent un même type de repliement avec une structure très proche. Cette propriété est exploitée par les méthodes de **prédiction de structures assistées par ordinateur** qui essaient de prédire la structure de la protéine à partir de sa séquence [98–100] ainsi que par les méthodes de design de protéines qui recherchent une séquence compatible avec un repliement donné.

3.1.1 Flexibilité des protéines

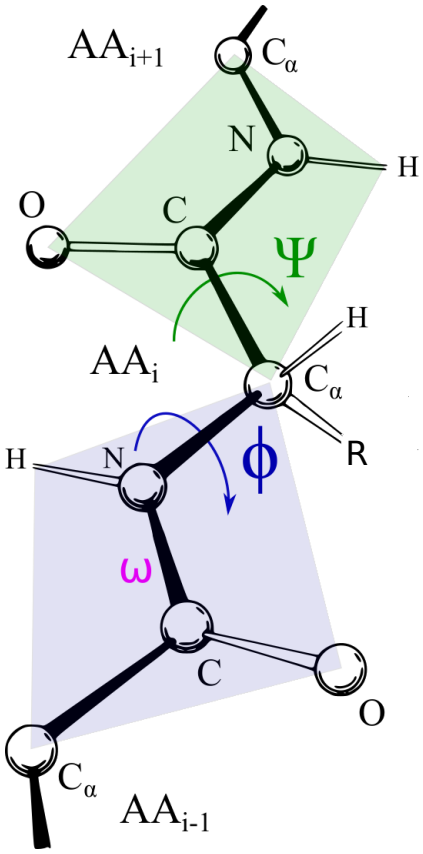
Les protéines sont des entités flexibles subissant des réarrangements conformationnels couvrant une échelle temporelle allant de la femtoseconde (10^{-15} s) à la seconde. Ces changements conformationnels vont de la simple réorientation d'une chaîne latérale à des réarrangements de domaines ou de boucles de larges amplitudes. Il a été montré que la flexibilité locale et les larges mouvements moléculaires jouent un rôle important sur la fonction des protéines [101–103]. Cependant, le plus souvent, les méthodes de biologie structurale et de biophysique ne permettent d'accéder qu'à une vision statique de la structure 3D de la protéine ou seulement à des informations partielles concernant leur flexibilité.

Les degrés de liberté d'une protéine s'expliquent au niveau atomique par des rotations autour de certains angles dièdres. En ce qui concerne le squelette d'une protéine, pour chaque acide aminé, on peut définir trois angles : φ, défini par les liaisons CO-NH — C_α-CO, ψ défini par les liaisons NH-C_α — CO-NH et ω l'angle autour de la liaison peptidique (voir Figure 3.4a). La flexibilité du squelette d'une protéine est principalement dictée par les rotations autours des angles φ et ψ. L'angle ω reste en général très proche de 180°(conformation trans).

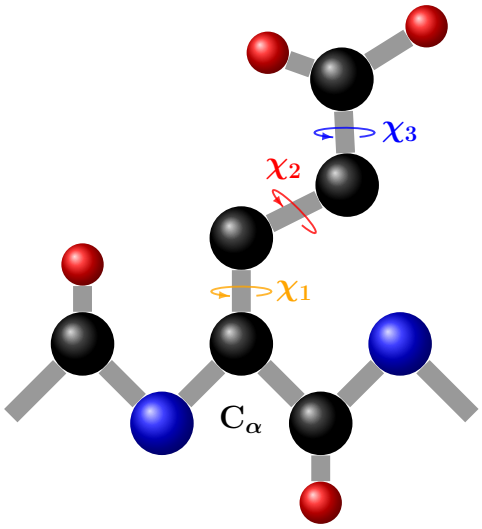
Les degrés de liberté des chaînes latérales sont définis par des angles dièdres χ entre les atomes lourds (figure 3.4b). Suivant la longueur de la chaîne, un acide aminé possède de un à cinq angles χ (figure 3.5a). Plus une chaîne latérale possède d'angles dièdres, plus elle peut adopter un profil conformationnel varié.

Les rotations autours des angles dièdres φ, ψ et χ permettent aux protéines d'adopter différentes conformations à partir d'une structure donnée. En 1969, Cyrus Levinthal remarqua que, en raison du très grand nombre de degrés de libertés dans une protéine, une telle molécule possède un nombre proprement astronomique de conformations possibles [104]. Le paradoxe de Levinthal découle du fait que si une protéine devait adopter exhaustivement toutes ses conformations avant de se replier dans son état natif alors il faudrait un temps considérable même si les transitions se font dans un temps très court. Cependant, ce n'est pas le cas et on observe des temps de repliement de l'ordre de 10^{-6} à 10^{-3} secondes.

Comme énoncé précédemment, les méthodes expérimentales fournissent au mieux



(a) Angles dièdres du squelette $\boldsymbol{\phi}$, $\boldsymbol{\psi}$ et $\boldsymbol{\omega}$.

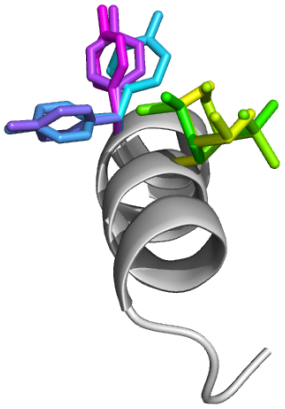


(b) Angles dièdres χ_i de la chaîne latérale du glutamate. Une sphère noire est un carbone, une sphère bleue est un azote et une sphère rouge est un oxygène.

Fig. 3.4 – Degrés de liberté d'une protéine

Nombre d'angle χ				
5	4	3	2	1
ARG	LYS	GLN	ASN	ALA
		GLU	ASP	GLY
		MET	HIS	VAL
			ILE	CYS
			LEU	SER
			PHE	THR
			PRO	
			TRP	
			TYR	

(a) Nombre d'angles dièdres pour chaque acide aminé. Les acides aminés sont écrit selon leur code 3-lettres.



(b) Ensemble de rotamères à deux positions différentes. Les rotamères de couleur vert-jaune sont des rotamères de la méthionine (MET) et ceux en bleu-violet sont les rotamères d'une tyrosine (TYR). Le squelette est représenté en gris.

Fig. 3.5 – Nombres d'angles dièdres et discrétisation des conformations d'une chaîne latérale en rotamères.

des informations partielles sur la flexibilité d'une protéine. Néanmoins, les méthodes de **modélisation moléculaire** permettent d'acquérir des informations complémentaires. Les méthodes basées sur la dynamique moléculaire [105–108] sont très coûteuses en temps de calcul et ressources informatiques. Elles ne sont donc pas adaptées dans le cadre du design computationnel de protéines qui requière d'explorer la flexibilité d'un grand nombre de mutants. Par conséquent, des méthodes modélisant de manière discrète des réarrangements locaux du squelette [109–111] et des chaînes latérales [112, 113] sont utilisées pour le design de protéines.

3.2 Design computationnel de protéines

Les méthodes expérimentales comme la mutagenèse ou l'évolution dirigée [114] ne permettent d'explorer qu'une petite partie de l'espace des séquences. En effet, il existe 20 acides aminés différents, donc si les séquences à tester proviennent de mutations concernant seulement 10 acides aminés, l'espace combinatoire est de $20^{10} = 10^{13}$ séquences de mutants. Cependant, les méthodes expérimentales sont restreintes par un nombre de mutants (au mieux de 10^4 à 10^8) qu'elles sont capables de produire et tester [115]. Pour illustrer ce problème supposons que pour produire et tester un seul mutant, il faille produire un milligramme de protéine en une seconde. Alors cela prendrait 325 millénaires pour produire 10.240 tonnes de protéines pour tester exhaustivement les 10.240 milliards de mutants. Cela étant évidemment inconcevable, il est nécessaire de focaliser les expériences sur les mutants les plus prometteurs pour l'objectif recherché. De plus, les méthodes expérimentales sont coûteuses, souvent fastidieuses et comportent une incertitude quant au résultat. C'est pourquoi, les méthodes computationnelles de design de protéine (en anglais et plus couramment *Computational Protein Design* ou CPD) jouent un rôle central dans les stratégies d'ingénierie de protéines.

3.2.1 Concept de base

Le CPD a pour objectif de limiter la quantité de mutants à tester expérimentalement tout en augmentant les chances de succès et diminuant le temps et le coût associé. Le CPD vise donc à identifier les mutants les plus pertinents vis à vis de la propriété et/ou de la fonction recherchée telle qu'une stabilité accrue, une affinité améliorée avec un partenaire, une nouvelle spécificité de substrat ou encore une nouvelle activité catalytique.

Le design de protéine consiste à l'identification de la (ou les) séquence(s) compatible(s) avec le repliement 3D de la protéine et qui conduiraient aux propriétés nouvelles et/ou optimisées. C'est pourquoi, il est généralement désigné comme étant **le problème inverse de la prédiction de structure** [116, 117]. Au delà de la complexité intrinsèque de la combinatoire des séquences, les méthodes de CPD doivent également prendre en compte la flexibilité des protéines. Ainsi le principal défi en CPD demeure l'exploration de l'espace des séquences combiné à celui des conformations, qui, de par sa taille, reste hors de portée des méthodes computationnelles actuelles.

Des simplifications sont alors introduites dans la modélisation du problème de sorte à le rendre plus gérable. A partir des coordonnées atomiques de la structure 3D d'une protéine (résolue expérimentalement ou issue d'une modélisation), le squelette est considéré comme rigide (fixé dans un état) et la flexibilité des chaînes latérales est représentée par un ensemble **discret** † de conformations issues de l'analyse statistique de structures de protéines connues. En effet, suivant l'observation que, dans une protéine, une chaîne latérale évite la plupart de ses conformations spatiales disponibles et adopte des conformations très ressemblantes, les conformations peuvent être regroupées selon la similitude de leurs angles diédraux [118]. Un représentant d'un groupe de conformations est appelé un **rotamère** (voir figure 3.5b). Les rotamères sont répertoriés dans des librairies plus ou moins denses. La librairie peut être dépendante des angles φ et ψ du squelette comme la librairie de **Dunbrack** [119] ou non dépendante comme la librairie **Penultimate** [120].

Accompagné de la représentation 3D du système, le modèle comprend une fonction qui évalue le design de la protéine, appelée **fonction d'énergie** (les fonctions d'énergie utilisées en mécanique moléculaire sont aussi appelées des **champs de force**). Cette fonction d'énergie permettra d'évaluer les mutants d'une protéine afin de les classer selon le critère défini par le problème de design.

3.2.2 Fonction d'énergie, enthalpie et entropie

Les fonctions d'énergie sont basées sur l'hypothèse d'Anfinsen qu'une protéine se replie en des conformations thermodynamiquement stables, de basses énergies [121]. Par conséquent, les méthodes de CPD ont besoin d'une fonction mathématique capable de discriminer les conformations stables des autres. Les fonctions d'énergie calculent l'**énergie potentielle** de la protéine à partir des interactions au niveau atomique afin de capturer cette stabilité.

†. Discrétiser est l'action de transformer un ensemble continue en un ensemble dénombrable de valeurs individuelles

Une fonction d'énergie est généralement représentée sous forme d'une combinaison linéaire de plusieurs termes énergétiques indépendants (exemple dans l'équation 3.1). Elle est composée de potentiels d'énergie liés et non-liés. Les potentiels liés sont engendrés par les liaison covalentes entre les atomes, on y retrouve les énergies de liaison (*bound*), angulaire (*ang*) et de torsion (*tor*) souvent modélisées via des oscillateurs harmoniques.

Les termes non-liés sont répartis en énergie électrostatique (*elec*), calculée à partir de la loi de Coulomb ainsi qu'en énergie de Van der Waals (*vdw*) approximée à partir du potentiel de Lennard-Jones et de liaisons hydrogènes (*hbond*). Généralement, l'énergie totale $E(\mathbf{x})$ d'une conformation spatiale $\mathbf{x}$ s'écrit donc de la manière suivante [122] :

$$\begin{aligned} E(\mathbf{x}) = & \underbrace{\omega_{bound}E_{bound}(\mathbf{x}) + \omega_{ang}E_{ang}(\mathbf{x}) + \omega_{tor}E_{tor}(\mathbf{x})}_{\text{Énergie covalente}} \\ & + \underbrace{+\omega_{hbond}E_{hbond}(\mathbf{x}) + \omega_{elec}E_{elec}(\mathbf{x}) + \omega_{vdw}E_{vdw}(\mathbf{x})}_{\text{Énergie non-covalente}} \end{aligned} \tag{3.1}$$

Les protéines sont présentes dans un solvant et la conformation stable de la protéine minimise l'exposition au solvant de ses chaînes latérales hydrophobes. Il existe deux façons de prendre en compte le solvant. La première prend en compte le solvant **explicitement** et la seconde **implicitement** [123]. Malgré la précision plus élevée de la modélisation explicite du solvant, elle reste très peu utilisée en CPD car cela rajoute un nombre conséquent de molécules de solvant à prendre en compte dans le calcul de l'énergie. Pour une modélisation en solvant implicite, un terme de solvatation (*solv*) est ajouté.

D'autres termes peuvent y être ajoutés/retirés suivant l'objectif du design. Par exemple, il est courant d'ajouter un terme essayant de prendre en compte l'entropie lorsque la protéine se replie [122] (qui correspond à l'énergie de l'état déplié de la protéine appelé aussi **énergie de référence**). De plus, afin d'atténuer certains clashs stériques dus à l'utilisation de rotamères avec un squelette rigide, il est possible de réduire le rayon atomique dans le potentiel de van der Waals. Communément, le rayon atomique est réduit de cinq pourcents [124, 125] ce qui permet de petits recouvrements entre les atomes.

Les fonctions d'énergie peuvent avoir des paramètres issus de lois de la physique, d'analyses statistiques ou un mélange des deux. Les fonctions d'énergie les plus utilisés en modélisation et dynamiques moléculaires [126] sont AMBER [127, 128], CHARMM [129, 130] et GROMOS [131]. Pour les méthodes de design comprenant une discrétisation de l'espace des conformations des chaînes latérales en rotamères, les fonctions d'énergie décompose généralement les potentiels en fonction des paires

d'acides aminés [132, 133] moyennant quelques approximations. L'énergie est séparée en **énergies unaires** regroupant les interactions internes à un rotamère et ses interactions avec les parties fixes et en **énergies binaires** regroupant les interactions des atomes entre deux rotamères (figure 3.6). Par conséquent, l'énergie $E(\mathbf{x})$ d'une conformation $\mathbf{x}$ définie par une séquence de rotamères $r_i, 1 \leq i \leq n$ se réécrit telle que :

$$E(\mathbf{x}) = E_c + \sum_{i=1}^{n} E_i(r_i) + \sum_{\substack{i=1 \\ i<j}}^{n} E_{i,j}(r_i, r_j)$$

Avec E_c l'énergie des parties fixes, $E_i(r_i)$ l'énergie du rotamère r_i à la position i et $E_{i,j}(r_i, s_j)$ l'énergie d'interaction du rotamère r_i et r_j à la position i et j respectivement. Cette décomposition permet d'économiser du temps de calcul. Au lieu de calculer l'énergie de chaque conformation $E(\mathbf{x})$ l'une après l'autre, la décomposition binaire de l'énergie permet de calculer une fois chaque terme unaire et binaire pour toute combinaison de r_i et r_j puis de stocker les termes dans **une matrice d'énergie** :

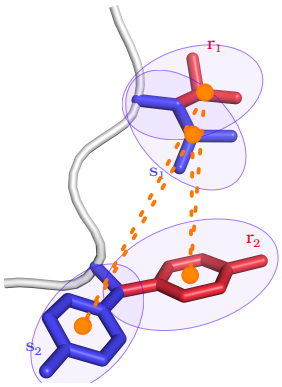
$$\mathcal{M} = \begin{pmatrix} E_1 & \cdots & E_{1,n} \\ & \ddots & \vdots \\ & & E_n \end{pmatrix} \qquad \text{avec} \qquad E_{i,j} = \begin{pmatrix} E_{i,j}(r_1, s_1) & \cdots & E_{i,j}(r_1, s_p) \\ \vdots & \ddots & \vdots \\ E_{i,j}(r_p, s_1) & \cdots & E_{i,j}(r_p, s_p) \end{pmatrix}$$

de taille maximale $\frac{n(n+1)}{2}p^2$ pour une protéine de n acides aminés avec au maximum p rotamères par position. Dans cette thèse, nous nous concentrerons sur les fonctions d'énergie dédiées au design de protéine. Nous allons voir deux fonctions d'énergies qui seront utilisées dans la suite (section 4.1.1 et chapitre 5). Ces fonctions d'énergie sont implémentées dans des logiciels de design de protéines : **Rosetta** [134] et **OSPREY** [135].

Rosetta est un logiciel de modélisation et de design de protéines incorporant plusieurs algorithmes pouvant notamment faire du *docking*, du remodelage de protéine, de la prédiction de structure, de la construction *ab initio*, de la relaxation de structure 3D [134]. Il est un des logiciels les plus utilisés dans le domaine [136–140].

OSPREY (**O**pen **S**ource **P**rotein **R**edesign for **Y**ou) est un autre logiciel de design dédié principalement au remodelage de protéines afin d'optimiser leur stabilité [2], affinité ou spécificité [141]. OSPREY s'est révélé efficace pour le design de différentes protéines, comme en atteste les validations expérimentales [142–144]

La première fonction d'énergie de Rosetta a été implémentée par Simon et al [145, 146]. Aujourd'hui, le programme Rosetta possède une multitude de fonctions d'énergie correspondant à des objectifs variés [147–153]. Ici, nous nous concentrerons



P_1 \ P_2		r_2	s_2
		-3.2	0.1
r_1	-1.7	-3.5	2.5
s_1	-2.4	-1.1	-2.3

Fig. 3.6 – Exemple d'une décomposition en pair d'acides aminés sur deux acides aminés aux positions P_1 et P_2 ayant respectivement un ensemble de rotamère $\{r_1, s_1\}$ et $\{r_2, s_2\}$. Les énergies unaires sont en violet et les énergies binaires en orange.

sur les dernières fonctions d'énergies pour le design de protéine *beta_nov15*† et *beta_nov16*† [154]. Les fonctions *beta_nov∗* sont une combinaison linéaire d'énergies E_i mélangeant termes physiques et statistiques, dépendantes des degrés de libertés géométriques θ_i et de l'acide aminé aa_i :

$$E = \sum_i \omega_i E_i(\theta_i, aa_i) \tag{3.2}$$

Un détail des différent termes de la fonction d'énergie *beta_nov15* est disponible dans le tableau 3.1 [154]. La fonction d'énergie *beta_nov16* est une extension de *beta_nov15* où il est notamment ajouté un terme prenant en compte la possibilité qu'une molécule d'eau puisse former un pont entre deux atomes polaires. Rosetta modélise la flexibilité des chaînes latérales en utilisant la librairie de rotamères de Dunbrack [119]. De plus, *beta_nov∗* est décomposable en termes binaires.

La seconde fonction d'énergie considérée dans la thèse est disponible dans le programme **OSPREY** [135]. La fonction d'énergie est une fonction physique et décomposable en termes binaires. Les termes de van der Waals et électrostatique sont issus soit du champ de force AMBER *ff96* [127, 128], soit du champ de force CHARMM [130]. Les contributions de solvant implicite sont modélisées à l'aide de

†. https://www.rosettacommons.org/docs/latest/rosetta_basics/scoring/Updates-beta-nov15

†. https://www.rosettacommons.org/docs/latest/Updates-beta-nov16

Terme E_i	Description	Poids ω_i	Référence
`fa_atr`	Attraction entre deux atomes de deux acides aminés différents (van der Walls)	1.0	[155, 156]
`fa_rep`	Répulsion entre deux atomes de deux acides aminés différents (van der Walls)	0.55	[155, 156]
`fa_intra_rep`	Répulsion entre deux atomes du même acide aminé (van der Walls)	0.005	[155, 156]
`fa_sol`	Solvant implicite suivant le modèle d'exclusion Gaussienne entre atomes d'acides aminés différents	1.0	[157]
`lk_ball_wtd`	Termes de solvatation implicite orientation-dépendant pour les atomes polaires	1.0	[158, 159]
`fa_intra_sol`	Solvant implicite suivant le modèle d'exclusion Gaussienne entre atomes du même acide aminé	1.0	[157]
`fa_elec`	Interaction électrostatique entre atomes chargés	1.0	[158]
`hbond`	Liaison hydrogène	1.0	[160, 161]
`dslf_fa13`	Pont disulfure	1.25	[161]
`rama_prepro`	Probabilité du squelette (φ, ψ) selon l'acide aminé	0.45	[158, 162]
`p_aa_pp`	Probabilité de chaque type d'acide aminé selon les angles φ, ψ	0.4	[162]
`fa_dun`	Probabilité d'apparition du rotamère selon φ, ψ	0.7	[163]
`omega`	Pénalité pour l'angle ω cis déviant de 0° et ω trans déviant de 180°	0.6	[164]
`pro_close`	Pénalité pour les prolines	1.25	[162]
`yhh_planarity`	Pénalité sinusoïdale pour les χ_3 des tyrosines non planaires	0.625	[161]
`ref`	Energie de référence	1.0	[125, 162]

TABLE *3.1* – Termes de la fonction d'énergie *beta_nov15*.

EFF1 [123]. OSPREY modélise la flexibilité des chaînes latérales en utilisant la librairie de rotamères Penultimate [120].

3.2.3 Échecs, succès et limites

Depuis les dernières décennies, de nombreuses réussites ont été obtenues en design de protéines via l'utilisation de méthodes computationnelles. Les premiers designs de protéines computationnels se sont, pour la plupart concentrés sur le cœur de la protéine [124, 165]. En effet, la structure repliée de la protéine est essentiellement stabilisée par les interactions hydrophobiques au cœur de la protéine [166].

Quelques autres designs complets de protéine construits "*from scratch*" et testés expérimentalement ont suivi [167–169]. L'un des succès les plus connus, en 2003, est la protéine **Top7**, publiée par Kuhlman et al. [170]. Top7 est une construction de protéine *de novo* de 93 acide aminés produite par Brian Kuhlman et Gautam Dantas dans le laboratoire de David Baker à l'université de Washington. La protéine à été conçue par le biais d'un ordinateur puis a été ensuite produite expérimentalement et enfin étudiée par cristallographie aux rayons X (PDB code : 1qys, Figure 3.7a). La structure 3D de la protéine produite expérimentalement s'est avérée être très proche de celle construire par ordinateur (avec une erreur des moindres carrée (RMSD) de 1.2 Å).

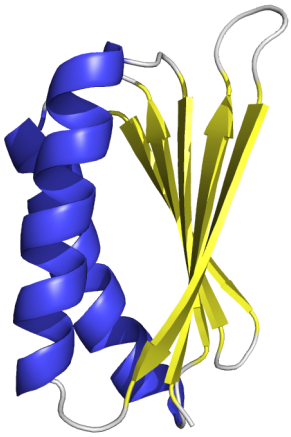
Plus tard en 2008, Jiang et al. créèrent *de novo* une enzyme, une retro-aldolase [†] capable de cliver une liaison carbone-carbone [137]. La même année, Röthlisberger et al. [138] combinèrent leurs méthodes de CPD [3] avec de l'évolution dirigée afin de créer de nouvelles enzymes catalysant la réaction d'élimination de Kemp [‡] pour laquelle aucune enzyme naturelle catalysant cette réaction n'est connue.

Parmi d'autres exemples, en 2014 Voet et al. créèrent pour la première fois une protéine de type beta propulseur entièrement symétrique et validée expérimentalement [171]. La protéine est constituée par l'assemblage de six domaines identiques connues sous le nom de "lames" (figure 3.7b). Elle possède la forme assez particulière d'une pizza qui lui donnera son nom.

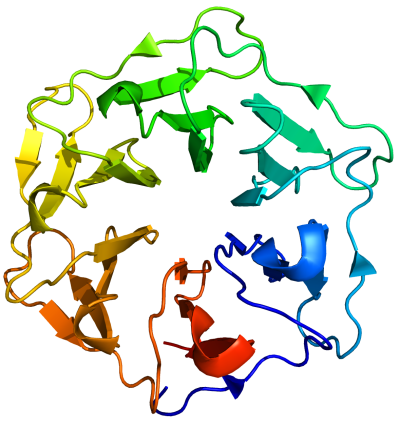
Dans un autre registre, celui de la prédiction de l'affinité de liaison, Kyle E. Roberts et al [142] allèrent de la théorie jusqu'à la pratique en utilisant leur algorithme de prédiction d'affinité K^* (voir section 3.3.2) afin de créer des inhibiteurs de la pro-

†. L'aldolase est une protéine de type enzyme mobilisée en particulier pour la glycolyse (transformation du glucose en énergie).

‡. Réaction de de transfert de proton d'un carbone à une base.



(a) Protéine Top7 constituée de deux hélices alpha (en bleues) et de cinq feuillets bêta (en jaunes). Code PDB : `1qys`.



(b) Protéine de type beta-hélice. Code : `3wwa`. Elle est aussi nommé pizza7H du fait de sa forme particulière.

Fig. 3.7 – Constructions de protéines guidées par des méthodes computationnelles.

téine CFTR $\Delta F508$ (pour Cystic fibrosis transmembrane conductance regulator) retrouvée chez les patients atteints de mucoviscidose.

Christine E. Tinberg et al [172], du laboratoire de David Baker, utilisèrent le logiciel de design **Rosetta** pour concevoir des petites protéines avec une forme complémentaire à un site d'accroche spécifique. Ils ont pu construire des protéines pouvant interagir avec le stéroïde digoxigénine (DIG), une petite molécule utilisée en biologie pour traiter des maladies cardiaques ou détecter certaines biomolécules. De manière similaire, en utilisant Rosetta et le score ZAFFI [173], Brian G. Pierce et al [174] ont pu créer un mutant de récepteur thérapeutique des cellules T quatre cents fois plus sélectif que les mutants précédemment construits. Ces succès ne sont qu'une petite fraction de ce qui a été accomplis durant ces dernières années [175–184]

Malgré la récente évolution du CPD, il reste encore des limitations très difficiles à surmonter. Comme pour les méthodes expérimentales, un des facteurs limitant est le temps. Imaginons que l'on veuille construire *de novo* un peptide composé de 10 acides aminés. L'espace des séquences possibles est de 20^{10}. De plus chaque séquence possède un espace conformationnel. Supposons que chaque chaîne latérale adopte 5 conformations, alors l'espace total **séquence-conformation** sera de l'ordre de $100^{10} = 10^{20}$. Enfin, si un ordinateur prend une nanoseconde pour calculer l'énergie potentielle d'une conformation alors cela prendrait un total de 3

millions d'années pour une énumération exhaustive. Par conséquent, les méthodes de CPD ne sont pas basées sur une énumération exhaustive, elles vont en effet chercher à réduire l'espace de séquences-conformations.

Le LISBP, laboratoire de l'INSA de Toulouse, et le MIAT, laboratoire de l'INRA de Toulouse se sont attaqués au problème d'optimisation consistant à chercher la séquence d'une protéine de plus basse énergie en alliant les méthodes provenant de la bio-informatique et des réseaux de fonctions de coût [4–7].

Cette thèse est plus particulièrement centrée sur la prédiction d'affinité de liaison entre deux partenaires protéiques qui est un problème plus complexe qu'une optimisation d'énergie.

3.3 L'affinité de liaison et l'énergie libre

Prédire l'affinité de liaison entre deux molécules est devenu capital pour toute une gamme d'applications biomédicales, technologiques et industrielles [172–174, 185]. Certaines méthodes expérimentales telles que la *résonance plasmon de surface*, la *titration calorimétrique isotherme* ou la *spectroscopie de fluorescence* permettent de mesurer la cinétique et thermodymanique d'une interaction protéine-ligand [186–190]. Ces méthodes expérimentales fournissent des informations importantes sur la dynamique de liaison des protéines telles que les sites de liaison, les acides aminés importants pour la liaison (*hot-spot*) ou encore les ligands les plus favorables. Elles sont parfois répertoriées dans des bases de données [191–195].

Cependant, les méthodes expérimentales ne peuvent mesurer qu'un nombre limité de complexes de protéines souvent dû au fait du temps de préparation, expression, purification, *etc.* et même si certaines méthodes comme le *deep sequencing* [196–198] permettent de traiter un éventail plus large de mutants, ce nombre reste toutefois limité. C'est pourquoi après avoir attaqué le problème d'optimisation de stabilité, des méthodes de CPD ont été développées pour l'**estimation de l'affinité de liaison**.

Généralement, les méthodes de CPD cherchent à prédire l'affinité de liaison en estimant le changement en énergie libre de Gibbs entre la forme liée et la forme non-liée du complexe. Nous rappelons ici les notions essentielles concernant l'énergie libre G [199]. L'énergie libre se décompose comme $G = \mathcal{H} - T\mathcal{S}$ avec $\mathcal{S}$ l'entropie, $\mathcal{H}$ l'enthalpie et T la température. L'enthalpie $\mathcal{H}$ et l'entropie $\mathcal{S}$ sont définies par :

$$\mathcal{H} = \sum_{\mathbf{x}} E(\mathbf{x})p(\mathbf{x}) \qquad \mathcal{S} = -k_B \sum_{\mathbf{x}} p(\mathbf{x}) \log p(\mathbf{x})$$

$$p(\mathbf{x}) = \frac{1}{Z}\exp(-\beta E(\mathbf{x}))$$

avec k_B la constante de Boltzmann et $\beta = \frac{1}{k_B T}$. La fonction de partition Z (voir section 1.1) apparaît dans le calcul de l'énergie libre G et il en découle que :

$$G = -k_B T \log(Z) \tag{3.3}$$

Au niveau expérimental, l'interaction entre deux protéines est mesurée par une constante d'affinité (inverse de la constante de dissociation) noté $K_a = \frac{1}{K_d}$. La notion de constante d'affinité découle de celle de constante d'équilibre K_{eq} qui caractérise l'équilibre d'une réaction chimique. En effet, pour une réaction à l'équilibre chimique, i.e (R) $\sum_i \alpha_i A_i = 0,\ i \in \{1, \cdots, n\}$ avec A_i composés, α_i leur coefficient stœchiométrique respectif et μ_i leur activité, la constante d'équilibre associée à la réaction R est définie par : $K = \prod_i \mu_i^{\alpha_i}$. A pression constante et lorsque les espèces chimiques sont suffisamment diluées dans un solvant, l'activité μ_i de l'espèce A_i est prise égale à la concentration $[A_i]$ dans le solvant. Dans notre cas, nous considérons trois entités, les deux protéines A et B et le complexe AB, ce qui nous donne la réaction $A + B = AB$. Dans le cadre des protéines, la constante d'équilibre devient la constante d'affinité K_a (ou de dissociation $K_d = \frac{1}{K_a}$) et donne :

$$K_a = \frac{\mu_{AB}}{\mu_A\ \mu_B} = \frac{[\ AB\ ]}{[\ A\ ]\,[\ B\ ]} \tag{3.4}$$

Évidemment, cette équation peut être étendue à n protéines en interaction mais pour plus de simplicité, nous resterons avec un complexe de deux protéines.

L'équation 3.4 est utilisée pour calculer expérimentalement la constante d'affinité d'un complexe de deux protéines. En mécanique classique, l'activité μ_S d'un système S possédant un ensemble discret de conformations, dans un solvant implicite, à volume constant et à température T peut être approximer par sa fonction de partition Z_S i.e $\mu_S \propto Z_S$ [200, 201]. L'équation 3.4 devient :

$$K_a = \frac{\mu_{AB}}{\mu_A\ \mu_B} \propto \frac{Z_{AB}}{Z_A\ Z_B} \tag{3.5}$$

La différence en énergie libre d'un complexe et de sa forme non liée $\Delta G = G_{AB} - (G_A + G_B)$ n'est qu'une autre façon de capturer l'affinité. En effet, il découle de l'équation 3.3 que :

$$\Delta G = -k_B T \log(K_a)$$

Pour résumer, l'affinité de liaison peut s'interpréter de deux façons différentes. La première manière est de calculer **un ratio de fonctions de partition** et la seconde est de passer par **la différence en énergie libre** qui se décompose en

différence en entropie et en enthalpie. La constante d'affinité entre une protéine A et B formant un complexe de protéines AB peut être calculée via :

$$\Delta G = -k_B T \log\left(\frac{Z_{AB}}{Z_A\ Z_B}\right) \tag{3.6}$$

$$\Delta G = G_{AB} - (G_A + G_B) \tag{3.7}$$

Afin de comparer la différence en énergie libre (ou l'affinité de liaison) d'un mutant (S) par rapport à une séquence de référence *ref* (généralement le type sauvage (*wild-type*)), il est courant de calculer la différence d'énergie libre de liaison :

$$\Delta\Delta G = \Delta G_{ref} - \Delta G_S \tag{3.8}$$

Prédire une constante d'affinité nécessite de calculer soit Z, soit $\mathcal{H}$ et $\mathcal{S}$. L'une ou l'autre façon requiert d'additionner un nombre exponentiel de termes. De ce fait, les méthodes de prédiction d'affinité se divisent en différentes catégories (voir tableau 3.2). Théoriquement, des méthodes comme la perturbation d'énergie libre, l'*umbrella sampling* ou l'intégration thermodynamique peuvent estimer la contribution entropique à l'énergie libre [202–207, 207–212]. Même si en théorie ces méthodes sont très précises, elles requièrent un échantillonnage très large (souvent basé sur des dynamiques moléculaires), ce qui est accompagné d'un temps de calcul extrêmement long, en particulier dans le cadre du CPD où il faut répéter la méthode autant de fois qu'il y a de mutants.

Par conséquent, certaines méthodes négligent la contribution entropique en approximant l'énergie libre par celle de la conformation d'énergie minimale. Cette hypothèse se justifie par le fait que si une protéine ne possède que très peu d'états stables et donc de basses énergies alors l'entropie de la protéine sera proche de zéro et son enthalpie proche de la conformation d'énergie minimale.

D'autre méthodes alternatives prennent en compte l'entropie de manière empirique afin d'éviter un calcul très lourd [213–219]. Très souvent, ces méthodes analysent des changements globaux de la protéine tels que le changement de surface accessible au solvant. Ces méthodes sont très avantageuses de par leur temps de calcul très court mais elles négligent certaines contributions, détectables seulement au niveau atomique et sont fortement dépendantes des données sur lesquelles se basent leurs analyses (*over-fitting*).

Enfin, des méthodes utilisent le lien entre l'énergie libre et la fonction de partition et considèrent l'entropie de manière explicite en discrétisant l'espace des conformations [141, 201, 220] (voir section 3.1). Ces méthodes sont souvent basées sur une librairie de rotamères et une fonction d'énergie décomposable en termes binaires. Cette manière de faire est souvent très coûteuse en temps à cause de la complexité de calcul de la fonction de partition.

Entropie empirique		
FoldX	potentiels physiques et statistiques	[213,214]
BeAtMuSic	potentiels statistiques	[215]
BindProf	potentiels basés sur un profil structural	[216]
MutaBind	potentiels physiques et statistiques	[218]
Marillet et al.	potentiels physiques et ajustement de poids par validation croisée	[219]

Entropie physique		
GOBLIN	estimation sans garantie de fonction de partition calculée par LBP	[220]
K^*	approximation ε de fonction de partition calculée par DEE/A^*	[2, 141]

TABLE *3.2* – Méthodes d'estimation d'affinité de liaison

Dans ce chapitre nous allons voir différentes méthodes permettant d'estimer la constante d'affinité ou de manière équivalente la différence en énergie libre.

3.3.1 Estimation avec une entropie empirique

Lorsqu'une ou plusieurs mutations affectent un complexe de protéine, les méthodes estimant le changement d'énergie libre avec une entropie empirique sont souvent les plus rapides et les plus utilisées dans le cadre du design d'interface protéine-ligand (autre protéine, peptide, molécule organique, *etc.*). Dans cette section, nous allons voir plus en détails cinq méthodes : FoldX [213], BeAtMuSiC [215], BindProf [216], MutaBind [218] et une méthode développée par Marillet et al [219].

FoldX

FoldX [213, 214, 221] calcule l'énergie libre de façon empirique et estime l'effet d'une ou plusieurs mutations sur la stabilité d'une protéine ou d'un complexe de

protéines. Même s'il n'est pas destiné à estimer le changement en affinité, il reste grandement utilisé en prédiction d'énergie d'interaction protéine-ligand. Il calcule l'énergie libre à partir de l'équation suivante :

$$\begin{aligned}\Delta G = \omega_1 \Delta G_{vdw} \ + \ \omega_2 \Delta G_{solvH} \ + \ \omega_3 \Delta G_{solvP} \ + \ \omega_4 \Delta G_{wb} \ + \ \omega_5 \Delta G_{hbond} \\ + \ \omega_6 \Delta G_{elec} \ + \ \omega_7 \Delta G_{kon} \ + \ \omega_8 \Delta G_{clash} \ + \ \omega_9 T \Delta S_{mc} \ + \ \omega_{10} T \Delta S_{sc}\end{aligned} \quad (3.9)$$

où les ω_i sont différents poids associés à chaque terme. Les termes usuels sont répartis dans les interactions avec le solvant, ΔG_{solvH} et ΔG_{solvP}, qui contribuent respectivement aux interactions hydrophobes et polaires et ΔG_{wb} qui prend en compte les molécules d'eau avec aux moins deux liaisons hydrogènes avec la ou les protéines. Les autres liaisons hydrogènes sont capturées par ΔG_{hbond}. À partir des lois de Coulomb, l'électrostatique est calculée dans ΔG_{elec} et est ajoutée une contribution pour les atomes appartenant à des chaînes latérales différentes, ΔG_{kon} selon l'équation de Schreiber et al. [222]. ΔG_{clash} est un terme pénalisant les clashs stériques dans la structure 3D. Il est possible d'atténuer les pénalités des clashes entre atomes en utilisant une version de la fonction d'énergie dite *soft*. Enfin, dans FoldX, l'entropie est calculée à partir de deux scores ΔS_{mc} et ΔS_{sc}.

ΔS_{mc} capture la contribution entropique lorsque le squelette est fixé dans une conformation. Ce terme est dérivé d'une analyse statistique de la distribution des angles ϕ/ψ d'un acide aminé sur un ensemble de données de structure 3D de haute résolution et non redondantes. ΔS_{sc} capture la contribution entropique lorsqu'une chaîne latérale est fixée dans une conformation. Ce terme est obtenu à partir des paramètres calculés par Abagayan et al [223]. Le programme FoldX et tous ses composants sont disponibles à l'adresse suivante `http://foldxsuite.crg.eu/`.

Afin de prédire l'affinité, il est conseillé d'utiliser tout d'abord le module `RepairPDB` pour optimiser la structure 3D des complexes puis le module `BuildModel` afin de générer les mutants et enfin le module `AnalyseComplex` pour estimer le score d'affinité en reposant sur l'équation 3.9 ci-dessus. Dans le chapitre 5, nous comparons nos méthodes à FoldX.

BeAtMuSiC

BeAtMuSiC [215] est une méthode ***gros grains*** (*corsed grain*) estimant le changement d'affinité de liaison entre deux protéines induit par **une mutation ponctuelle**. BeAtMuSiC est basé sur des calculs empiriques dérivés d'analyses statistiques sur des données expérimentales de mesure de l'affinité de liaison sur une base de donnée de 368 mutations en alanine [147, 224] et sur des termes empiriques de stabilité de repliement issus d'analyses statistiques effectuées sur un jeu

de données de 2648 mutants provenant d'une précédente publication [225]. La méthode est disponible sur un serveur en ligne à l'adresse http://babylone.ulb.ac.be/beatmusic. BeAtMuSiC s'est retrouvé dans les meilleurs prédicteurs de la 26ème compétition *CAPRI* [226]. Cette méthode a été utilisée, par exemple pour comprendre l'impact de mutations sur l'affinité de liaison entre la protéine gp120 du VIH avec l'anticorps CH103 [227].

MutaBind

MutaBind [218] estime le changement d'énergie libre sur des mutations ponctuelles en utilisant des potentiels issus de fonctions d'énergies de mécanique moléculaire (CHARMM) et empiriques (FoldX). La méthode repose sur une optimisation de la structure 3D en appliquant le protocole `BuildModel` de FoldX suivi d'une minimisation sous contrainte harmonique. Ensuite, la méthode calcule une estimation de ΔG avec une fonction d'énergie mixte. Un détail de la fonction d'énergie est disponible dans [218] et sur https://www.ncbi.nlm.nih.gov/research/mutabind/index.fcgi/method. Enfin, cet outil calibre les potentiels de la fonction par régression linéaire multiple et forêt d'arbres aléatoires. MutaBind est disponible en ligne à l'adresse suivante https://www.ncbi.nlm.nih.gov/research/mutabind/.

Cette approche s'est révélée plus performante que 22 autres méthodes, notamment que FoldX et BeAtMuSiC, pour prédire l'affinité de liaison de complexes protéine-protéine issus de la 26ème compétition CAPRI et tout particulièrement pour prédire les mutations déstabilisant l'interaction [218]. Entre autre, MutaBind a été utilisé afin d'étudier pourquoi le rat taupe était moins sujet aux cancers que l'Homme [228]. En effet, il a été utilisé afin de mettre en évidence les différences d'interaction entre la protéine suppresseur de tumeur p53 avec différentes protéines de réparation de l'ADN.

BindProf

BindProf [216] prédit le changement d'énergie libre pour des mutations situées à l'interface du complexe. Il est basé sur une fonction d'énergie représentant un score du motif de l'interface (*profile interface score*) calibré sur un ensemble de protéines avec une interface de structure similaire [229]. Le score de l'interface est une somme des probabilités de trouver un acide aminé à une position donnée. Ces probabilités sont calculées sur un ensemble de données structurales. En effet, BindProf émet l'hypothèse que la structure étant liée à la fonction, cette information permettrait d'estimer le changement en énergie libre. Il utilise des algorithmes de *machine learning* afin de construire le profil de l'interface et calculer son score. La méthode

est disponible en ligne à l'adresse suivante https://zhanglab.ccmb.med.umich.edu/BindProf/.

Le score de BindProf basé sur le profil structural de l'interface est plus rapide à calculer et fournit parfois une meilleure approximation que d'autres fonctions d'énergies physiques *full-atom* [216]. Une extension de BindProf, appelé BindProfX, a été récemment publiée [230]. Cette version est basée sur le même score que la précédente mais traite de manière différente les probabilités du score d'interface afin de calculer la différence en énergie libre. De plus, BindProfX incorpore la fonction d'énergie de FoldX avec une simple combinaison linéaire des deux fonctions.

Méthode de Marillet et al [219]

Marillet et al. [219] ont développé une méthode permettant d'estimer l'affinité de liaison d'un système macromoléculaire. Ils ont créé différents modèles linéaires, de 5 variables pris sur un panel de 12 variables, optimisés par une validation croisée *5-fold*† sur différentes sous-parties d'un jeu de donnée.

Ces 12 variables sont des scores prenant en compte l'aire de surface et sa géométrie, les *packing properties* [231–234] et les interactions avec le solvant ainsi que l'électrostatique [235, 236].

La base donnée SAB [237] a été divisée en sept sous-parties discriminant des structures considérées comme rigides (3 sous-parties), des structures considérées comme flexibles (2 sous-parties), des structures intermédiaires (1 sous-partie) et des structures de hautes résolutions, dont la résolution est inférieure à 2.5 Å (1 sous-partie).

Les validations croisées sont faites sur 1 585 modèles linéaires contenant 5 variables parmi les 12 mentionnées plus haut et cela pour chacune des sept sous-parties de données (chaque validation croisée *5-fold* est répétée 1000 fois). Différentes statistiques sont calculées (voir les informations supplémentaires de [219]) et le meilleur modèle pour une sous-partie, nommé le modèle spécifique de la sous-partie, est celui qui maximise la corrélation médiane. Le modèle le plus performant sur l'ensemble des données s'est révélé être aussi efficace sur les sept sous-parties, mettant en évidence l'importance de la considération de la morphologie de l'interface et les interactions avec le solvant dans l'estimation de l'affinité de liaison. Ils montrent aussi l'importance d'une bonne résolution de la structure 3D ainsi que l'*over-fitting* dont les méthodes empiriques souffrent le plus souvent.

†. ajustement sur 4/5 des données et test sur les 1/5 restant

3.3.2 Estimation avec une entropie explicite

Comme nous l'avons vu dans la section 3.1, la combinatoire conformationnelle d'une protéine peut atteindre des proportions colossales et rendre l'énergie libre ou la fonction de partition incalculables. De ce fait, certaines méthodes échantillonnent l'espace des conformations. Il existe plusieurs méthodes permettant d'échantillonner l'espace des conformations afin de calculer la différence en énergie libre telles que l'*umbrella sampling* [238–240], l'intégration thermodynamique [241] ou la perturbation en énergie libre par dynamique moléculaire [242]. Cependant ces méthodes sont très coûteuses en temps de calcul lorsqu'il s'agit du cadre du design de protéines qui nécessite l'évaluation d'un grand nombre de mutants.

Nous détaillerons deux méthodes basées sur une discrétisation de l'espace des conformations, GOBLIN et K^*. Nous comparerons nos méthodes avec GOBLIN et K^* (voir section 4.1.2 et chapitre 5).

GOBLIN

GOBLIN (Graphical mOdel for BiomoLecular INteraction) [220] est une méthode qui utilise une décomposition binaire de l'énergie potentielle associée à une librairie de rotamères (voir section 3.1) puis une traduction en champ de Markov afin d'utiliser l'algorithme Loopy Belief Propagation (voir section 2.3) pour calculer l'enthalpie $\mathcal{H}$ et l'entropie $\mathcal{S}$. La décomposition en énergie binaire est faite à partir d'une énergie potentielle décomposable de la forme suivante :

$$E_{goblin}(\mathbf{x}) = \omega_{ljrep}E_{ljrep}(\mathbf{x}) + \omega_{ljatr}E_{ljatr}(\mathbf{x}) + \omega_h E_h(\mathbf{x}) + \omega_{rot}E_{rot}(\mathbf{x})$$

avec $E_{ljrep}(\mathbf{x})$ et $E_{ljatr}(\mathbf{x})$ les énergies répulsives et attractives du potentiel de Lennard-Jones pour modéliser les interactions de van der Walls, $E_h(\mathbf{x})$ l'énergie des liaisons hydrogène comme elle est calculée dans le champ de force Rosetta [160]. Le terme $E_{rot}(\mathbf{x})$ est une correction contrecarrant l'effet de discrétisation des rotamères sur l'entropie. En effet, si on suppose que l'on a un seul acide aminé avec exactement n rotamères ayant une probabilité uniforme $p(x_i) = \frac{1}{n}$ pour $i \in \{1, \cdots, n\}$ alors on a l'entropie $\mathcal{S} = \sum_{i=1}^{n} p(x_i)\log(p(x_i)) = log(n)$. De ce fait, si $n \to \infty$ alors $S \to \infty$.
Un terme d'énergie libre noté $\Delta G_{conf} = G_{conf}(AB) - (G_{conf}(A) + G_{conf}(B))$ est calculé à partir des résultats fournit par LBP. Finalement, GOBLIN calcule le score final $\Delta\Delta G$:

$$\Delta\Delta G = \Delta\Delta G_{conf} + \Delta\Delta G_{SASA} + \Delta\Delta G_{coop}$$

avec :

- $\Delta\Delta G_{conf} = \Delta G^{mut}_{conf} - \Delta G^{wt}_{conf}$ où ΔG^{mut}_{conf} est le terme retourné par LBP pour le mutant et ΔG^{wt}_{conf} pour le sauvage.
- $\Delta\Delta G_{SASA}$ est un terme prenant en compte la perte de surface accessible au solvant lorsque le complexe se forme [213].
- $\Delta\Delta G_{coop}$ est un terme proportionnel à l'aire de la surface d'interaction [243].

Dans leurs travaux, Kamisetty et al. optimisent les poids de leurs fonctions d'énergie sur le jeu de données en minimisant l'erreur des moindres carrées sur l'énergie libre [220].

Ils ont comparé l'estimation de l'affinité, d'un score calculé à partir d'une simple optimisation (voir équation 3.11), d'un score calculé à partir de l'enthalpie et d'un score calculé à partir de l'énergie libre (voir equation 3.7). Ils ont ainsi montré l'importance de l'incorporation de l'entropie dans l'estimation de l'affinité.

OSPREY et l'algorithme K^*

L'algorithme K^* développé par Georgiev et al. [2,141] est basé sur le couplage *Dead End Elimination* `DEE` et A^* [244] et calcule une approximation de la constante d'affinité K_a en passant par le calcul d'un ratio de fonctions de partition (voir équation 3.6). K^* calcule une approximation (avec garantie ε) de chaque fonction de partition afin de ne pas souffrir d'un temps de calcul considérable dû à la complexité du calcul de fonction de partition. Premièrement, comme expliqué dans section 3.4, une matrice d'énergie binaire est calculée en décomposant l'énergie du champ de force AMBER [128] appliquée sur la librairie de rotamères Penultimate [120].

La seconde phase est une réduction de l'espace des rotamères fortement dominés en utilisant `DEE` [245](figure 3.8). `DEE` consiste à éliminer un rotamère s à la position i si l'on peut prouver qu'en aucun cas on ne peut obtenir une meilleure énergie avec s qu'avec un autre rotamère r à cette position. Dans ce cas, on dit que s est dominé. Une condition suffisante pour la dominance peut être obtenue en raisonnant uniquement sur les termes énergétiques impliquant i et les voisins de i (dans $\mathcal{N}(i)$) :

$$\left[E_i(s_i) + \sum_{j \in \mathcal{N}(i)} \min_{t_j} E_{ij}(s_i, t_j)\right] - \left[E_i(r_i) + \sum_{j \in \mathcal{N}(i)} \max_{t_j} E_{ij}(r_i, t_j)\right] > 0 \qquad (3.10)$$

Il existe plusieurs autres conditions suffisantes plus fortes pour le DEE [246], mais leur coût calculatoire est sensiblement plus élevé.

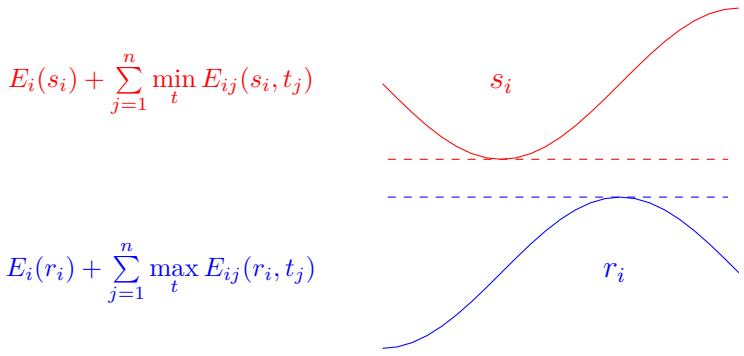


Fig. 3.8 – Exemple d'élimination d'un rotamère par `DEE`. Le rotamère s_i est fortement dominé par le rotamère r_i, on peut donc éliminer s_i sans risque de faire disparaître une conformation d'énergie minimum.

La troisième phase est une énumération des conformations restantes par un algorithme A^* (voir l'algorithme 1.2 et la section 1.2). K^*utilise la capacité de A^* à énumérer les conformations dans un ordre d'énergie croissant afin d'accumuler les conformations de basse énergie en premier et les ajouter dans une estimation de la fonction de partition Z. Dans l'optique de ne pas avoir à accumuler toutes les conformations, il s'arrêtera lorsque la quantité de conformations déjà amassées sera suffisante. Pour cela, l'ensemble des conformations est divisé en trois sous-ensembles : P l'ensemble des conformations éliminées par DEE, V l'ensemble des conformations déjà visitées par A^* et U l'ensemble des conformations encore à explorer. La fonction de partition est donc aussi divisée en une somme de trois quantités : $Z = Z_P + Z_V + Z_U$ sommant sur leur sous-ensemble respectifs. Seule la valeur de Z_V est connue. Néanmoins, il est possible de majorer Z_P en calculant un minorant m_P sur l'énergie de toutes les conformations éliminées par DEE de sorte que :

$$Z_P \leq |P| \exp\left(\frac{-m_P}{k_B T}\right)$$

De plus, si on note E^* la dernière énergie retournée par A^* alors trivialement :

$$Z_U \leq |U| \exp\left(\frac{-E^*}{k_B T}\right)$$

Et en combinant les deux équations ci-dessus, on obtient un majorant sur Z :

$$Z \leq Z_V + |P| \exp\left(\frac{-m_P}{k_B T}\right) + |V| \exp\left(\frac{-E^*}{k_B T}\right)$$

De ce fait, si à un moment de la recherche, on sait que la dernière énergie retournée E^* vérifie :

$$E^* \geq -k_B T \left[\log \left(\frac{\varepsilon}{1+\varepsilon} Z_V - |P| \exp \left(\frac{-m_P}{k_B T} \right) \right) - \log |U| \right]$$

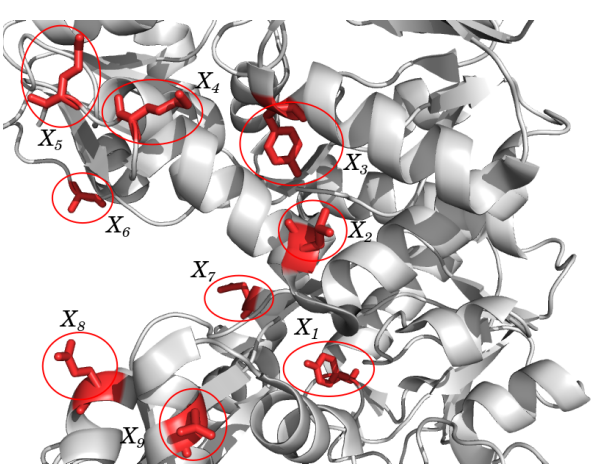
alors l'accumulation d'énergie Z_V est une ε approximation de Z, c'est à dire que $Z_V \leq Z \leq (1+\varepsilon) Z_V$, et A^* peut s'arrêter.

L'avantage de l'algorithme A^* est qu'il énumère les conformations par ordre croissant d'énergie. Il accumule donc les conformations de faibles énergies en premier, ce qui est une bonne stratégie pour une distribution d'énergies avec peu de conformations de basse énergie. Cependant, il reste exponentiel en espace et en temps. De plus l'utilisation d'une élimination DEE est à double tranchant. D'une part, DEE permet de réduire l'espace des conformations à explorer et donc le temps et l'espace utilisés par A^*. D'autre part, si trop de conformations ont été supprimées alors il est possible que l'approximation ε ne soit jamais atteinte. Pour pallier cet effet, un seuil E_ω est ajouté au côté droit de l'équation 3.10. Si par malheur l'approximation ε n'est pas atteinte, alors K^* relance une recherche A^* avec un seuil différent (facilement déterminable).

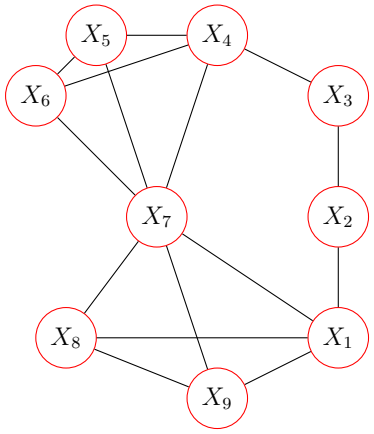
K^* s'est révélé efficace dans de nombreux cas de prédiction d'affinité [142–144]. Des extensions ont été développées [111, 247], le tout implémenté dans le programme OSPREY disponible à l'adresse suivante `http://www.cs.duke.edu/donaldlab/osprey.php` [248].

3.4 Traduction CPD vers modèle graphique

Retranscrire le problème de CPD en un modèle graphique se fait assez directement [5,6] (figure 3.9 et section 1.1). Les n positions d'acides aminés i considérés comme flexibles correspondent aux n variables X_i appartenant à l'ensemble $\mathbf{X}$. Les domaines d_i des variables X_i sont constitués des rotamères de l'acide aminé i. Si l'on souhaite muter un acide aminé i, on fait l'union des ensembles des rotamères des mutants (et du type sauvage, si souhaité). Une affectation de rotamères r_i à une position i est équivalent à une affectation de valeur x_i à une variable i. Enfin les énergies unaires et binaires définissent les fonctions. Dans un champ de Markov, la fonction potentielle d'une affectation $\mathbf{x}$ est définie par une transformation



(a) Protéine avec 9 acides aminés considérés mutables (en rouge).



(b) Exemple de traduction en modèle graphique de 9 variables.

Fig. 3.9 – Traduction d'un problème de CPD à 9 résidus mutables en modèle graphique de 9 variables.

en potentiel de Gibbs : $P(\mathbf{x}) = \exp(-\beta E(\mathbf{x}))$. Il en découle l'équivalence suivante :

$$E(\mathbf{x}) = E_c + \sum_{E_i \in \mathscr{E}} E_i(r_i) + \sum_{E_{ij} \in \mathscr{E}} E_{i,j}(r_i, r_j)$$
$$P(\mathbf{x}) = P_c \times \prod_{\phi_i \in \Phi} \phi_i(x_i) \times \prod_{\phi_{ij} \in \Phi} \phi_{ij}(x_i, x_j)$$

Il est aussi possible de traduire l'énergie ci-dessus en un réseau de fonctions de coût comme vu dans le chapitre 1.

Afin d'avoir le mutant d'une protéine le plus stable en énergie potentielle, il faut avoir la conformation $\mathbf{x}^*$ la plus stable c'est à dire la conformation de plus basse énergie (donc de plus haut potentiel). Une fois cette conformation $\mathbf{x}^*$ obtenue, il suffit de faire correspondre la valeur d'une variable avec le rotamère correspondant. Pour résoudre ce problème de stabilité, il faut résoudre le problème d'optimisation suivant :

$$\mathbf{x}^* = \underset{\mathbf{x} \in D_\mathbf{X}}{\operatorname{argmax}} P(\mathbf{x}) = \underset{\mathbf{x} \in D_\mathbf{X}}{\operatorname{argmin}} E(\mathbf{x})$$

Nous verrons dans la prochaine section que de nombreux succès ont déjà été obtenus grâce à cette méthode d'optimisation dans le cadre du CPD. Mes deux laboratoires d'accueil, le LISBP et le MIAT ont utilisé cette traduction de CPD vers modèle graphique ainsi que des outils provenant des CFN pour résoudre ce

problème d'optimisation [4–7] (voir section 1.4). Ils ont montré que ces nouvelles méthodes de CPD sont plus performantes en terme de temps de calcul et plus précises qu'une méthode d'optimisation stochastique très usuelle implémentée dans le programme Rosetta.

Bien que l'énergie de la conformation la plus stable soit parfois une bonne approximation de l'enthalpie, souvent satisfaisante au niveau d'une protéine seule, cela peut s'avérer insuffisant pour prédire l'affinité d'un complexe de protéines [141]. En effet, supposons qu'un mutant soit plus stable que le type sauvage. Il est possible que ce mutant soit encore plus stable dans sa forme non liée qu'en complexe. Dans ce cas, le complexe ne se formera pas. Par conséquent, il faut aussi prendre en compte la stabilité de la forme non liée. On appelle communément ce score

$$\Delta E = E_{AB}(\mathbf{x}^*) - (E_A(\mathbf{x}^*) + E_B(\mathbf{x}^*)) \tag{3.11}$$

Néanmoins, le score ΔE peut aussi s'avérer être insuffisant dans le cas où le complexe et/ou les partenaires possèdent de nombreux états de basses énergies [141, 220]. C'est pourquoi dans cette thèse, nous avons à la fois développé une méthode basée sur la conformation d'énergie minimale (GMEC) et une méthode basée sur la fonction de partition pour estimer la constante d'affinité (voir sections 3.3 et les équations 3.7 et 3.6). Par conséquent, il faut définir trois modèles graphiques différents. Celui du complexe, du premier partenaire puis du second. Les modèles graphiques sont définis de la même manière qu'antérieurement (figure 3.9). L'union des deux modèles graphiques des protéines ne donne pas celui du complexe, il faut donc pré-calculer les matrices d'énergies des trois entités séparément (figure 3.10). Par conséquent, toutes les méthodes présentées dans les chapitres 1 et 2 peuvent être appliquées à l'aide de la modélisation ci-dessus.

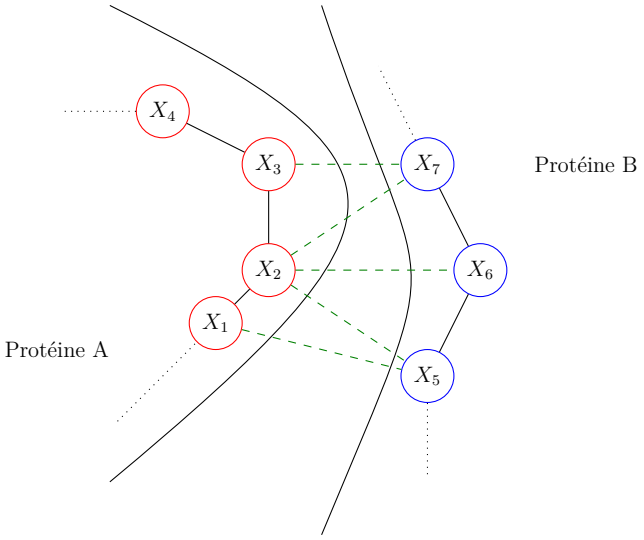


Fig. 3.10 – Complexe de protéines AB formé par les protéines A et B en contact par leur interface contenant respectivement les variables $\{X_1, X_2, X_3, X_4\}$ et $\{X_5, X_6, X_7\}$.

Deuxième partie

Calcul approximatif de la fonction de partition

Application au design de protéines

Chapitre 4

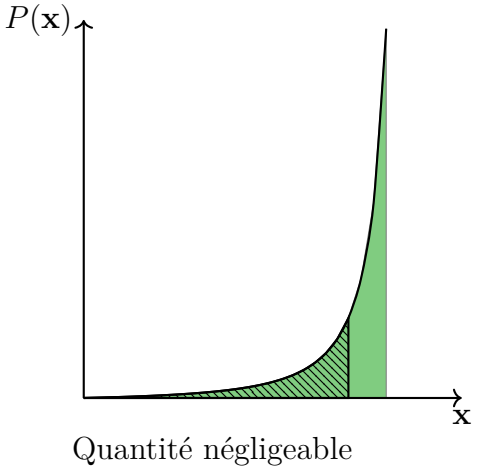
Approximation de la fonction de partition avec garantie déterministe

> "Although this may seem a paradox, all exact science is dominated by the idea of approximation."
>
> Bertrand Russell (1872-1970)

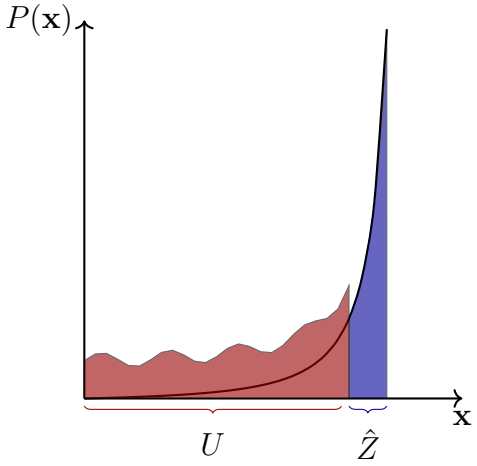
Nous avons vu dans la section 1.2 que le problème de calcul de la fonction de partition est classé #P-complet et dans la section 3.3 son rôle dans l'estimation de la constante d'affinité entre deux protéines. Théoriquement, une valeur exacte de Z offrirait l'estimation la plus fiable possible. Malheureusement, à cause de son temps de calcul exponentiel, cet objectif est souvent irréalisable.

Dans le chapitre 2, nous avons recensé différentes méthodes pour calculer Z allant de méthodes exactes à des méthodes sans garantie. Cependant, certains algorithmes, comme l'algorithme K^* présenté dans la sous-section 3.3.2 sont capables de fournir une approximation de Z tout en conservant une garantie sur l'écart entre approximation et valeur exacte.

Dans ce chapitre, nous allons expliquer et détailler trois algorithmes développés durant cette thèse. Le premier Z^*_ε est basé sur un Branch&Bound associé à un `DFS` adapté pour le comptage [8, 9]. Il utilise un majorant, resserré par les cohérences locales, afin d'élaguer des affectations négligeables. Le second `#HBFS`, utilise la recherche `HBFS` combinant `DFS` et `BFS` [10], un minorant et un majorant de Z, dans le but de fournir un encadrement de Z de plus en plus serré en fonction du temps.



(a) La fonction de partition est représentée par l'aire sous la courbe. La partie hachurée représente la quantité de potentiels que l'on pourrait négliger.



(b) L 'aire rouge représente la majoration U sur la quantité de potentiels négligés. L'aire en bleue représente la quantité de fonction de partition $\hat{Z}$ comptabilisée.

Fig. 4.1 – Tracés des potentiels $P(\mathbf{x})$ en fonction des conformations $\mathbf{x}$.

Enfin, `BTDZ` utilise une décomposition arborescente pour ne pas avoir à recalculer les parties conditionnellement indépendantes du modèle graphique.

4.1 Z^*_ε : élagage avec une garantie ε

L'algorithme Z^*_ε est adapté à des modèles graphiques pour lesquels une partie importante des contributions accumulées dans Z sont globalement négligeables. Il s'appuie sur l'hypothèse que les protéines ayant un repliement stable devraient fréquemment mener à des modèles de ce type. La majorité des algorithmes exacts essaient de détecter les affectations de probabilité 0 assez tôt dans l'arbre de recherche afin d'élaguer l'arbre de recherche et de les éliminer de la fonction de partition. L'exploitation des potentiels nuls peut déjà être très bénéfique. Toutefois, si nous pouvions également mettre de côté les potentiels globalement négligeables alors nous pourrions gagner un temps conséquent pour les modèles graphiques ayant une distribution de potentiels très concentrée, comme représentée dans la figure 4.1a.

En effet, si l'on considère que de nombreuses affectations ne contribuent globalement pas de façon significative à la fonction de partition, on peut simplement décider de retirer la contribution de ces affectations de la somme afin d'obtenir une approximation tout à fait acceptable. Cependant, pour ne pas souffrir des mêmes faiblesses qu'un algorithme sans garantie, il nous faut garder une information sur la quantité de potentiels que l'on a négligée. Afin d'appliquer cette idée , nous allons donc définir quatre quantités :

- Z la valeur exacte de la fonction de partition
- $\hat{Z}$ la valeur approchée de Z
- U une majoration sur la quantité totale de probabilités éliminées durant la recherche
- $U_b(\mathbf{x}_q)$ un majorant sur la fonction de partition du sous-arbre enraciné à un nœud n correspondant à l'affectation (partielle) $\mathbf{x}_q$.

La valeur approchée $\hat{Z}$ sera calculée sur un sous-ensemble $\mathbf{Q} \subset D_X$ d'affectations. Trivialement, on sait que la valeur exacte de Z sera bien compris entre $\hat{Z}$ et $\hat{Z}+U$ (Voir figure 4.1b). Nous avons alors :

$$Z = \sum_{\mathbf{x} \in D_X} P(\mathbf{x}) \quad ; \quad \hat{Z} = \sum_{\mathbf{x} \in \mathbf{Q}} P(\mathbf{x}) \quad ; \quad U \geq \sum_{\mathbf{x} \in D_X \setminus \mathbf{Q}} P(\mathbf{x})$$

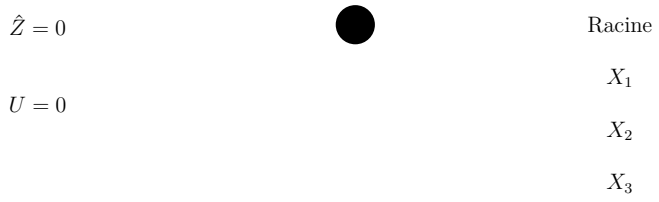
Si on assure que la quantité U est ε négligeable par rapport à $\hat{Z}$ c'est à dire que $U < \varepsilon \hat{Z}$ alors on obtient :

$$\hat{Z} \leq Z \leq \hat{Z} + U \quad \Leftrightarrow \quad \hat{Z} \leq Z \leq (1+\varepsilon)\hat{Z}$$

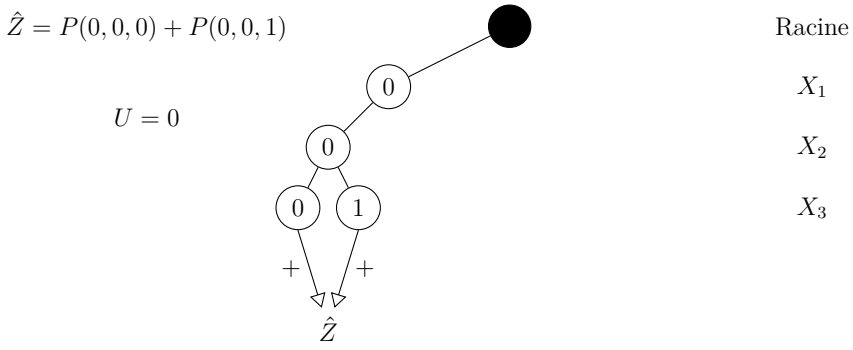
Et donc que $\hat{Z}$ est une ε-approximation de Z.

Dans son exploration de l'arbre de recherche, l'algorithme Z^*_ε [8, 9] se comportera à la manière d'un `DFS-B&B` (voir section 1.2). L'algorithme se déroule comme dans la figure 4.2. Afin d'accélérer la recherche et d'éviter de récolter les affections une à une, on ajoute une phase d'élimination `Sum-prod VE` (cf sous-section 2.1.1) de toutes variables d'un degré inférieur ou égal à 3 en *preprocessing* et à chaque branchement de valeur (à la volée) [56, 249]. On peut retranscrire ces étapes en un algorithme récursif (algorithme 4.1).

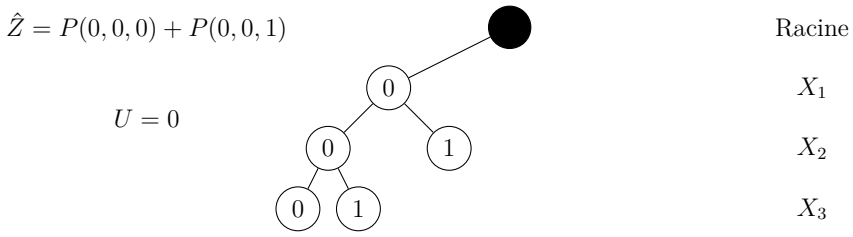
L'algorithme Z^*_ε dépend d'un majorant U_b. Ce majorant sera calculé à chaque branchement de valeur de l'arbre de recherche. Il est primordial de choisir un majorant efficace autant en qualité qu'en temps de calcul. En effet, un majorant plus resserré implique théoriquement plus d'élagages. Mais si le calcul de ce majorant est long, alors le temps gagné en éliminant des sous-arbres sera certainement contrebalancé par le temps de calcul du majorant.



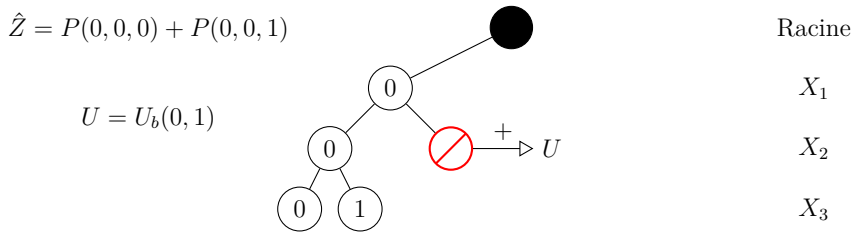
(a) On a la racine avec $\hat{Z} = 0$ et $U = 0$.



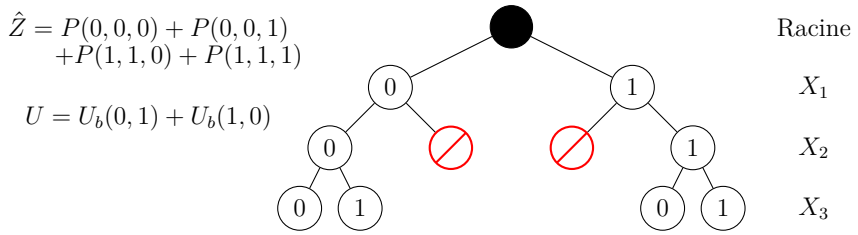
(b) On descend dans l'arbre et on ajoute les potentiels à $\hat{Z}$.



(c) On backtrack sur $(x_1 = 0)$, on branche $(x_2 = 1)$ et on vérifie : $U_b(0,1)+U < \varepsilon\hat{Z}$.



(d) Si l'invariant est vérifié alors on n'explore pas le sous-arbre et on ajoute $U_b(0)$ à U



(e) Si l'invariant est vérifié alors on n'explore pas le sous-arbre et on ajoute $U_b(1,0)$ à U

Fig. 4.2 – Recherche Z_ε^* sur un exemple simple de 3 variables contenant chacune deux valeurs. À chaque branchement de nœud, on vérifie l'invariant et on décide d'élaguer ou non.

Algorithme 4.1 : Algorithme de comptage approximatif Z^*_ε.

```
1  Fonction Z*_ε(x, X, Ẑ, U)
2      si X = ∅ alors
3          Ẑ ← Ẑ + P(x);
4      Choisir X_i ∈ X;
5      X ← X − {X_i};
6      pour x_i ∈ d_i faire
7          x ← x ∪ {x_i};
8          Elimination(X) ;        // Applique l'élimination de variables
9          Propagation(X) ;            // Applique les cohérences locales
10         si (U + U_b(x) ≤ εẐ) alors
11             U ← U + U_b(x);
12         sinon
13             Z*_ε(x, X, Ẑ, U);
14 U ← 0 ;
15 Ẑ ← 0 ;
16 Z*_ε(∅, X, Ẑ, U)) ;
17 retourner Ẑ, U ;
```

Majorer la fonction de partition

Nous avons créé deux majorants afin d'avoir le choix entre efficacité et temps de calcul.

Propriété 4.1.1. *Pour tout champ de Markov $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \Phi, \prod \rangle$ (cf. chapitre 1), on définit un premier majorant sur la fonction de partition Z appelé U_{b_1} tel que :*

$$U_{b_1} = \left(\prod_{\substack{\phi_S \in \Phi \\ |S|<2}} \sum_{\boldsymbol{x} \in D_X} \phi_S(\boldsymbol{x}) \right) \cdot \left(\prod_{\substack{\phi_S \in \Phi \\ |S| \geq 2}} \max_{\boldsymbol{x} \in D_X} \phi_S(\boldsymbol{x}) \right)$$

Preuve 4.1.0.1. *La démonstration découle de la définition de la fonction de partition Z :*

$$Z = \sum_{\boldsymbol{x} \in D_X} \left(\prod_{\phi_S \in \Phi} \phi_S(\boldsymbol{x}) \right)$$

$$Z = \sum_{\boldsymbol{x} \in D_X} \left(\prod_{\substack{\phi_S \in \Phi \\ |S|<2}} \phi_S(\boldsymbol{x}) \cdot \prod_{\substack{\phi_S \in \Phi \\ |S| \geq 2}} \phi_S(\boldsymbol{x}) \right)$$

$$Z = \sum_{\boldsymbol{x} \in D_X} \left(\prod_{\substack{\phi_S \in \Phi \\ |S|<2}} \phi_i(x_i) \cdot \prod_{\substack{\phi_S \in \Phi \\ |S| \geq 2}} \phi_S(\boldsymbol{x}) \right) \tag{4.1}$$

Trivialement, on sait que pour toute affectation $\boldsymbol{x} \in D_S$, on a $\phi_S(\boldsymbol{x}) \leq \max_{\boldsymbol{x} \in D_S} \phi_S(\boldsymbol{x})$. En appliquant cela au terme de droite de l'équation 4.1 et en exploitant le fait que le nouveau terme ne dépend plus de $\boldsymbol{x}$, alors :

$$Z \leq \left(\sum_{\boldsymbol{x} \in D_X} \prod_{\substack{\phi_S \in \Phi \\ |S|<2}} \phi_i(x_i) \right) \cdot \left(\prod_{\substack{\phi_S \\ |S| \geq 2}} \max_{\boldsymbol{x} \in D_X} \phi_S(\boldsymbol{x}) \right)$$

Puisque l'ensemble $\{\phi_S, |S| < 2\}$ ne contient que des fonctions unaires (ou zéro-aire dans le cas d'un CFN avec $c_\varnothing$), la distributivité nous permet d'échanger les symboles somme et produit, et on obtient la quantité désirée. □

Le premier majorant U_{b_1} ne prend en compte que les fonctions unaires du modèle graphique. Son calcul est assez rapide mais il voit sa précision faiblir si les fonctions

potentielles d'arité supérieure à 1 possèdent une contribution importante dans le calcul de Z. En utilisant un arbre recouvrant de potentiel maximal, plus connu sous le nom de *Maximum Spanning Tree*, on pourra prendre en compte les coûts binaires les plus importants. Grâce à une élimination de variables sur $T' = T \cup \{\phi_S \in \Phi \ : \ |S| < 2\}$, on peut obtenir la fonction de partition $Z_{T'}$ de l'arbre T' de manière exacte et cela en un temps polynomial en $O(nd^2)$. Ensuite, il suffit de multiplier $Z_{T'}$ par $\prod_{\phi_S \in \Phi \setminus T'} \max_{\mathbf{x} \in D_X} \phi_S(\mathbf{x})$ afin d'obtenir un majorant plus précis que U_{b_1} noté U_{b_t}.

Propriété 4.1.2. *Pour tout champ de Markov $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \Phi, \prod \rangle$ (cf. chapitre 1), on définit un second majorant sur la fonction de partition :*

$$U_{b_t} = \underbrace{\left(\sum_{\boldsymbol{x} \in D_X} \prod_{\phi_S \in T'} \phi_S(\boldsymbol{x}) \right)}_{\text{élimination de variables}} \cdot \left(\prod_{\phi_S \in \Phi \setminus T'} \max_{\boldsymbol{x} \in D_X} \phi_S(\boldsymbol{x}) \right)$$

Preuve 4.1.0.2. *La preuve est essentiellement la même que la précédente. Il faut juste remplacer $\{\phi_S, |S| < 2\}$ et son complémentaire $\{\phi_S, |S| \geq 2\}$ par l'ensemble $\{\phi_S \in T'\}$ (et respectivement son complémentaire). Le premier terme peut être calculé en $O(nd^2)$ temps en utilisant l'élimination de variables.* □

L'arbre couvrant de potentiel maximal sur lequel U_{b_t} repose est calculé à partir de l'algorithme de Prim [250–252]. Nous avons utilisé la médiane des potentiels de chaque fonction binaire pour mettre un poids sur les arêtes du graphe. L'algorithme de Prim s'exécute en $O(e \log n)$.

Même si, en théorie U_{b_t} est qualitativement meilleure que U_{b_1}, elle est aussi plus chère à calculer. En effet, à chaque branchement de valeur, les poids médians sur les arêtes ne sont plus les mêmes donc l'arbre couvrant de potentiel maximal pourrait également changer. Par conséquent, on devrait le recalculer à chaque branchement de valeur. En pratique, nous avons fait le choix de calculer un seul arbre couvrant de potentiels maximal à la racine dont on se servira tout au long de la recherche, perdant éventuellement en qualité mais gagnant en vitesse.

Renforcement par cohérences locales

Dans la section 1.4, nous avons vu que les cohérences locales créent du déterminisme et dans un même temps font croître le minorant $c_\varnothing$. Nous avons utilisé le déterminisme créé par les cohérences locales pour simplifier le calcul des deux bornes U_{b_1} et U_{b_t} et le minorant $c_\varnothing$ pour les renforcer. D'une part, dans un champ

de Markov binaire, l'arc cohérence garantit que $\max_{\mathbf{x} \in D_X} \phi_S(\mathbf{x}) = 1$ et d'autre part si on note $p_\varnothing$ l'équivalent de $c_\varnothing$ transformé par l'équation 1.1, alors on obtient :

$$U_{b_1} = p_\varnothing \prod_{\phi_i \in \boldsymbol{\Phi}} \sum_{x_i \in \mathbf{X}} \phi_i(x_i)$$

$$U_{b_t} = p_\varnothing \sum_{\mathbf{x} \in D_X} \prod_{\phi_S \in T'} \phi_S(\mathbf{x})$$

Ces bornes se calculent en $O(nd)$ et $O(nd^2)$ respectivement mais leur principal défaut réside dans le fait qu'elles ne portent que sur une partie des potentiels. Cependant les cohérences locales (cf section 1.4) peuvent aspirer une part importante des coûts des fonctions d'arité supérieure à 2 dans les coûts unaires puis dans $c_\varnothing$. Par conséquent, les cohérences locales permettent de renforcer ces majorants.

4.1.1 Comparaison Z^*_ε versus VEC, miniC2D, Ace et Cachet

Dans le but de vérifier l'efficacité du système d'élagage de Z^*_ε, nous nous sommes comparés à d'autres algorithmes de comptage pondéré exacts (voir section 2.1) sur deux types de benchmarks [9]. Le premier algorithme est implémenté dans VEC [56] et utilise l'élimination de variables `Sum-Prod VE` et l'exploitation d'un ensemble coupe-cycle qu'il faut énumérer. Le second Cachet [55] est basé sur les solveurs SAT et le *caching*. Les deux derniers sont des algorithmes basés sur la compilation de connaissances et sont implémentés dans Ace et miniC2D. Quant à notre algorithme, Z^*_ε a été implémenté dans le programme toulbar2 disponible en open source [†].

Le premier benchmark est un ensemble d'instances composées de champs de Markov de cinq complexes de protéines (code PDB : `1acb, 1cbw ,1jrh, 1ppf, 1r0r`) et de leurs mutants (voir section 3.2). Les structures cristallographiques des cinq complexes de protéines (non mutées) ont été extraites de la Protein Data Bank. Les atomes d'hydrogènes ont été ajoutés en utilisant le programme `tleap` du logiciel de modélisation et de simulation de biomolécules, *Amber 14* [253]. Afin de relâcher la structure 3D, nous avons effectué 1000 étapes de minimisations des coordonnées des atomes du complexe, comprenant 500 étapes de gradient descendant (méthode de la plus grande pente) et 500 étapes de gradient conjugué, en utilisant le module *sander* d'*Amber 14*.

Afin de ne pas avoir des champs de Markov intraitables mais de tout de même prendre en compte l'interaction entre les protéines, nous avons considéré comme

†. https://mulcyber.toulouse.inra.fr/projects/toulbar2/

flexibles, seulement les acides aminés présents à l'interface du complexe. Les mutants ont été sélectionnés selon la base de donnée de SKEMPI [193]. Tout acide aminé d'une protéine présent à moins de 12 Å d'un acide aminé de l'autre protéine est considéré comme faisant partie de l'interface. Nous avons utilisé le champ de force du programme OSPREY [135] afin de calculer les énergies décomposables par paires d'acides aminés et générer les matrices d'énergie (voir section 3.2.2) des protéines sauvages et des mutants. Enfin, une traduction des matrices d'énergie en format *UAI* à été nécessaire pour VEC et toulbar2 puis une étape d'encodage en SAT pondéré [9] pour miniC2D et Cachet (voir sous-section 2.1.2). Ace intègre sont propre encodage en SAT pondéré.

Nous avons ainsi effectué nos tests sur un total de 349 mutants issus des cinq complexes de protéines. Le système le plus gros possède 22 variables avec un domaine maximal de 34 valeurs.

Tout d'abord, nos deux majorants U_{b_1} et U_{b_t}, renforcés par la cohérence locale `EDAC`, ont été été testés sur les 349 instances (voir figure 4.3). Comme attendu, le majorant U_{b_t} est plus proche de la valeur exacte de Z.

Puis nous avons comparé le temps mis par Z^*_ε pour calculer Z à une garantie $\varepsilon = 10^{-3}$ avec les deux majorants renforcés, soit par `EDAC` [22], soit par `VAC` [37]. Le résultat est représenté sur un cactus plot dans la figure 4.4. La meilleure combinaison est la plus légère U_{b_1}+ `EDAC`. U_{b_1} est plus rapide à calculer que U_{b_t}, il en va de même pour `EDAC` par rapport à `VAC`. Même s'il à déjà fait ses preuves `VAC` [5] ne semble pas rentable sur des systèmes trop petits.

Par la suite, nous avons utilisé Z^*_ε avec U_{b_1}+ `EDAC` et nous l'avons comparé avec VEC, miniC2D et Cachet (figure 4.5). miniC2D et Ace n'ont pas été représentés sur le cactus plot. Minic2d n'a pas pu lancer 294 des 349 instances à cause d'une limitation en taille lors de l'encodage en $\#SAT$ pondéré. De plus, sur les 55 instances restantes, les plus petites, il n'a pu résoudre que 7 des instances en moins d'une heure. En ce qui concerne Ace, il a résolu 55 instances, toutes les autres ont dépassé la limite en mémoire de 60Gb. Il est clair que Z^*_ε surpasse les autres outils sur les instances d'interaction protéine-protéine.

Pour voir comment Z^*_ε se comporte sur d'autres types d'instances que des complexes de protéines, nous avons utilisé un second benchmark composé d'instances provenant de la compétition PIC'2011 [†]. Nous avons utilisé la même valeur $\varepsilon = 10^{-3}$ et nous nous sommes comparés aux trois mêmes outils de comptage que précédemment (voir tableau 4.1). Les résultats montrent clairement qu'en dépit du manque de sophistication de Z^*_ε autant au niveau *caching* qu'au niveau exploitation de la structure et de l'indépendance, l'algorithme peut, dans plusieurs cas, surpasser

†. http ://www.cs.huji.ac.il/project/PASCAL

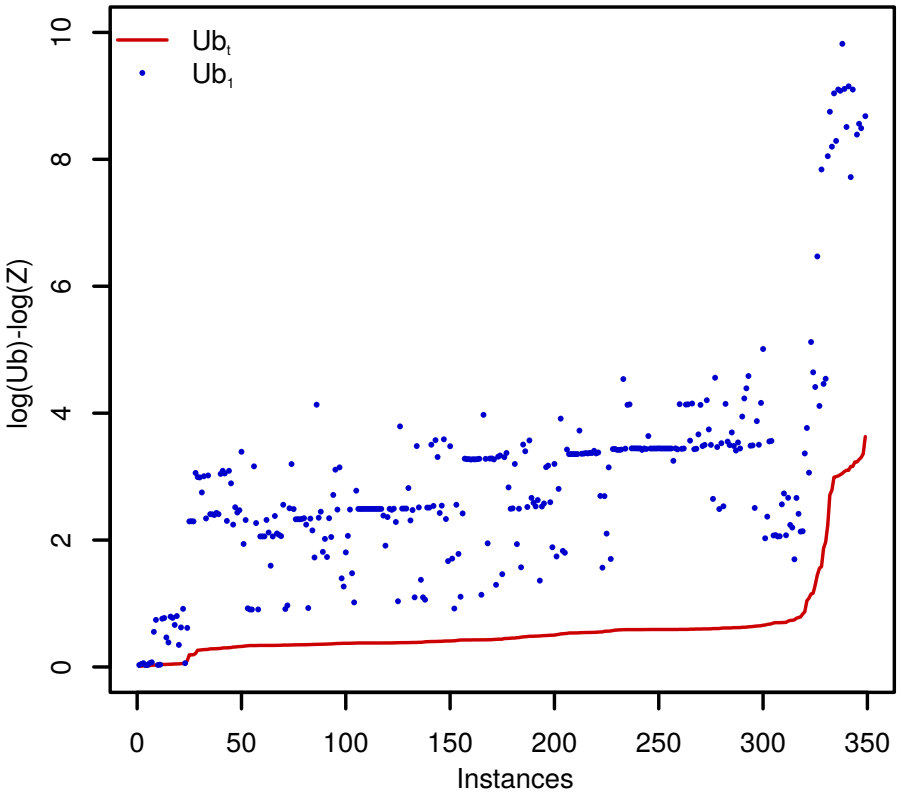


Fig. 4.3 – Écart $\log(U_b) - \log(Z)$ pour les majorants U_{b_1} (points bleus) et U_{b_t} (trait rouge) renforcés par la cohérence locale `EDAC`. Les instances sur l'axe x sont rangées par ordre croissant dans la taille de l'écart pour le majorant le plus fort U_{b_t}

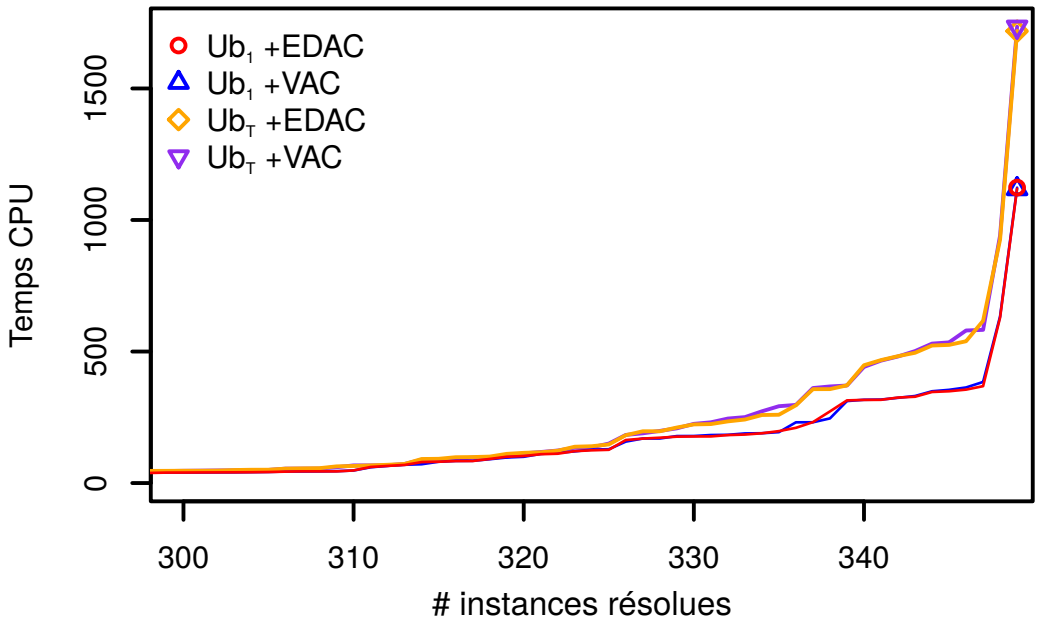


Fig. 4.4 – Cactus plot du temps d'exécution de Z^*_ε avec les quatre combinaisons de majorant. Un point (x, y) indique le nombre x d'instance(s) résolue(s) si la limite de temps imposée sur chaque système est de y seconde(s).

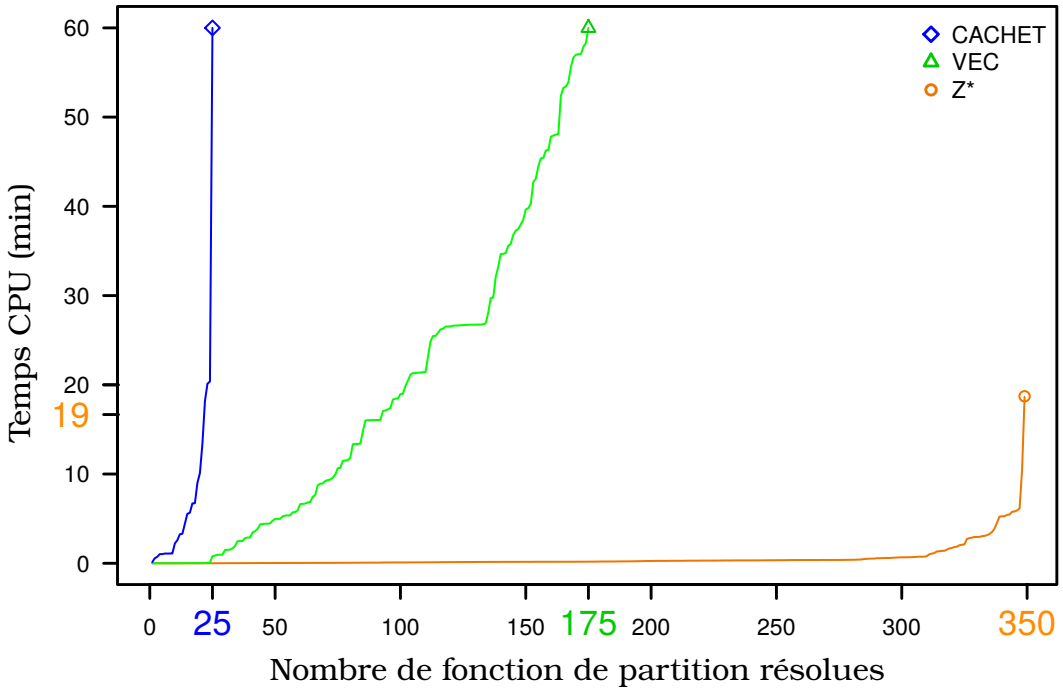


Fig. 4.5 – Cactus plot du temps d'exécution de VEC, Cachet et Z_ε^* avec $U_{b_1}+$ EDAC comme majorant. Un point (x, y) indique le nombre x d'instance(s) résolue(s) si la limite de temps imposée sur chaque système est de y seconde(s).

ses compétiteurs. VEC et Ace semblent préférer la catégorie des grilles. Ceci est probablement dû aux domaines booléens et à la petite largeur d'arbre des grilles considérées. On peut voir que Cachet s'est retrouvé dominé dans toutes les catégories.

Instance	Z_ε^*	minic2d	ace	vec	cachet
smokers_10	< **0.01**	0.66	< **0.01**	< **0.01**	0.26
smokers_20	< **0.01**	312.83	< **0.01**	< **0.01**	332.17
rbm_20	7.46	T	**3.376**	16.17	941.14
rbm_ferro_20	< **0.01**	T	3.24	16.18	893.84
rbm_21	13.75	T	**6.854**	34.35	2041.64
rbm_ferro_21	< **0.01**	T	6.87	34.17	1975.78
rbm_22	33.75	T	**14.411**	72.78	T
rbm_ferro_22	< **0.01**	T	14.42	72.43	T
grid10x10.f10	66.32	< **0.01**	< **0.01**	< **0.01**	T
grid20x20.f10	T	T	**13.32**	2104.57	T
grid20x20.f15	T	T	**13.67**	2099.24	T
grid20x20.f2	T	T	**13.60**	2107.92	T
grid20x20.f5	T	T	**13.61**	2102.32	T
GEOM30a_3	< **0.01**	< **0.01**	< **0.01**	< **0.01**	2.67
GEOM30a_4	7.66	78.60	< **0.01**	< **0.01**	43.77
GEOM30a_5	67.48	368.36	< **0.01**	< **0.01**	405.38
GEOM40_2	< **0.01**	< **0.01**	< **0.01**	< **0.01**	< **0.01**
GEOM40_3	< **0.01**	< **0.01**	< **0.01**	< **0.01**	3.62
GEOM40_4	1.42	0.58	< **0.01**	< **0.01**	616.59
GEOM40_5	12.67	12.80	< **0.01**	< **0.01**	828.50
myciel5g_3	**37.20**	T	M	T	T
myciel5g_4	M	T	M	T	T
myciel5g_5	M	T	M	T	T
queen5_5_3	367.20	T	**83.00**	945.20	T
queen5_5_4	**2423.67**	T	M	T	T

TABLE *4.1* – Temps d'exécution en seconde des instances provenant de la compétition PIC'2011. Il y a trois différentes catégories : réseaux bayésiens dynamiques (DBN, rbm), Grilles, et problèmes de coloriage. M : Mémoire insuffisante ($60GB$), T : Temps limite dépassé ($3600s$). Nous avons mis en gras les meilleurs temps.

L'algorithme Z_ε^* s'est montré plus performant que d'autres algorithmes autant sur des instances d'interactions protéine-protéine que sur des instances issues d'un

challenge officiel. Cependant ces algorithmes étant exacts, ils ne fournissent pas exactement la même garantie que Z^*_ε. Cela montre l'efficacité de l'élagage de notre algorithme et confirme l'hypothèse qu'un nombre important de potentiels est négligeable. Mais qu'en est-il de l'efficacité de Z^*_ε face à un autre outil de comptage donnant une ε garantie ? Nous avons donc comparé Z^*_ε à K^*, développé par le groupe de Bruce Donald.

4.1.2 Comparaison Z^*_ε versus K^*

K^* calcule une approximation de la constante d'affinité K_a en se basant sur des ε approximations de la fonction de partition du complexe et des partenaires (voir la sous-section 3.3.2 pour plus de détails). Nous avons fixé la précision sur $\varepsilon = 10^{-3}$ pour Z^*_ε et K^*. L'algorithme K^* est disponible en open source dans le programme OSPREY [†]. Nous avons choisi de nous comparer à la dernière version stable, la version 2.0.

K^* propose une option (paramétrée par un seuil σ) permettant de ne pas avoir à estimer la totalité des affinités de liaison des mutants. En effet, cette option compare la valeur de la meilleure estimation de l'affinité de liaison trouvée à un majorant sur l'affinité de liaison qu'il est en train de calculer, si le majorant est en dessous de la meilleure estimation alors le calcul est abandonné [141]. Nous n'avons pas utilisé cette option car nous voulions nous comparer en temps sur la totalité des mutants d'un complexe. Une comparaison entre K^* et Z^*_ε a déjà été publiée dans [8] mais elle ne comportait que très peu d'instances. Afin de confirmer ces résultats, nous avons appliqué K^* et Z^*_ε sur 1003 mutants de 30 complexes de protéines (voir tableau 4.2), sélectionnés dans la base de données SKEMPI [193] .

Les complexes de protéines ont été modélisés comme dans la section précédente (voir sous-section 4.1.1). Nous avons aussi ajouté un *cut-off* de 100 kcal.mol^{-1} sur l'énergie unaire d'un rotamère. C'est à dire que si l'énergie d'un rotamère dépasse 100 kcal.mol^{-1}, il sera considéré comme étant en clash avec un autre rotamère et il sera ignoré. Le problème le plus gros, *1cho*, inclut 162 mutants et donc un total de $(162 + 1) \times 3 = 489$ instances. Ce complexe possède 22 variables, la première protéine en a 12 et la seconde 10, les trois champs de Markov ont un domaine maximal de 34 valeurs.

Nous avons représenté le temps de calcul total de tous les K_a par complexe dans la figure 4.6. On peut y voir que Z^*_ε est beaucoup plus rapide que K^*. En effet, K^*ne termine que 18 prédictions de constantes d'affinité dans les 72 heures imparties alors que Z^*_ε termine la totalité en moins de 56 heures. Cette différence d'efficacité

†. http://www.cs.duke.edu/donaldlab/osprey.php

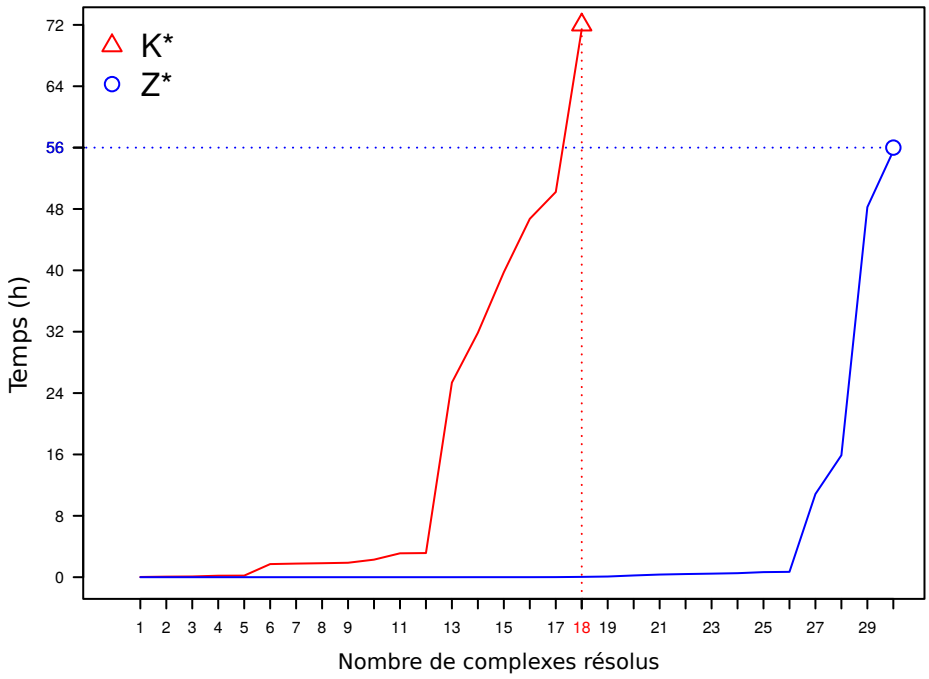


Fig. 4.6 – Cactus plot du temps d'exécution de K^* et Z^*_ε avec $U_{b_1}+$ `EDAC` comme majorant. Un point (x, y) indique le nombre x d'instance(s) résolue(s) si la limite de temps imposée sur chaque système est de y seconde(s). Le temps limite est de 72h et la mémoire est limitée à 60 Go.

est probablement dû au fait que le majorant de K^* est faible et donc que A^* peine à énumérer les conformations. De plus, K^* doit redémarrer son énumération si le `DEE` a élagué trop de conformations.

PDB	Partenaire A	Partenaire B	# Acides aminés			res	# Mutants
			AB	A	B		
1acb	Bovine α-chymotrypsin	Eglin c	304	241	63	2.00	6
1a22	Human growth hormone	hGH binding	372	180	192	2.60	16
1cbw	Bovine α-chymotrypsin	BPTI	297	239	58	2.60	18
1cho	Bovine α-chymotrypsin	Turkey ovomucoid 3rd domain	291	238	53	1.80	163
1cse	Subtilisin Carlsberg	Eglin c	337	274	63	1.20	6
1dvf	IgG1-kappa D1.3 Fv	E5.2 Fv	451	223	228	1.90	17
1eaw	Membrane-type serine protease	BPTI	297	241	56	2.93	10
1fcc	IgG1 MO61 Fc	Protein A/Z	262	206	56	3.20	9
1fc2	Protein A/Z	IgG1 MO61 Fc	251	44	207	2.80	12
1ffw	Chemotaxis protein CheY	Chemotaxis protein CheA	196	128	68	2.70	9
1f47	FtsZ fragment	ZipA	161	17	144	1.95	11
1h9d	AML1 Runx1 Runt domain	Core-binding factor beta	250	125	125	2.60	7
1iar	Interleukin-4	Interleukin-4 receptor	317	129	192	2.30	22
1jrh	mAbs A6	Interferon gamma receptor	442	347	95	2.80	22
1jtg	TEM-1 β-lactamase	BLIP	427	262	165	1.73	16
1ktz	Transforming growth factor beta 3	TGF-beta type II receptor	188	82	106	2.15	13
1ppf	Human leukocyte elastase	Turkey ovomucoid 3rd domain	274	218	56	1.80	163
1r0r	Subtilisin Carlsberg	Turkey ovomucoid 3rd domain	325	274	51	1.10	163
1s1q	Tumor susceptibility gene 101 protein	Ubiquitin	213	141	72	2.00	7
1tm1	Subtilisin BPN	Chymotrypsin inhibitor 2	345	281	64	1.70	26
1vfb	IgG1-kappa D1.3 Fv	HEW Lysozyme	352	223	129	1.80	16
1xd3	UCH-L3	Ubiquitin	304	229	75	1.45	11
2ftl	Bovine trypsin	BPTI	280	222	58	1.62	17
2j0t	MMP1 Interstitial collagenase	Metalloproteinase inhibitor 1	285	161	124	2.54	14
2o3b	Nuclease	Sugar-non-specific nuclease inhibitor	376	241	135	2.30	8
2pcb	CYTOCHROME C PEROXIDASE	CYTOCHROME C	398	294	104	2.80	7
2sic	SUBTILISIN BPN'	SSI	382	275	107	1.80	11
3bk3	Bone morphogenetic protein 2	Crossveinless 2	171	104	67	2.70	12
3sgb	Streptomyces griseus proteinase B	Turkey ovomucoid 3rd domain	235	185	50	1.80	163
4cpa	Carboxypeptidase A	Metallocarboxypeptidase inhibitor	344	307	37	2.50	8

TABLE *4.2* – Complexes de protéines issus de la base de données SKEMPI. De gauche à droite : Code PDB du complexe A-B, Nom des protéines A et B, Nombre d'acides aminés de AB, A et B, Résolution de la structure 3D (Å), Nombre de mutants.

Z^*_ε a montré son efficacité sur les instances issues de complexes de protéines. Néanmoins, il ne possède pas la capacité de fournir un résultat avant que l'algorithme ne s'achève. Les algorithmes permettant de donner une valeur et une garantie à tout moment sont classés comme algorithme "***anytime***". De tels algorithmes sont utiles en pratique car ils fournissent tout le temps un résultat avec l'incertitude correspondante. Dans l'optique de satisfaire à cette propriété *anytime*, nous avons développé le premier algorithme de comptage combinant cohérences locales, élimination de variables, calcul de minorant et de majorant et une recherche mixant `DFS` et `BFS`.

4.2 `#HBFS` : recherche arborescente hybride

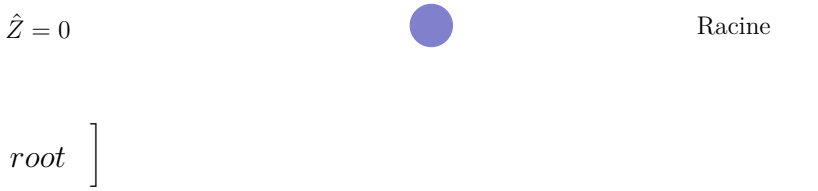
L'algorithme Z^*_ε nécessite la spécification *a priori* d'un ε. L'algorithme ne s'arrête pas avant d'avoir obtenu cette garantie. Dans le but d'avoir un algorithme permettant d'obtenir une garantie à n'importe quelle moment, nous avons adapté l'algorithme de recherche *Hybrid Best First Search* (`HBFS`) [10] développé pour l'optimisation en un algorithme qui puisse calculer la fonction de partition Z. Nous avons nommé cet algorithme `#HBFS`.

Le principe de recherche de `#HBFS` est assez simple (Figure 4.7). A l'instar d'un `DFS` qui plonge dans l'arbre sans jamais remonter à la racine, `#HBFS` effectue plusieurs `DFS` en partant de nœuds contenus dans une liste de nœuds ouverts notée *open*.

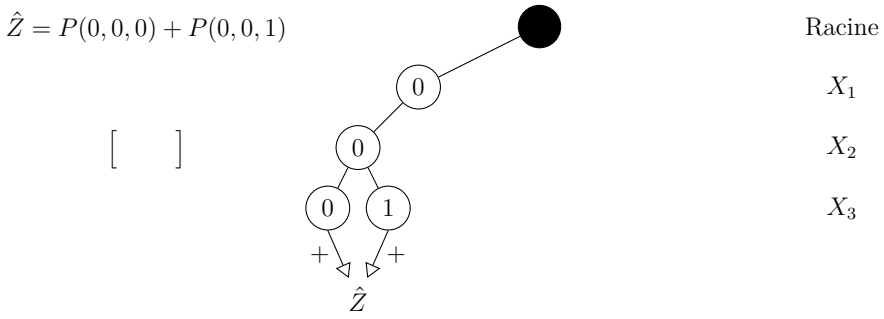
Chaque `DFS` n'a qu'un nombre limité de backtrack(s). L'algorithme débute à la racine puis il se lance dans une recherche en profondeur jusqu'à épuisement de son carburant de backtrack(s). Dès lors, le `DFS` s'arrête et l'appel récursif insère chaque nœud des branches inexplorées dans *open* pendant la remontée, calculant de surcroît un majorant et un minorant sur la fonction de partition du sous-arbre correspondant. Enfin selon une heuristique prédéfinie, il choisit, retire et lance un `DFS` à partir d'un nœud de *open*. Durant la recherche, l'algorithme ajoute au compte courant $\hat{Z}$ chaque potentiel trouvé sur les feuilles.

Si l'on **somme les minorants** (resp. majorants) stockés dans la liste *open*, avec la quantité $\hat{Z}$, alors on obtient un minorant (resp. majorant) global sur Z.

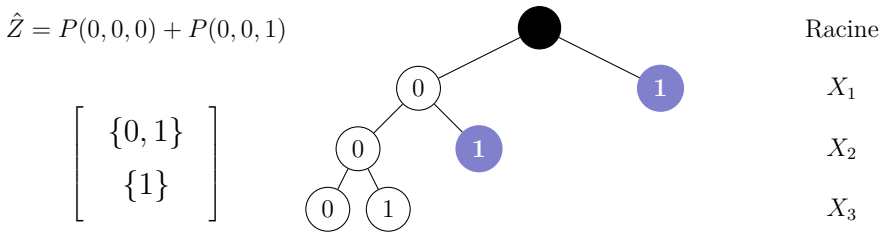
Propriété 4.2.1. *Soit $\mathcal{M}$ un champ de Markov. Soit $\mathcal{O}$ la liste open de `#HBFS`. Soit $\hat{Z}$ la quantité de potentiels déjà récoltée. Soient $z_{lb}(n)$ un minorant et $z_{ub}(n)$ un majorant de la fonction de partition du sous-arbre enraciné en n. Alors on peut*



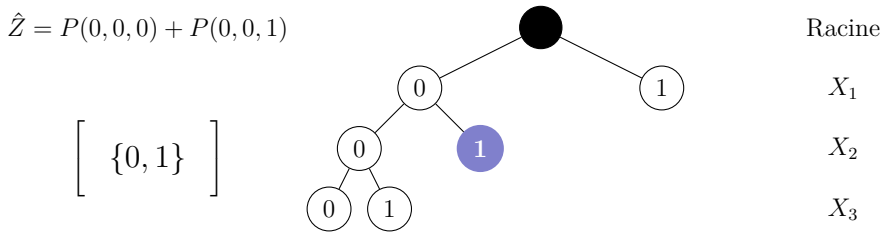
(a) On commence avec une liste de nœuds ouverts contenant la racine



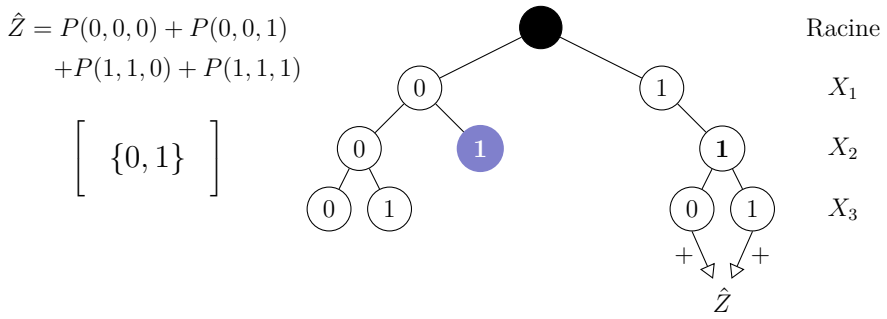
(b) On retire la racine de la liste de nœuds ouverts. On plonge dans l'arbre en effectuant un `DFS` et on récolte les potentiels.



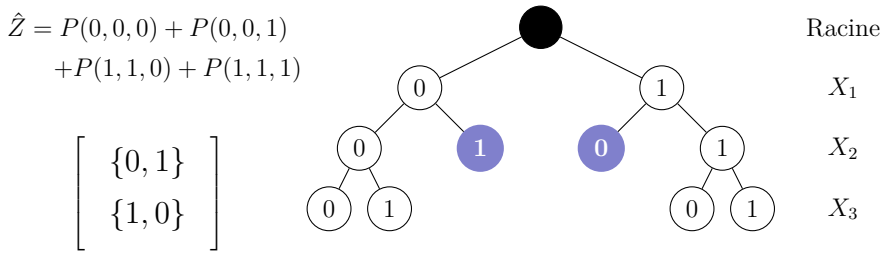
(c) On remonte à la racine et on ajoute les premiers nœuds des branches laissées pendantes dans la liste de nœuds ouverts.



(d) Selon l'heuristique, on choisit et on retire un nœud de la liste de nœuds ouverts.



(e) On plonge dans l'arbre en réalisant un `DFS` et on récolte les potentiels.



(f) On remonte à la racine et on ajoute les premiers nœuds des branches laissées pendantes dans la liste de nœuds ouverts...

Fig. 4.7 – Sharp Hybrid Best First Search : `#HBFS`. Les nœuds en bleu sont les nœuds appartenant à la liste *open*, représentée à gauche de l'arbre.

---

encadrer la fonction de partition Z du champ de Markov $\mathcal{M}$ par :

$$\hat{Z} + \sum_{n \in \mathcal{O}} z_{lb}(n) \;\leq\; Z \;\leq\; \hat{Z} + \sum_{n \in \mathcal{O}} z_{ub}(n) \tag{4.2}$$

Démonstration. On sait déjà que $\hat{Z}$ est un minorant de Z. Soit $z(n)$ la fonction de partition d'un sous-arbre enraciné en un nœud n. Alors on a :

$$Z = \hat{Z} + \sum_{n \in \mathcal{O}} z(n)$$

De plus, on sait que $z_{lb}(n) \;\leq\; z(n) \;\leq\; z_{ub}(n)$. Alors on obtient :

$$z_{lb}(n) + \hat{Z} \qquad \leq \qquad z(n) + \hat{Z} \qquad \leq \qquad z_{ub}(n) + \hat{Z}$$

□

A l'aide de l'équation 4.2, on peut obtenir une garantie ε au moment où l'algorithme s'arrête, ou inversement l'algorithme pourra s'arrêter lorsque qu'une garantie ε sera atteinte. Il suffira de résoudre l'équation suivante :

$$\varepsilon + 1 = \frac{\hat{Z} + \sum\limits_{\mathbf{x} \in \mathcal{O}} z_{ub}(\mathbf{x})}{\hat{Z} + \sum\limits_{\mathbf{x} \in \mathcal{O}} z_{lb}(\mathbf{x})}$$

Tout comme Z^*_ε, le minorant et majorant sont améliorés par les cohérences locales et le comptage pondéré par l'élimination de variables `Sum-Prod` à la volée et en *preprocessing*.

Lorsqu'un nœud est placé dans la liste *open*, au lieu de mémoriser la totalité du modèle graphique, `#HBFS` procède comme `HBFS` et mémorise seulement les décisions prises (assignation, élimination). Il les rejoue au moment de leur extraction pour obtenir un modèle graphique équivalent [10]. Afin de ne pas avoir un taux de décisions redondantes trop conséquent, l'algorithme tente de garder le ratio τ du nombre de décisions rejouées R sur le nombre total de nœud N dans un intervalle fixe $[\alpha, \beta]$. Pour cela, le nombre de backtracks alloué à une `DFS` est incrémentalement mis à jour et limité par une variable globale M (algorithme 4.2), comme dans l'algorithme `HBFS`.

4.2.1 Dynamique de recherche selon l'heuristique

Afin d'étudier l'influence du choix de nœuds dans la liste *open* sur la recherche arborescente, nous avons testé sept heuristiques différentes sur quelques

Algorithme 4.2 : Algorithme `#HBFS`. b : nombre actuel de backtracks, k : limite de backtracks pour un `DFS`, M : nombre limite de backtracks. Le ratio $\tau = \frac{R}{N}$ est gardé dans l'intervalle $[\alpha, \beta]$.

```
1  Fonction DFS(x, X, Ẑ, b)
2      si X = ∅ alors
3          Ẑ ← Ẑ + P(x);
4      sinon
5          Choisir X_i ∈ X ;
6          X ← X − {X_i};
7          pour x_i ∈ d_i faire
8              x ← x ∪ {x_i};
9              Elimination(X) // SumProd VE
10             Propagation(X) // Cohérences locales
11             si b ≤ k alors
12                 DFS(x, X, Ẑ, b) ;
13                 b ← b + 1 ;
14             sinon
15                 z_lb ← LowerBoundZ(x) ;    // Calcul le minorant pour x
16                 z_ub ← UpperBoundZ(x) ;    // Calcul le majorant pour x
17                 Open ← {x, z_lb, z_ub}
18 Fonction #HBFS(X)
19     z_lb ← LowerBoundZ(∅) ;
20     z_ub ← UpperBoundZ(∅) ;
21     Open ← [(∅, z_lb, z_ub)] ;
22     tant que Open ≠ ∅ ou (Ẑ+Z_UB)/(Ẑ+Z_LB) ≤ ε + 1 faire
23         Choisir et retirer x ∈ Open ;
24         restore(x, X) ;              // Rejoue les décisions et Màj de R
25         DFS(x, X, Ẑ, 0) ;    // DFS à partir de x pour k backtrack(s)
26         Z_LB ← Σ_{x∈Open} z_lb ;
27         Z_UB ← Σ_{x∈Open} z_ub ;
28         si R > 0 alors
29             si τ > β and k ≤ M alors
30                 k = k × 2;
31             si τ < α and k ≥ 2 alors
32                 k = k/2;
33     retourner Ẑ + Z_LB, Ẑ + Z_UB ;
```

instances tirées d'une modélisation d'interactions protéine-protéine à partir de matrices d'énergies de Rosetta (voir section 5 pour plus d'information).

Heuristiques

h_0	plus petit $c_\varnothing$	Amasse les probabilités les plus hautes
h_1	plus petit ε	Amasse les probabilités les plus sûres
h_2	plus grand ε	Réduit l'incertitude
h_3	plus grand z_{ub}	Amasse la plus grosse masse de probabilités
h_4	plus grand z_{lb}	Amasse la plus grosse masse de probabilités
h_5	plus grand $z_{lb} + z_{ub}$	Amasse la plus grosse masse de probabilités
h_6	plus grand $z_{ub}^2 - z_{lb}^2$	Réduit l'incertitude et amasse la plus grosse masse de probabilités

TABLE *4.3* – Tableau récapitulatif des heuristiques de choix de nœuds pour l'algorithme `#HBFS`.

L'heuristique h_0 est la fonction de sélection par défaut de `HBFS` [10], elle choisit le nœud avec le minorant $c_\varnothing$ le plus petit afin d'orienter la recherche vers une solution de coût minimal (et donc de potentiel maximal). Lorsque l'heuristique fournit le même résultat pour deux nœuds différents alors elle sélectionne le nœud le plus profond dans l'arbre de recherche (ce qui sera aussi notre stratégie). Cette heuristique a été développée dans le cadre de l'optimisation. Nous avons donc décidé de tester différentes heuristiques mettant en jeu z_{lb} et z_{ub} afin de peut-être trouver une heuristique plus performante que h_0.

Un détail des heuristiques est disponible dans le tableau 4.3. L'heuristique h_1 a été développée dans l'idée d'explorer les nœuds contenant le moins d'incertitude *a contrario* de h_2. L'heuristique h_3 choisit le plus grand majorant et prend en compte les coûts unaires avec U_{b_1}. h_4 a été développé par dualité à h_3 et choisit le minorant le plus grand alors que h_5 essaie de prendre en compte à la fois le minorant et le majorant en calculant la moyenne des deux. h_3, h_4 et h_5 ont tous pour but d'amasser le plus de masse de probabilités lors d'un `DFS`. La dernière heuristique a été réfléchie afin d'explorer les sous-arbres possédant à la fois le plus d'incertitude mais aussi le plus de masse de probabilité en choisissant le plus grand $(z_{ub} + z_{lb})(z_{ub} - z_{lb}) = z_{ub}^2 - z_{lb}^2$. Nous avons testé les sept heuristiques, ainsi que Z_ε^*, sur trois complexes de protéine `1ahw, 1cho` et `1ktz`, et pour chacun, trois

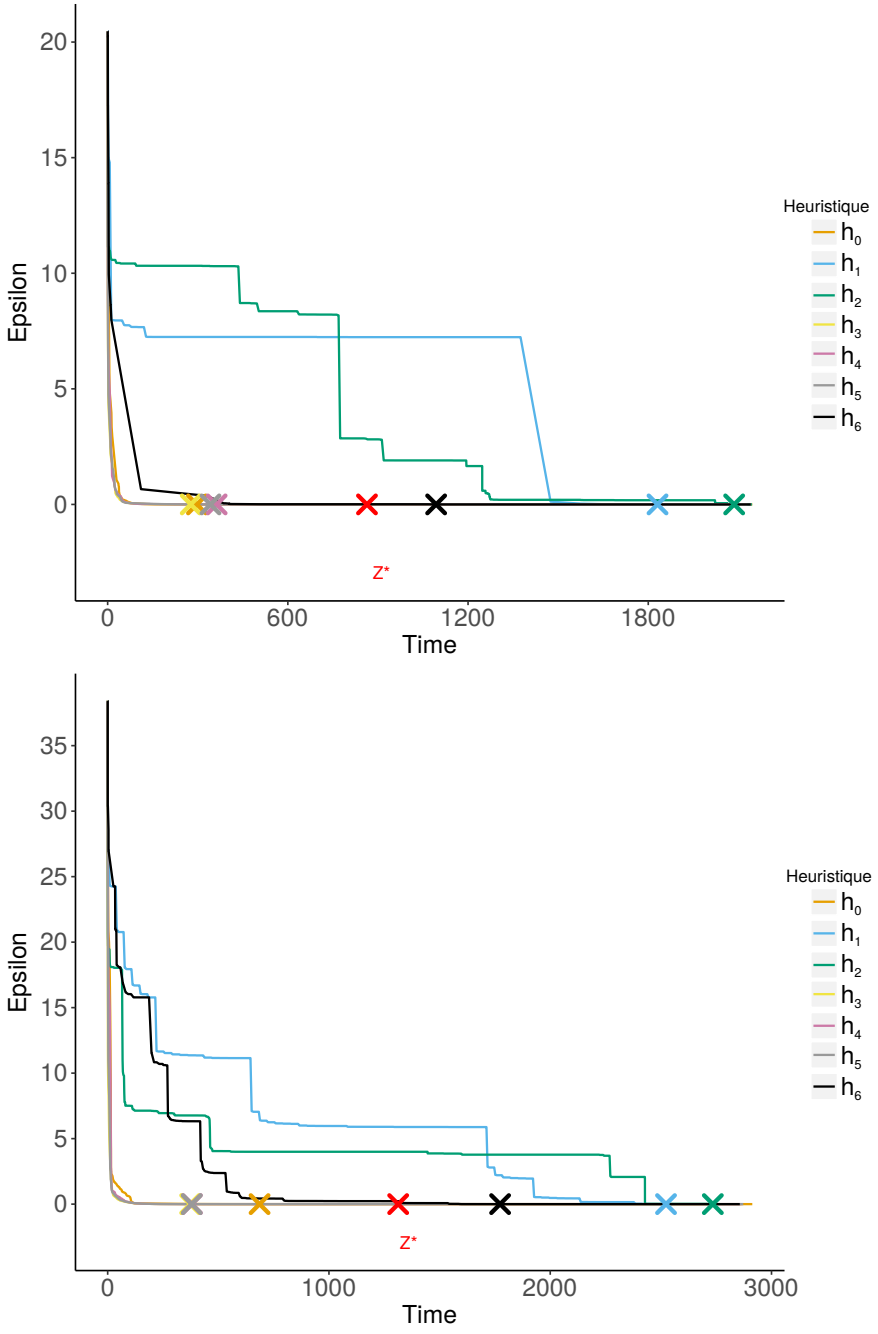


Fig. 4.8 – Graphique de l'évolution de l'incertitude ε en fonction du temps pour (haut) le mutant 271S de `1cho` et (bas) le mutant 110A de `1ktz`. Une croix représente le point d'arrivée à $\varepsilon = 10^{-3}$. La croix rouge est le point pour l'algorithme Z^*_ε.

mutations. Tous les résultats étaient sensiblement en accord, par conséquent nous avons représenté seulement deux d'entre eux dans la figure 4.8.

Les résultats montrent que, sur des instances issues de modélisation d'interface de complexe de protéines, les heuristiques h_1, h_2 et h_6 sont bien moins performantes que les autres. Ceci peut s'expliquer par le fait que les instances issues de modélisation de complexe de protéines ont une distribution de potentiels très concentrée, ce qui rend les heuristiques basées sur la récolte de plus hautes probabilités meilleures que celles basées sur la réduction d'incertitude.

Il est important de souligner que pour certaines instances (non représentées ici), l'heuristique h_1 choisit des nœuds menant à des sous-arbres avec une incertitude de plus en plus grande. Par conséquent, l'évolution de ε en fonction du temps se met à stagner (ce que l'on peut un peu apercevoir sur le long plateau bleu tracé par h_1 dans le graphique du haut de la figure 4.8).

Cette particularité de la distribution de potentiels explique aussi le fait que les heuristiques h_3, h_4 et h_5 agissent quasiment de la même façon (les courbes/croix sont superposées sur la figure 4.8). En effet, nous avons pu remarquer qu'aucun intervalle $[z_{lb}, z_{ub}]$ de la liste *open* ne s'entrecoupe ce qui a pour conséquence que le classement des nœuds suivant les trois heuristiques est le même. De plus, la figure 4.8 montre que les heuristiques h_0, h_3, h_4 et h_5 fournissent toujours une approximation à 10^{-3} avant Z^*_ε. Nous avons donc un nouvel algorithme *anytime* plus performant que son prédécesseur sur ces exemples.

Il serait intéressant de comparer ces heuristiques sur des instances issues d'autres modélisations comme par exemple ceux de certaines compétitions (PIC2011, UAI, *etc.*) et de trouver une heuristique prenant à la fois en compte l'incertitude, la quantité de probabilité et la profondeur dans l'arbre de recherche. Cela reste à faire.

4.3 `BTDZ` : exploiter la structure du graphe

Z^*_ε et `#HBFS` se montrent tous deux efficaces sur des champs de Markov avec une distribution de potentiels assez concentrée. Cependant, ils n'exploitent en aucun cas la structure du graphe (même si l'élimination de variables à la volée peut être considérée comme telle[†]). L'objectif de notre nouvel algorithme `BTDZ` est de tirer avantage de la structure du graphe afin de ne pas avoir à recalculer les parties conditionnellement indépendantes du graphe. En effet, lors d'une affectation de

†. L'élimination de variables calcule Z exactement si le graphe est un 2-arbre partiel

variables, il peut arriver que le sous graphe restant soit composé de parties indépendantes (voir section 1.5). Dans ce cas, il serait judicieux de pouvoir calculer une seule fois la fonction de partition de ces parties conditionnellement indépendantes en vue de les réutiliser si le même schéma se représente.

Premièrement, on rappelle la définition d'une décomposition arborescente déjà présentée dans la section 1.5.

Définition 4.3.1 (Décomposition arborescente [41]). *Soit* $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \mathscr{F}, \bigotimes \rangle$ *un modèle graphique. Une* **décomposition arborescente** *de* $\mathcal{M}$ *est un couple* $(\mathbf{C}, T)$ *tel que* $\mathbf{C}$ *est* ***un ensemble de clusters*** *de sommets/variables et* T *est* ***un ensemble d'arêtes*** *connectant les clusters. Un cluster* C_i *est un ensemble de sommets/variables* $C_i \subset \boldsymbol{X}$*. Une arête connectant deux clusters contient une liste de variables* X_i*, on l'appelle* ***le séparateur*** *noté* $\mathbf{sep(C_i, C_j)} = C_i \cap C_j$*.* ***L'ensemble des fils*** *d'un cluster* C_i *est noté* $ch(C_i)$ *et* ***l'ensemble des ancêtres*** *est noté* $p(C_i)$*. Pour les modèles graphiques, on introduit* $\mathscr{F}(C_i)$ *l'ensemble des fonctions de* $\mathscr{F}$ *associées au cluster* C_i*. L'ensemble* $\mathbf{C} = \{C_1, ..., C_m\}$ *vérifie les conditions suivantes :*

1. *L'ensemble des clusters doit contenir toutes les variables :*

$$\bigcup_{C_e \in \mathbf{C}} C_e = \boldsymbol{X}$$

2. *Chaque fonction est représentée dans un cluster et un seul :*

$$\forall f \in \mathscr{F},\ \exists!\ C_e \in C \mid f \in \mathscr{F}(C_e)$$

3. *Si une variable* X_i *apparaît dans deux clusters* C_s *et* C_t*, elle se doit d'appartenir aux clusters* C_e *construisant le chemin allant de* C_s *à* C_t*.*

Avec ceci, nous prendrons un exemple très simple de décomposition arborescente représentée dans la figure 4.9. Soit une décomposition arborescente (C, T) que l'on choisit enracinée au cluster C_1. On définit l'ensemble $ch(C_i)$ des clusters fils de C_i comme étant les clusters connectés à C_i et n'étant pas encore affectés. Par convention, nous choisissons que les fonctions liant les variables du séparateur appartiennent toujours au cluster père. Notons $Z(C_i|A_i)$ la fonction de partition du sous-problème enraciné en C_i sachant l'affectation du séparateur A_i (provenant du cluster père) et $\Phi(C_i) \subset \Phi$ l'ensemble des fonctions potentielles appartenant au cluster C_i alors on peut récrire l'équation 2.1 de Z par :

$$Z = Z(C_1|\varnothing) = \sum_{x \in C_1} \prod_{\phi_S \in \Phi(C_1)} \phi_S(x[S]) \left(\prod_{C_i \in ch(C_1)} Z(C_i|x[sep(C_1, C_i)]) \right) \tag{4.3}$$

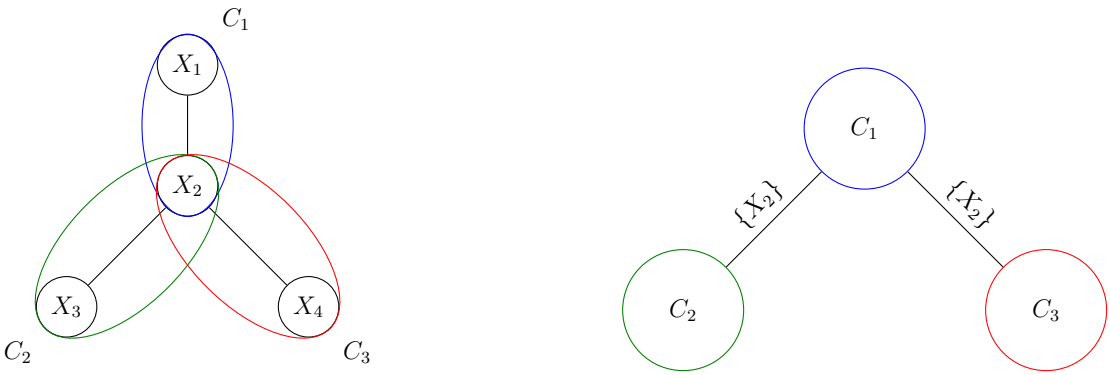


Fig. 4.9 – Décomposition arborescente d'un graphe de 4 variables X_1, X_2, X_3, X_4 en trois clusters C_1, C_2, C_3. La variable X_2 sépare les clusters C_2 et C_3 de C_1. On a donc $sep(C_1, C_2) = \{X_2\}$ et $sep(C_1, C_3) = \{X_2\}$. Si l'on affecte la variable X_2, on se retrouve avec trois sous-problèmes indépendants contenant chacun une variable.

Lorsque la fonction de partition $Z(C_i|A_i)$ est calculée alors le cluster C_i devient la racine du sous-problème P_i conditionné par l'affectation du séparateur A_i. De ce fait, la quantité $Z(C_i|A_i)$ peut être calculée indépendamment et récursivement puis mémorisée. Si l'on tombe sur un sous-problème déjà calculé, il suffira de prendre la valeur gardée en mémoire. La règle de calcul récursive est la suivante : les $Z(C_i|A_i)$ des sous-problèmes enracinés en C_i se multiplient entre eux et les $Z(C_i|A_i)$ des différentes affectations de séparateur A_i s'additionnent (voir équation 4.3 et figure 4.10).

`BTDZ` est amélioré par `SumProd VE` en *peprocessing* et à la volée ainsi que par les cohérences locales. Cependant, les cohérences locales peuvent engendrer une navigation de coûts entre clusters. En effet, pour toute assignation A_r d'un séparateur, si une cohérence locale fait passer une quantité de coût $\Delta_{rs}(A_r)$ du cluster C_r au cluster C_s alors les fonctions de partition $Z(C_r|A_r)$ et $Z(C_s|A_s)$ seront décalées par rapport aux fonctions de partition déjà mémorisées.

Il faut donc contrebalancer ces décalages par transferts de coût afin de toujours avoir en mémoire une fonction de partition dans l'**état initial du cluster** (avant toute propagation par cohérences locales). Cette gymnastique est faite dans le cadre de l'optimisation [254], il en est de même pour le comptage, il suffit de multiplier/diviser les Δ_{rs} à la fonction de partition utilisée/mémorisée.

Algorithme 4.3 : `BTDZ`. Premier appel : `BTDZ`$(\varnothing, C_1, C_1)$. La multiplication ou la division des Δ est rendue implicite, elle est faite lors de la mémorisation ou de l'utilisation d'une fonction de partition d'un cluster.

```
1  Fonction BTDZ(A, C_e, V)
2      tant que V ≠ ∅ faire
3          n = choose(V) ;                    // Choisit une variable non affectée
4          a = choose(d_i) ;                  // Choisit une valeur
5          Z = Z + BTDZ(A ∪ {i = a}, C_e, V − {i}) ;   // Appel récursif,
           affecte la valeur a
6          Z = Z + BTDZ(A ∪ {i ≠ a}, C_e, V) ;   // Appel récursif, retire
           la valeur a
7      sinon
           /* Toutes les variables du cluster sont affectées      */
8          S = Children(C_e) ;                // résout les clusters fils
9          tant que S ≠ ∅ faire
10             C_f = pop(S) ;                 // Choisit un cluster fils
11             si C_f est déjà compté alors
12                 Z = Z × Z(C_f|A) ;
13             sinon
14                 Z(C_f|A) = BTDZ(A, C_f, C_f) ;        // Appel récursif
15                 Z = Z × Z(C_f|A) ;
16     retourner Z ;
```

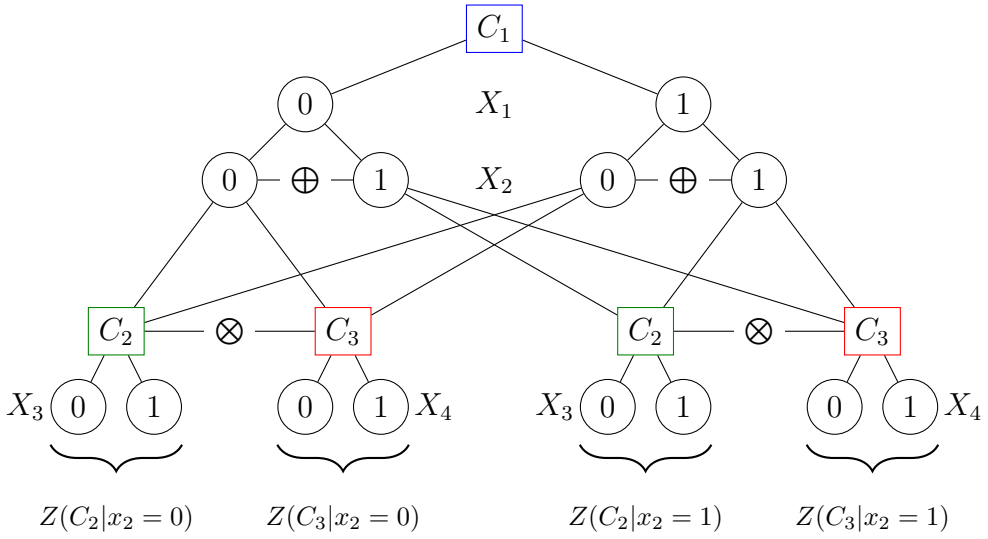
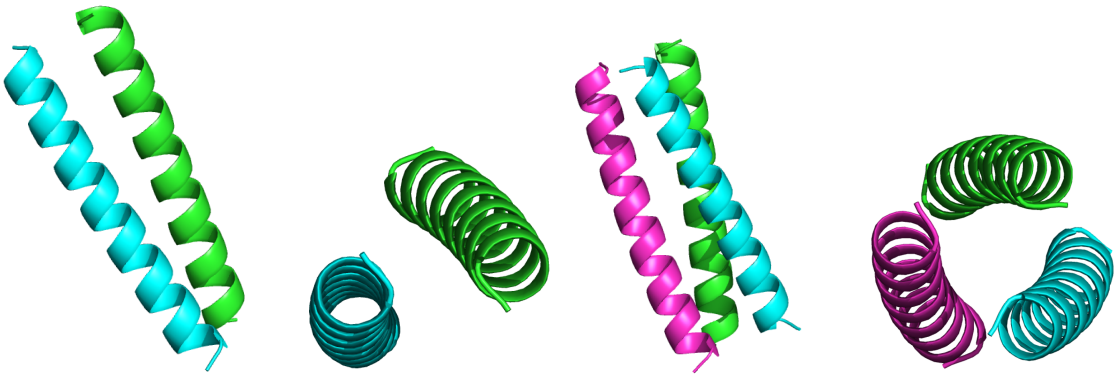


Fig. 4.10 – Règle de calcul récursive de la fonction de partition sur un modèle graphique de 4 variables. Il faut multiplier $\otimes$ au niveau des nœuds clusters (carré) et additionner $\oplus$ au niveau des nœuds variables (rond).

4.3.1 Application de `BTDZ` aux superhélices

`BTDZ` a été développé afin de pouvoir résoudre les modèles graphiques avec une forte indépendance conditionnelle. Pour tester la force de `BTDZ`, nous l'avons comparé à `#HBFS` sur des champs de Markov issus de modélisations d'assemblages moléculaires, appelés **superhélice** (ou *coiled coil*). Une superhélice est un motif structural d'interaction de protéines dans lequel des hélices α sont enroulées ensemble les unes autour des autres (figure 4.11). Les complexes les plus fréquents sont des dimères et des trimères. Notamment, on retrouve les superhélices dans les facteurs de transcriptions [255] et dans la membrane de rétrovirus, comme le VIH, qui est responsable de la fusion du virus avec la cellule hôte [256].

Notre hypothèse est que la structure des superhélices correspondra à un modèle graphique avec une forte indépendance conditionnelle et donc à une petite largeur d'arbre. Nous avons utilisé `CCBuilder` développé par Wood et al [257] afin de créer des dimères et trimères de superhélices avec différentes inclinaisons suivant les paramètres *radius* et *pitch*. Le *radius* est la distance (en Å) entre l'axe de la superhélice et le centre du complexe, basiquement ce paramètre détermine la distance entre les hélices. Le *pitch* est la distance (en Å) pour laquelle une hélice s'enroule autour de l'axe de la superhélice, c'est ce paramètre qui détermine l'enroulement des hélices entre elles. L'application est disponible en ligne à l'adresse



(a) Dimère d'hélices α en superhélice *(b)* Trimère d'hélices α en superhélice

Fig. 4.11 – Complexe de protéines en forme de superhélices.

suivante http://coiledcoils.chm.bris.ac.uk/app/cc_builder/. Nous avons joué avec le *pitch* et le *radius* sur un dimère parallèle, un trimère ainsi que sur un tétramère. Les résultats étant sensiblement les mêmes, nous avons seulement représenté en détail des instances de dimères, disponible dans le tableau 4.4. Afin de produire des champs de Markov à partir de ce jeu de superhélices, nous les avons modélisées suivant le protocole détaillé dans la section 5.1. Le tableau 4.4 montre les performances en temps de calcul de `BTDZ` face à `#HBFS` avec l'heuristique h_3 (voir section 4.2.1). On remarque que contrairement au *radius*, le *pitch* n'influe pas drastiquement sur le temps de résolution.

La diminution du *radius* augmente fortement le temps de calcul car il réduit la distance entre les monomères et par conséquent, le nombre d'interactions entre monomères augmente aussi. Cela à pour effet d'augmenter la largeur d'arbre du modèle graphique et donc a fortiori le temps de calcul.

Néanmoins, `BTDZ` permet de résoudre exactement des instances issues de modélisations de complexes protéine-protéine que `#HBFS` est loin de pouvoir résoudre, ce qui prouve son efficacité sur les modèles graphiques de petite largeur d'arbre. Cependant, lorsque la largeur d'arbre des modèles graphiques augmente `BTDZ` ne semble plus capable de calculer la fonction de partition. Cela est dû au fait que `BTDZ` calcule la fonction de partition exactement et qu'il se retrouve donc confronté au même problème que les algorithmes exacts auxquels on a comparé Z^*_εdans la section 4.1.1, c'est à dire le nombre exponentiel de termes à additionner. Une façon de contrecarrer le problème de `BTDZ` serait de le combiner avec la recherche `#HBFS` présentée dans la section précédente, de la même manière que cela a été fait pour l'optimisation [10].

# AA	*Radius*	*Pitch*	largeur d'arbre	Temps	
				`BTDZ`	`#HBFS` (ε)
56	5	90	17	M	T (59 795)
56	5	120	17	T	T (0.61)
56	5	180	17	T	T (0.58)
56	6	90	7	369	T (0.39)
56	6	120	7	200	T (0.33)
56	6	180	7	223	T (0.16)

TABLE *4.4* – Instances issues de dimères en forme de superhélice. Les instances sont décrites par : nombre d'acide aminés, nombre d'acide aminés par tour d'hélices α, *radius* (en Å), *pitch* (en Å) et le temps de résolution pour `BTDZ` et `#HBFS` en secondes (accompagné de l'incertitude ε notée entre parenthèses). (M) signifie que l'instance a dépassé la limite en mémoire de 60 Go et (T) la limite en temps de 24h.

Chapitre 5

Estimation de la constante d'affinité intermoléculaire

> "Occurrences in this domain are beyond the reach of exact prediction because of the variety of factors in operation, not because of any lack of order in nature."
>
> *Albert Einstein (1879-1955)*

Les interactions entre protéines jouent un rôle essentiel dans de nombreux processus biologiques et le remodelage de leurs interfaces est important pour diverses applications, notamment biotechnologiques et biomédicales [172–174]. Dans la section 3.3, nous avons mis en évidence la difficulté combinatoire pour estimer l'affinité de liaison intermoléculaire et de surcroît la tendance de certaines méthodes à utiliser des analyses statistiques afin de calculer l'énergie libre de façon empirique (voir section 3.3.1). Nous avons comparé nos méthodes, présentées ci-dessous, à une méthode de cette catégorie, **FoldX** [213] associant dans une fonction d'énergie des termes provenant d'analyses statistiques et des termes physiques.

Nous avons aussi détaillé quelques méthodes estimant l'énergie libre à partir du calcul de fonctions de partition de macromolécules dans l'espace des rotamères (voir section 3.3.2). Dans cette catégorie de méthodes, nous avons choisi de nous comparer à **GOBLIN**, une méthode qui déduit l'énergie libre à partir d'une estimation de la fonction de partition fournie par un algorithme sans garantie *Loopy Belief Propagation* (voir section 2.3).

Nous avons développé un outil computationnel `EasyEJayZ` permettant d'estimer l'affinité de liaison de complexes protéine-protéine et l'impact de mutations sur

les changements d'affinité, à partir d'une structure 3D d'un complexe, déterminée expérimentalement ou modélisée (coordonnées atomiques au format PDB) et d'une liste de mutants. Ce *package* python est entièrement automatique, de la préparation des systèmes moléculaires liés et non liés (avec les dernières fonctions d'énergie issues du programme de design Rosetta), jusqu'à l'estimation de l'affinité de liaison.

L'estimation de l'affinité de liaison peut se faire selon deux méthodes. Soit à partir de la méthode EasyE fournissant un score ΔE calculé à partir d'une différence d'énergie des conformations d'énergie minimale du système lié et non lié. Soit à partir de la méthode JayZ fournissant un score ΔG calculé à partir d'une différence du logarithme des fonctions de partition du système lié et non lié. L'outil est disponible en *open source* à l'adresse `https://sourcesup.renater.fr/frs/?group_id=3441`. Pour une utilisation plus facile, un *Quick Start* est disponible avec le package. Un organigramme des méthodes utilisées dans le package est disponible dans la figure 5.1.

Afin de tester l'efficacité de notre package, nous l'avons mis en œuvre sur un jeu de 21 complexes de protéines issus de la base de données PROXiMATE [258], qui englobe la base de données SKEMPI [193], regroupant des données expérimentales d'affinité de liaison sur les complexes et leurs mutants. Une vue détaillée des complexes est disponible dans le tableau 5.1. Ensuite, nous avons comparé qualitativement nos estimations aux méthodes FoldX (section 3.3.1) et GOBLIN (section 3.3.2). Enfin, comme l'algorithme Z_ε^* fournit un résultat avec garantie, nous avons pu calculer la contribution entropique (probabilités marginales) des chaînes latérales de chaque acide aminé à l'interface d'un complexe.

5.1 Matériels et méthodes

Les complexes protéines-protéines ont été sélectionnés à partir de la base de données de PROXiMATE/SKEMPI [193, 258] (tableau 5.1). Nous avons choisi de retirer du jeu de données les complexes présentant les caractéristiques suivantes :

- des données non publiées.
- si les acides aminés que l'on veut mutés sont absents dans la structure cristallographique.
- une incohérence au niveau des données (les K_a expérimentaux provenant de plusieurs expériences induisant des ordres des complexes selon K_a incohérents).
- un nombre de mutants inférieurs à 10.
- les complexes non traitables par GOBLIN (du fait d'une limite en mémoire).

Enfin, si plusieurs expériences ont été effectuées, qu'il n'y avait pas de contradiction

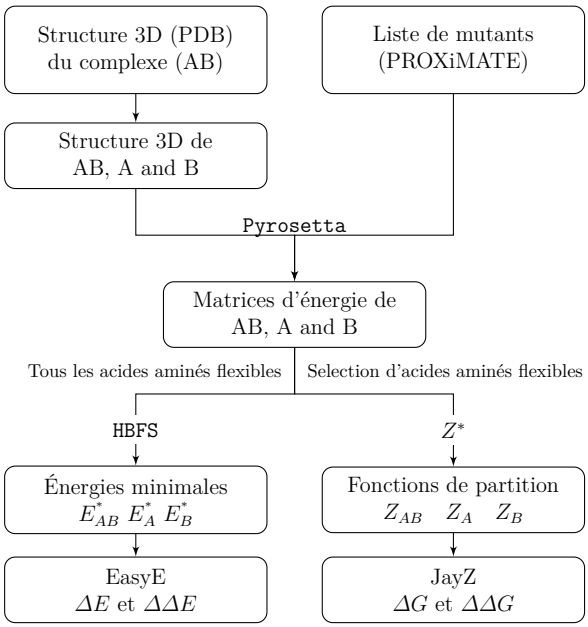


Fig. 5.1 – Organigramme de EasyE et JayZ pour l'estimation de l'énergie de liaison.

mais qu'il y avait un nombre de mutants différents, nous avons sélectionné les données comportant le plus de mutants.

Finalement, notre jeu de données regroupe un total de 1 094 mutants (de une à huit mutations) répartis sur 21 complexes comprenant des protéines connues telles que *Bovine α-chymotrypsin* (`1cho`), *Human leukocyte elastase* (`1ppf`),*Subtilisin Carlsberg* (`1r0r`) et *Streptomyces griseus proteinase B* (`3sgb`) ; toutes les quatre en interaction avec la protéine *Turkey ovomucoid 3rd domain*. Ces quatre complexes représentent à eux seuls 60% des données expérimentales. De plus, notre jeu de données comporte 9 complexes sur les 21 ayant une résolution supérieure à 2.5 Å.

Les structures des 21 complexes proviennent de la banque de données de structures de protéines (la PDB) [97] et ont été résolues par cristallographie aux rayons X. Pour les structures 3D des partenaires seuls, nous avons pris les structures des complexes et nous les avons simplement séparées. En effet, les structures des partenaires sous forme liés sont considérées comme une bonne approximation pour les états non-liés. Dans notre outil, la séparation est faite automatiquement . Les hydrogènes ont été ajoutés en utilisant la version 144 de PyRosetta [259].

Les matrices d'énergies, du complexe et des partenaires, pour la protéine sauvage et ses mutants, ont été calculées avec les dernières fonctions d'énergies décomposables de Rosetta *beta_nov15* et *beta_nov16* (voir section 3.2.2) et la librairie de rotamères de Dunbrack [119, 154, 260].

Nous avons testé les fonctions d'énergie `beta_nov15` et `beta_nov16` dans leurs versions *hard* et *soft* [154, 260]. La version *soft* a une énergie répulsive de Lennard-Jones atténuée ce qui autorise de petits recouvrements entre atomes (voir section 3.2.2). Nous avons aussi testé les différentes extensions de la librairie de rotamères de Dunbrack, nommées `EX1` et `EX2`. `EX1` ajoute des conformations supplémentaires à ± 1 écart-type pour l'angle χ_1. `EX2` fait de même, pour les angles χ_1 et χ_2. Dans la suite, nous désignerons par `EX0` la librairie par défault.

Comme le calcul de la fonction de partition est exponentiel en temps, nous avons dû choisir un sous-ensemble d'acides aminés à l'interface du complexe afin de pouvoir utiliser notre algorithme Z^*_ε. L'interface contient tous les acides aminés mutables et ceux ayant une chaîne latérale à moins de 3 Å d'un acide aminé mutable. Les acides aminés ne faisant pas partie de l'interface ont été fixés à leur conformation d'énergie minimale. Quand elle était connue, la température à été fixée selon les données expérimentales et sinon, à 298 K .

Afin d'étudier la contribution de l'entropie dans l'estimation de l'affinité de liaison protéine-protéine, nous avons calculé deux scores différents. Pour la méthode EasyE, nous avons calculé le score ΔE (voir la section 3.4) que nous rappelons dans l'équation 5.1 et pour la méthode JayZ, nous avons calculé le score ΔG (voir la

TABLE *5.1* – Complexes de protéine issus de PROXiMATE/SKEMPI. Pour chaque complexe A-B : PDB ID, nom de la protéine A, nom de la protéine B, nombre d'acides aminés de AB, A et B, résolution (en Å) de la structure cristallographique (par rayon X) du complexe, nombre de mutants, alanine scan (croix pour oui), variabilité de la différence en énergie libre en kcal.mol^{-1} (min, max and variance σ).

PDB	Partenaire A	Partenaire B	# AA			res	Mut	A	Δ G		
			AB	A	B			S	min	max	σ
1ktz	Transforming growth factor beta 3	TGF-beta type II receptor	188	82	106	2.2	13		−8.85	−4.37	1.11
3sgb	Streptomyces griseus proteinase B	Turkey ovomucoid 3rd domain	235	185	50	1.8	163		−16.99	−8.55	1.70
1ppf	Human leukocyte elastase	Turkey ovomucoid 3rd domain	274	218	56	1.8	163		−16.18	−5.65	1.92
1cho	Bovine α-chymotrypsin	Turkey ovomucoid 3rd domain	291	238	53	1.8	163		−17.35	−8.06	1.92
1cbw	Bovine α-chymotrypsin	BPTI	297	239	58	2.6	18		−13.32	−5.50	2.35
1ak4	Cyclophilin A	HIV-1 capsid protein	310	165	145	2.4	13		−8.06	−3.68	1.04
1iar	Interleukin-4	Interleukin-4 receptor	317	129	188	2.3	36		−13.91	−9.59	0.98
1r0r	Subtilisin Carlsberg	Turkey ovomucoid 3rd domain	325	274	51	1.1	163		−16.36	−7.85	1.72
4oga	Human insulin receptor, soluble	Human insulin	330	42	288	3.5	43	×	−9.31	−6.45	0.54
1dan	Factor VIIa	Tissue factor	386	306	80	2.0	71		−11.69	−8.6	0.44
1vfb	IgG1-kappa D1.3 Fv	HEW Lysozyme	352	223	129	1.8	27	×	−10.76	−6.92	1.16
3n4i	Klebsiella pneumoniae beta-lactamase SHV-1	Streptomyces clavuligerus (BLIP)	430	265	165	1.6	26	×	−7.74	−4.64	0.88
1dvf	IgG1-kappa D1.3 Fv	E5.2 Fv	451	223	228	1.9	17	×	−10.85	−6.57	1.25
1c4z	Human ubiquitin-conjugating enzyme E2 L3 (UbcH7)	Human ubiquitin-protein ligase E3A (E6AP)	494	350	144	2.6	47		−6.04	−3.09	0.57
2bdn	Mouse antibody 11K2 scFv	Human C-C motif chemokine 2/(MCP-1)	499	68	431	2.5	13		−9.77	−6.54	0.87
3bpl	Human IL-4 with IL-4R	Cytokine receptor common subunit gamma	522	328	194	2.9	17	×	−5.51	−4.46	0.33
1jrh	mAbs A6	Interferon gamma receptor	540	432	108	2.8	22		−11.34	−5.77	1.88
1b41	Human acetylcholinesterase (AChE)	Dendroaspis angusticeps fasciculin-2 (fas-2)	592	531	61	2.8	23		−10.68	−7.38	0.78
1fss	Tetronarce californica acetylcholinesterase (AChE)	Dendroaspis angusticeps fasciculin-2 (fas-2)	593	532	61	3.0	23		−10.52	7.48	0.77
1ao7	Human T-cell receptor A6	Human HLA class I histocompatibility antigen TAX-HLAA2	607	283	324	2.6	24		−8.64	−5.68	0.95
1ahw	Immunoglobulin FAB 5G9	Tissue factor	628	428	200	3.0	12	×	−13.42	−10.19	1.00

section 3.3) que nous rappelons dans l'équation 5.2.

$$\Delta E = E_{AB}(\mathbf{x}^*) - (E_A(\mathbf{x}^*) + E_B(\mathbf{x}^*)) \tag{5.1}$$

$$\Delta G = -k_B T \left(\log\left(Z_{AB}\right) - \left(\log\left(Z_A\right) + \log\left(Z_B\right)\right)\right) \tag{5.2}$$

Le score ΔE est calculé à partir des énergies des conformations d'énergies minimales (GMEC) des partenaires et du complexe. Pour cela, nous avons utilisé les méthodes d'optimisation provenant des réseaux de fonctions de coût couplées à l'algorithme de recherche `HBFS` [10]. Le score ΔG est calculé à partir des fonctions de partition des partenaires et du complexe. Pour cela, nous avons utilisé notre algorithme Z^*_ε (section 4.1) afin d'obtenir une approximation à au moins 10^{-3} près.

Nous avons défini la différence en énergie libre (calculée ou expérimentale) pour chaque mutant S par rapport à une mesure de référence ref, fixée à l'estimation de l'affinité de la protéine sauvage WT, comme suit :

$$\Delta\Delta E = \Delta E_{WT} - \Delta E_S \qquad \Delta\Delta G = \Delta G_{WT} - \Delta G_S$$

Nous avons comparé nos deux méthodes avec FoldX version 4 et GOBLIN suivant les protocoles définis dans les sections 3.3.1 et 3.3.2.

Afin d'évaluer la qualité des estimations de l'affinité de liaison des différentes méthodes, nous avons considérés trois statistiques : **le coefficient de corrélation de Pearson** R qui évalue la qualité d'une régression linéaire, **l'erreur des moindres carrées** (*Root Mean Square Error* : *RMSE*) qui évalue la distance des estimations par rapport aux valeurs expérimentales et **l'aire au dessous de la courbe** ROC sensibilité/spécificité (*Area Under the Curve* : *AUC* , *Receiver operating characteristic* : ROC) qui évalue la sensibilité d'un classificateur.

Chaque estimation ΔG_S peut être classée comme positive si on prédit $\Delta G_{WT} \geq \Delta G_S$ et négative sinon. Puis comme vraie si l'estimation est en accord avec les données expérimentales et fausse sinon. Cela permet de classer chaque séquence dans une des quatre catégories suivantes : Vrai Positif, Vrai Négatif, Faux Positif, Faux Négatif (VP, VN, FP, FN). La sensitivité (TPR) est le taux de vrai positif et la spécificité (SPC) est le taux de vrai négatif :

$$TPR = \frac{VP}{VP + FN} \qquad SPC = \frac{VN}{VN + FP}$$

Nous avons calculé une p-valeur où H_0 suppose une prédiction aléatoire optimale. On peut considérer un estimateur désignant aléatoirement un mutant comme positif ou négatif suivant une loi de Bernouilli de paramètre μ (un peu comme un pile ou face). Pour un système donné, si σ est la proportion de mutants considérés

comme vrais alors l'estimateur précédent est optimal si $\mu = \sigma$. Par conséquent, $\mu = \sigma = \frac{\#V_{exp}}{n}$, avec n le nombre de mutants, devient le choix optimal pour l'estimateur aléatoire.

La p-valeur associée à H_0 est la probabilité que l'estimateur aléatoire possède une précision supérieure ou égale à l'estimateur que l'on souhaite évaluer. Si, pour un système donné, on a n mutants, alors cela est équivalent à n répétitions de la loi de Bernoulli et donc une loi binomiale de paramètre n, σ amenant à avoir un compte $\#V$ de vrais supérieurs aux nombres de vrais de l'estimateur. On a donc la p-valeur suivante :

$$P(\#V \geq VP + VN) = \sum_{i=VP+VN}^{n} P(\#V = i) = \sum_{i=VP+VN}^{n} \binom{n}{i} \mu^i (1-\mu)^{n-i}$$

Finalement, les méthodes expérimentales introduisent généralement un bruit résultant en une imprécision de $\pm 20\%$ [261]. Par conséquent, dans le but de simuler ce bruit, chaque mesure expérimentale a été représentée par une valeur tirée aléatoirement selon une loi normale $\mathcal{N}(K_a, 0.20K_a)$ restreinte à l'intervalle $[0.8K_a, 1.20K_a]$. Chaque analyse statistique a été reproduite 1000 fois puis nous avons pris la moyenne empirique sur les 1000 occurrences comme résultat final.

5.2 Résultats et discussions

Les fonctions d'énergie sont décomposées en une combinaison linéaire de termes énergétiques (voir la section 3.2.2 et l'équation 3.1). Dans la plupart des évaluations de méthodes de prédiction d'affinité de liaison, la base de données initiale de mutants est divisée en plusieurs parties afin d'ajuster les poids sur une partie et confronter la méthode sur les parties restantes [219, 220]. Même si une validation croisée sur des échantillons aléatoires permet d'éviter un sur-ajustement des poids, l'application sur un jeu de données extérieures reste difficile [219]. En outre, il est difficile de construire des sous-ensembles de tests et d'ajustement totalement indépendants ; par exemple certains échantillons partageront le même squelette ou le même repliement. De plus, lorsque l'on estime une librairie de mutants pour une interaction protéine-protéine, il n'y a généralement pas de données expérimentales disponibles *a priori*. Par conséquent, afin de nous placer dans le cas réel, nous avons testé les performances des méthodes présentées ici, sans aucun ajustement de poids des fonctions d'énergies.

TABLE *5.2* – temps CPU (en secondes) pour EasyE avec `beta_nov15_`*soft*. Pour chaque complexe : PDB ID, taille de l'espace séquence-conformation, degré médian moyen de chaque variable dans la modèle graphique associé, temps CPU pour le calcul de la matrice d'énergie, temps CPU minimum, maximum et moyen pour un calcul de ΔE et temps CPU total pour tous les mutants, temps CPU total.

PDB	Taille	Med Deg	CPU Mat	temps CPU par ΔE Min	Max	Moyenne	All	Total
1ktz	10^{275}	17	27	0.46	0.48	0.47	6	33
3sgb	10^{215}	19	176	4	4	4	651	827
1ppf	10^{225}	20	154	3.79	4.04	3.85	627	781
1cho	10^{264}	20	170	4.16	4.43	4.21	687	857
1cbw	10^{253}	19	26	0.70	0.71	0.70	13	39
1ak4	10^{270}	20	29	0.79	0.83	0.80	10	39
1iar	10^{311}	21	45	1.22	1.38	1.30	48	93
1r0r	10^{271}	20	174	4.23	4.75	4.50	734	908
4oga	10^{298}	20	37	1.10	1.17	1.13	49	86
1dan	10^{551}	22	62	2.18	2.44	2.26	161	223
1vfb	10^{335}	22	54	1.80	1.93	1.84	50	104
3n4i	10^{372}	24	51	1.52	1.78	1.59	41	92
1dvf	10^{444}	22	53	2.12	2.46	2.21	38	91
1c4z	10^{472}	22	68	2.18	2.48	2.32	109	177
2bdn	10^{500}	21	76	4.38	5.14	4.78	62	138
3bpl	10^{528}	22	53	3.54	4.07	3.73	63	116
1jrh	10^{476}	19	41	1.59	1.68	1.63	36	77
1b41	10^{493}	23	50	1.78	1.92	1.84	42	92
1fss	10^{549}	25	62	2.25	2.60	2.35	54	116
1ao7	10^{576}	22	80	3.36	4.71	3.80	91	171
1ahw	10^{630}	22	129	3.70	4.62	4.25	51	180
Total							3623	5240

Dans un premier temps, nous avons comparé les scores fournis par la méthode EasyE en utilisant les fonctions d'énergie, `beta_nov15` ou `beta_nov16` dans leurs versions *hard* ou *soft* et ce, avec trois niveaux différents de discrétisation de rotamères,`EX0`, `EX1` ou `EX2`. Il est intéressant de noter que le problème d'optimisation sur le plus gros complexe `1ahw`, constitué de 628 acides aminés, avec un modèle graphique issu de `beta_nov16`_*soft* et `EX0`, a été résolu en seulement 8.18 secondes de temps CPU sur un processeur Xeon E5-2687W@3GHz et qu'aucun système ne prit plus de 21 secondes (Tableau 5.2 et 5.3). Même si les extensions de librairie de rotamère de Dunbrack rendent les problèmes beaucoup plus gros, allant jusqu'à 10^{927} conformations pour `1ahw` avec `EX2`, ceux là restent encore faisables. Comme il a déjà été montré par Kellogg et al. [262] dans le cadre de l'énergie libre de repliement, nous avons observé que les résultats obtenus avec la version *soft* sont toujours meilleurs que ceux obtenus avec la version *hard* (figure 5.2).

En ce qui concerne le *RMSE*, on peut remarquer que l'utilisation des versions *soft* de `beta_nov15` et `beta_nov16` fournit des résultats très proches, de l'ordre de 2.20 kcal.mol^{-1} avec n'importe quelle extension de librairie de rotamères tandis que les version *hard* ne descendent pas au-dessous des 30 kcal.mol^{-1} même en utilisant l'extension `EX2`. Le *RMSE* obtenu avec les versions *hard* est très élevé et cela pourrait provenir du fait que les conformations d'énergie minimale des mutants sont très hautes en énergie. Les versions *soft* ont une énergie répulsive de Lennard-Jones atténuée ce qui autorise de petit "clash" entre atomes et compense le fait que l'on a fixé le squelette. On peut donc en déduire que les énergies élevées des conformations optimales des mutants sont sûrement dues à des chaînes latérales trop proches. De plus, les versions *soft* ont aussi une corrélation bien meilleure que les versions *hard*, de l'ordre de 0.40 contre 0.20.

Dans un second temps, nous avons comparé l'estimation de EasyE en utilisant `beta_nov15` ou `beta_nov16` version *soft* et la librairie de rotamères par défaut `EX0` avec les estimations fournies par FoldX et GOBLIN (tableau 5.4). Il est aussi utile de souligner que les résultats précédemment publiés pour FoldX et GOBLIN dans [213, 220, 230, 263–265] concernent des énergies libres de repliement et/ou incluent en général une exclusion de valeurs aberrantes que nous n'avons pas souhaité réaliser. En effet, ces valeurs ne peuvent pas être détectées *a priori* dans un cas habituel d'utilisation. Nos résultats ne sont donc pas comparables à ceux présentés dans ces publications.

Le tableau 5.4 rassemble les moyennes empiriques de l'AUC (%), de la corrélation et du *RMSE* (en kcal.mol^{-1}) prisent sur les 21 complexes de protéines. Un détail

TABLE *5.3* – temps CPU (en secondes) pour EasyE avec `beta_nov16_`*soft*. Pour chaque complexe : PDB ID, taille de l'espace séquence-conformation, degré médian moyen de chaque variable dans la modèle graphique associé, temps CPU pour le calcul de la matrice d'énergie, temps CPU minimum, maximum et moyen pour un calcul de ΔE et temps CPU total pour tous les mutants, temps CPU total.

PDB	Taille	Med Deg	CPU Mat	temps CPU par ΔE Min	Max	Moyenne	All	Total
1ktz	10^{197}	19	47	0.83	0.91	0.87	11	58
3sgb	10^{235}	20	261	6.69	7.29	6.85	1117	1378
1ppf	10^{253}	22	280	6.28	7.10	6.54	1065	1345
1cho	10^{293}	22	298	6.76	7.56	6.97	1137	1435
1cbw	10^{278}	21	39	1.09	1.18	1.11	20	59
1ak4	10^{303}	21	45	1.45	1.56	1.48	19	64
1iar	10^{348}	22	78	2.30	2.79	2.52	93	171
1r0r	10^{301}	22	316	6.75	7.82	7.06	1151	1467
4oga	10^{330}	21	54	1.81	1.98	1.89	81	135
1dan	10^{615}	23	86	4.21	7.46	5.28	375	461
1vfb	10^{370}	23	88	3.09	3.22	3.16	85	173
3n4i	10^{414}	25	75	2.68	2.96	2.79	73	148
1dvf	10^{492}	22	86	3.93	4.70	4.33	74	160
1c4z	10^{521}	23	108	4.15	6.10	4.84	227	335
2bdn	10^{547}	21	112	7.32	10.14	8.52	111	223
3bpl	10^{588}	23	76	7.16	9.55	8.30	141	217
1jrh	10^{433}	20	66	2.66	3.26	2.91	64	130
1b41	10^{548}	25	51	4.33	7.11	5.09	117	168
1fss	10^{609}	26	109	4.57	6.27	5.17	119	221
1ao7	10^{636}	22	137	9.59	20.96	13.43	322	459
1ahw	10^{693}	23	176	5.79	8.18	6.73	81	257
Total							6483	9064

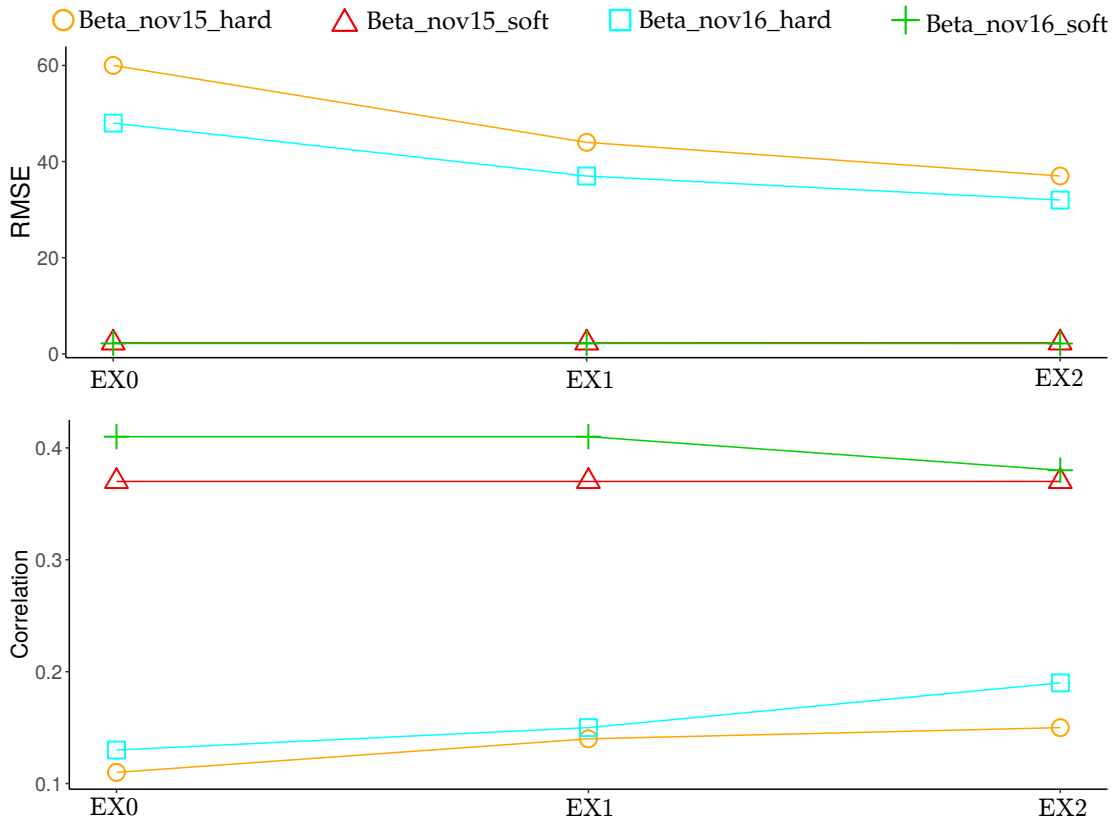


Fig. 5.2 – Comparaison de l'estimation fournie par la méthode EasyE avec `beta_nov15`_*hard* (cercle orange), `beta_nov15`_*soft* (triangle rouge), `beta_nov16`_*hard* (carré bleu) et `beta_nov16`_*soft* (croix verte) et trois niveaux de discrétisation de rotamères (`EX0`, `EX1` et `EX2`). Moyenne du *RMSE* (en kcal.mol^{-1}) sur les 21 complexes protéine-protéine (haut). Moyenne de la corrélation R (bas).

		Correlation	AUC	RMSE
EasyE	`beta_nov15_`*soft*	0.37	69	2.28
EasyE	`beta_nov16_`*soft*	0.41	70	2.18
FoldX		0.24	58	1.94
GOBLIN		0.16	65	2.20

TABLE *5.4* – Moyennes de la corrélation, AUC (%) et *RMSE* (kcal.mol^{-1}) pour les méthodes EasyE, FoldX et GOBLIN sur les 21 complexes de protéines.

des résultats par système est disponible dans le tableau 5.5. Comme le montre les valeurs des corrélations et des AUCs, la méthode EasyE fournit de meilleurs résultats qualitatifs que FoldX et GOBLIN, que cela soit avec `beta_nov15_`*soft* ou `beta_nov16_`*soft*. En effet, la corrélation moyenne de la méthode EasyE avec `beta_nov16_`*soft* est de 0.41 alors qu'elle est seulement de 0.24 pour FoldX et chute à 0.16 pour GOBLIN. De plus, EasyE fournit une meilleure corrélation que GOBLIN pour 18 complexes, et que FoldX pour 16 complexes sur les 21 (tableau 5.5). La méthode EasyE dépasse une corrélation de 0.50 sur 10 des 21 complexes alors que FoldX et GOBLIN atteignent le même niveau de performance sur 5 et 3 complexes respectivement. Pour les autres complexes dont la corrélation est inférieure à 0.50, deux d'entre eux ont une corrélation tout de même proche de ce seuil et deux sont négatifs (tableau 5.5) contre quatre corrélations négatives pour FoldX et GOBLIN.

L'AUC moyenne de la méthode EasyE est de 70% vs. 58% pour FoldX et 65% pour GOBLIN. De plus, sur 21 complexes, EasyE fournit une meilleure précision que FoldX dans 14 cas et que GOBLIN dans 12 (tableau 5.5). L'AUC est supérieure à 50% sur 19 des 21 complexes comparée à 13 et 17 complexes pour FoldX et GOBLIN. L'AUC la plus basse pour EasyE est de 31%, pour le complexe `1ahw`, contre 31% et 42% pour FoldX et GOBLIN. Ce complexe est celui comportant le plus d'acides aminés avec une résolution assez basse de 3 Å ce qui appuie l'hypothèse corrélant la résolution à la prédiction de l'affinité énoncé dans Marillet et al [219].

Afin que la p-valeur soit significative, nous avons sélectionné les treize complexes avec plus de 20 mutants. Douze de ces complexes ont des p-valeurs associées inférieures à 1% pour EasyE contre onze pour FoldX et neuf pour GOBLIN. Enfin, les trois méthodes ont des *RMSE* respectables et assez similaires, de 1.94 vs 2.20 vs 2.18 kcal.mol^{-1}, pour FoldX, GOBLIN et EasyE.

TABLE *5.5* – Root Mean Square Error (RMSE en kcal.mol^{-1}), coefficient de corrélation de Pearson (R) et AUC (%) sur les estimations fournies par EasyE, GOBLIN et FoldX sur 1094 mutants provenant de 21 complexes protéine-protéine : PDB ID, nombre de mutants, RMSE moyen, R moyen, AUC moyenne.

		EasyE						GOBLIN			FoldX		
PDB	Mut	Beta_nov15_*soft*			Beta_nov16_*soft*								
		RMSE	R	AUC	RMSE	R	AUC	RMSE	R	AUC	RMSE	R	AUC
1ktz	12	1.63	0.44	75	1.57	0.48	75	2.18	0.14	58	1.41	0.78	92
3sgb	162	3.03	0.40	57	2.46	0.54	67	2.32	0.23	46	2.00	0.46	60
1ppf	162	3.38	0.66	71	3.60	0.65	70	2.36	0.32	60	4.79	0.01	63
1cho	162	3.18	0.30	59	2.89	0.38	62	3.06	0.28	64	2.54	0.19	54
1cbw	17	5.04	0.71	77	4.48	0.81	83	1.91	0.72	83	1.75	0.86	97
1ak4	12	1.71	0.61	92	1.38	0.73	92	2.40	−0.06	75	2.73	0.04	42
1iar	35	1.84	0.44	62	1.96	0.55	67	1.62	0.17	65	1.17	0.51	68
1r0r	162	2.80	0.56	71	2.78	0.50	63	2.50	0.15	46	2.04	0.35	58
4oga	42	0.93	0.20	51	0.86	0.26	54	2.19	0.23	64	0.70	0.48	56
1dan	70	1.34	0.53	62	1.44	0.50	59	1.66	0.22	58	0.77	0.43	55
1vfb	25	2.30	0.65	96	2.88	0.65	96	2.04	0.55	92	1.80	0.54	65
3n4i	25	3.99	−0.23	71	4.11	−0.16	75	3.68	0.17	78	2.46	−0.24	50
1dvf	16	1.89	0.54	69	1.94	0.47	69	1.81	0.13	88	1.74	0.55	75
1c4z	46	1.39	0.44	66	1.32	0.53	75	1.73	0.33	59	1.61	0.22	45
2bdn	12	1.42	0.28	90	1.18	0.30	89	1.62	−0.17	60	1.46	−0.31	21
3bpl	16	1.06	0.23	32	1.16	0.17	47	1.84	0.51	75	0.60	0.22	38
1jrh	20	2.88	0.26	66	2.25	0.53	74	2.81	−0.18	31	2.02	0.36	58
1b41	22	2.05	0.44	59	2.29	0.26	53	1.98	0.26	58	2.38	0.02	43
1fss	22	1.33	0.49	78	1.47	0.42	68	1.68	0.23	60	2.76	0.09	47
1ao7	22	3.57	0.09	100	2.74	0.20	100	2.61	0.12	100	2.70	−0.03	100
1ahw	11	1.11	−0.18	47	1.04	−0.08	31	2.28	−0.64	42	1.40	−0.45	31
Moyenne	1073	2.28	0.37	69	2.18	0.41	70	2.20	0.18	65	1.94	0.24	58

TABLE *5.6* – temps-CPU (en secondes) de JayZ avec beta_nov15_*soft* et beta_nov16_*soft*. Pour chaque complexe : PDB ID, nombre d'acides aminés flexibles (nombre de variables dans le modèle graphique associé), degré médian moyen dans le modèle graphique associé, Taille moyenne de l'espace séquence-conformation, temps CPU minimum, maximum, moyen et total pour un calcul de ΔG et temps CPU total sur tous les mutants.

PDB	Flex	Beta_nov15_*soft*						Beta_nov16_*soft*					
		Taille	Med Deg	CPU par ΔG Min	Max	Moyenne	*CPU* Total	Taille	Med Deg	CPU par ΔG Min	Max	Moyenne	*CPU* Total
1ktz	14	10^{19}	8	23	1619	359	4672	10^{20}	8	2501	101252	31878	414415
3sgb	13	10^{25}	8	6	2938	290	47350	10^{26}	8	36	233698	16617	2708615
1ppf	14	10^{26}	8	13	163128	7536	1228408	10^{27}	8	0	195094	24514	3995723
1cho	14	10^{26}	8	11	147187	5645	869263	10^{27}	9	0	157848	10086	1613828
1cbw	2	10^{4}	0	1	27	9	163	10^{4}	0	2	49	17	307
1ak4	13	10^{11}	4	0	1	1	9	10^{12}	4	1	1	1	16
1iar	31	10^{46}	12				T	10^{50}	13				T
1r0r	15	10^{24}	9	24	169920	19181	3126629	10^{29}	9	0	173293	33161	5405338
4oga	53	10^{63}	10				T	10^{69}	11				T
1dan	80	10^{103}	8				T	10^{114}	9				T
1vbf	26	10^{31}	10				T	10^{34}	11				T
3n4i	32	10^{43}	10				T	10^{45}	10				T
1dvf	25	10^{31}	11				T	10^{35}	11				T
1c4z	51	10^{59}	16				T	10^{65}	16				T
2bdn	9	10^{15}	3	0	3	0	3	10^{15}	3	0	4	0	4
3bpl	21	10^{29}	7				T	10^{32}	7				T
1jrh	20	10^{25}	10				T	10^{28}	10				T
1b41	17	10^{28}	9	15	107439	27049	622121	10^{29}	9				T
1fss	16	10^{26}	7	82	132235	21650	497972	10^{28}	7				T
1ao7	8	10^{12}	2	0	3	2	40	10^{13}	2	0	8	4	91
1ahw	20	10^{25}	5	7	1285	512	6143	10^{28}	5	61	20561	7030	84364

Comme pour la méthode EasyE, nous avons évalué les performances de la méthode JayZ pour estimer l'affinité de liaison en utilisant l'algorithme Z^*_ε avec une garantie de $\varepsilon = 10^{-3}$ et les fonctions d'énergie `beta_nov15` et `beta_nov16` dans leurs versions *hard* et *soft* ainsi que la librairie de rotamères par défaut `EX0`. Nous avons fixé la limite de temps de calcul d'une fonction de partition à 48 heures (temps CPU). Dans cette limite de temps, la version `beta_nov15`_*soft* des fonctions d'énergie n'a pas pu retourner une valeur de Z pour 9 complexes sur 21 et la version `beta_nov16`_*soft* n'a pas pu retourner de valeur de Z pour 11 des 21 complexes (tableau 5.6). La taille de l'espace de comptage peut atteindre jusqu'à 10^{114} avec des complexes atteignant un nombre d'acides aminés flexibles pouvant aller jusqu'à 80. En ce qui concerne les versions *hard*, les modèles graphiques associés aux complexes étaient bien plus facile à résoudre.

Nous avons comparé le score de EasyE et de JayZ sur les quatre fonctions d'énergie et nous avons été surpris par le comportement quasi-similaire des deux scores. Théoriquement, un score basé sur les fonctions de partition (ΔG de JayZ) devrait être plus précis qu'un score calculé seulement à partir des énergies des conforma-

tions minimales (ΔE de EasyE). Cependant, la figure 5.3 montre par exemple, sur le complexe `3sgb`, que $\Delta\Delta G$ semble être une version légèrement perturbée de $\Delta\Delta E$. En effet, les deux scores ont une corrélation supérieure à 99%. Cela en va de même pour les autres complexes, excepté pour `1ahw` avec les versions *soft* des fonnctions d'énergie.

Effectivement, pour `1ahw`, l'estimation fournie par JayZ est nettement meilleure que celle fournie par EasyE, avec une corrélation de 0.65 contre -0.18 pour `beta_nov15_`*soft* et de 0.32 contre -0.08 pour `beta_nov16_`*soft*.

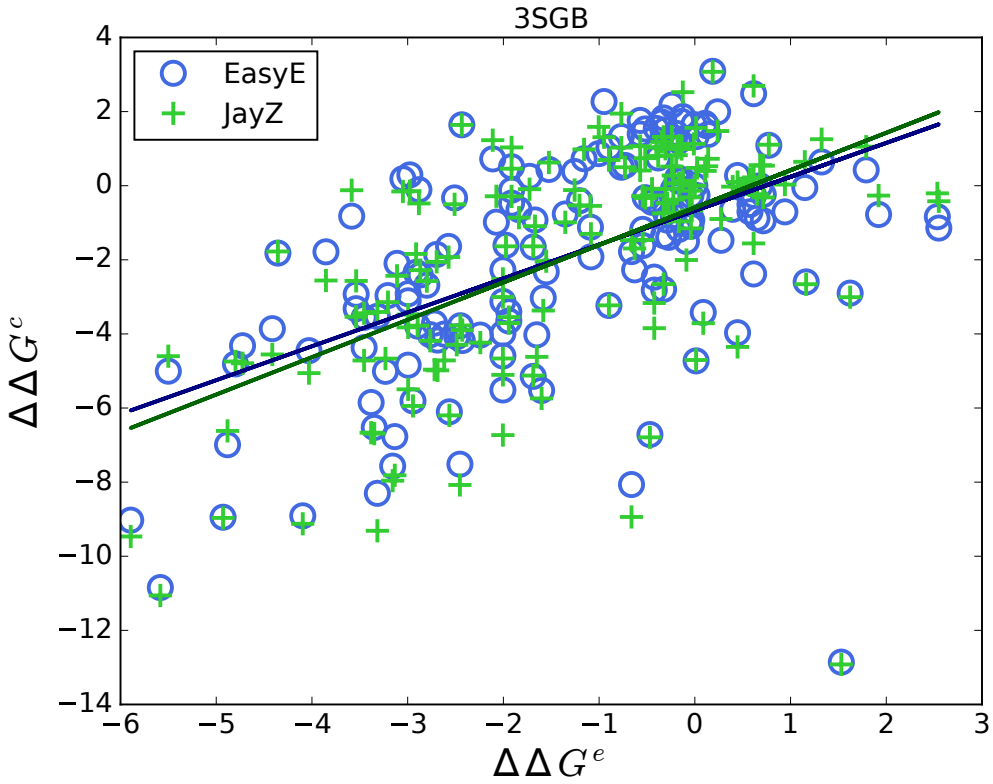


Fig. 5.3 – Scatter plot du $\Delta\Delta G^e$ versus $\Delta\Delta E$ de EasyE (cercle bleu) et $\Delta\Delta G$ de JayZ (croix verte) accompagnés de leur régression linéaire pour le complexe `3sgb`. Les matrices d'énergies ont été calculées à partir de `beta_nov16_`*soft*

Afin de regarder plus en détail pourquoi `1ahw` se comportait de cette façon, nous avons estimé les distributions des probabilités marginales des rotamères pour chaque acide aminé de l'interface. Nous avons fait cela en combinant l'aptitude

des algorithmes de CFN à énumérer exhaustivement les conformations dans un seuil d'énergie donné et dans un temps raisonnable [4, 5, 7, 10] avec la valeur de la fonction de partition retournée par Z^*_ε.

En utilisant `HBFS`, nous avons pu énumérer entre 80 et 90% de la fonction de partition pour le mutant Y574A de `1ahw`, l'estimation la plus aberrante avec EasyE et corrigée par JayZ. Alors que Z^*_ε, n'a pris que **5 minutes** pour calculer Z à 10^{-3} près, l'énumération des conformations et de leur énergie a pris **5 jours**. Cela démontre bien l'importance de l'élimination de variable à la volée dans l'algorithme Z^*_ε.

À titre d'anecdote, nous avions décidé, un peu naïvement, d'écrire les conformations et leur énergie dans un fichier texte afin de calculer les marginales. Nous nous sommes vite aperçus que cela était impossible car pour seulement 40% de masse de probabilités, cela prenait a peu près 300 Go d'espace de stockage. Pour rendre ce fait un peu plus concret, en imprimant recto-verso les 300 Go de caractères sur des feuilles A4 puis en empilant les feuilles les une sur les autres, la pile obtenue attendrait quasiment la taille de l'Everest. Nous avons résolu ce problème en calculant les marginales à la volée dans le programme `toulbar2`.

Les marginales pour chaque acide aminé de l'interface pour l'état lié (à gauche) et non lié (à droite) sont représentées dans la figure 5.4. Les rotamères optimaux prédits par la méthode d'optimisation sont indiqués par des étoiles blanches. Pour la plus part des acides aminés, le rotamère optimal est celui qui a la plus grande probabilité et est le même pour l'état lié et non-lié ce qui signifie que c'est le rotamère contribuant le plus à l'énergie libre.

Cependant, cela n'est pas vrai pour quelques acides aminés, notamment les acides aminés Y316, K539 et D595 (marqués en rouge). Pour ces acides aminés, le rotamère optimal représente une fraction plus petite de probabilité dans l'état non-lié que dans l'état lié. De plus, pour les acides aminés K539 et D595, dans l'état non-lié, la plus grande probabilité n'est pas représentée par le rotamère optimal. Ce comportement pourrait expliquer pourquoi, dans le cas de `1ahw`, la méthode estimant l'affinité de liaison à partir des fonctions de partition est plus précise que celle n'utilisant que les conformations d'énergie optimale. Pour les autres complexes, le rotamère optimal doit occuper une grande fraction de la contribution à l'entropie ce qui rend les contributions entropiques des autres rotamères négligeables (comme on peut le remarquer pour le reste des acides aminés de l'interface de `1ahw`).

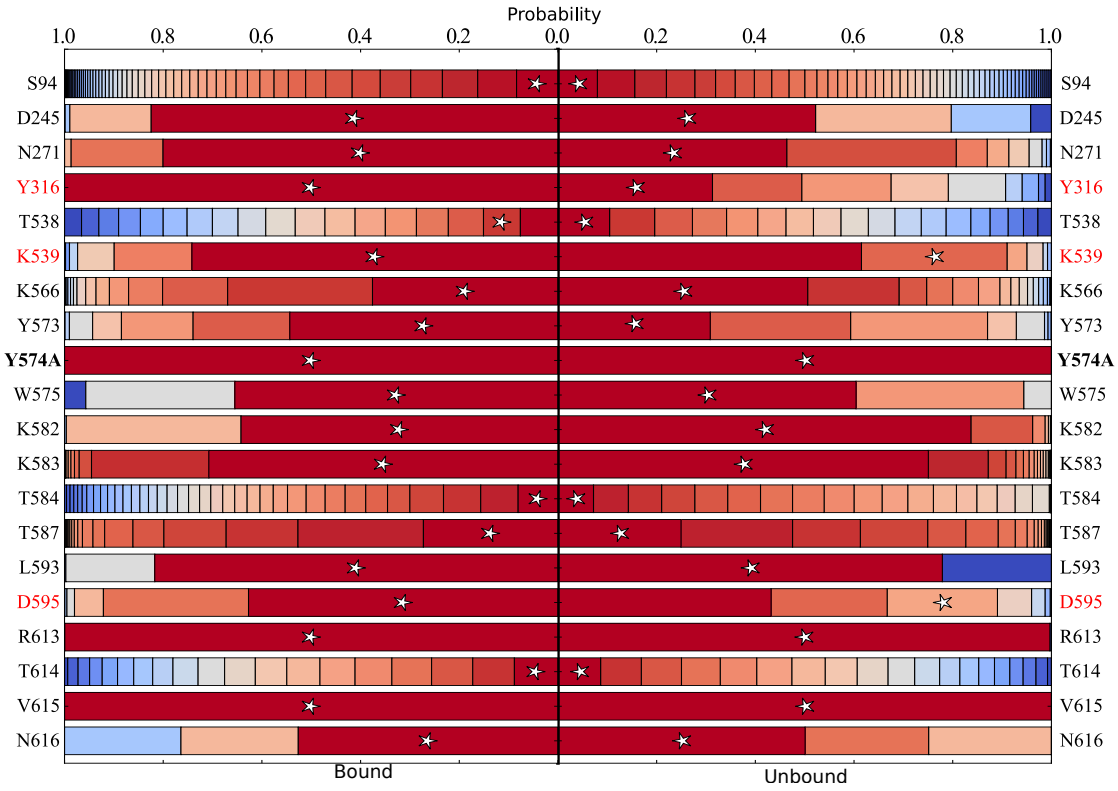


Fig. 5.4 – Distribution de probabilité marginale des rotamères pour les acides aminés de l'interface du mutant Y574A de `1ahw` pour l'état lié (gauche) et non-lié (droite). Les rotamères de la conformation d'énergie optimale sont indiqués par des étoiles blanches.

5.3 Conclusion

En s'appuyant sur des outils déterministes d'optimisation et de comptage provenant des modèles graphiques, nous avons montré qu'il est maintenant possible de concevoir des méthodes d'estimation d'affinité dont la fiabilité n'est limitée que par la modélisation du problème, basée sur la structure 3D des complexes. Ces nouvelles méthodes enlèvent toute l'incertitude que les algorithmes sans garantie peuvent présenter, tout en conservant la capacité de traiter des systèmes de taille raisonnable. De plus, il devient aussi possible d'explicitement calculer l'entropie conformationnelle des chaînes latérales que nous avons représentée dans une sorte de *heat map* contenant la contribution de chaque rotamère, pour tout acide aminé présent à l'interface du complexe.

Notre évaluation sur un grand jeu de données a permis d'obtenir de meilleurs résultats que des méthodes existantes dans le domaine, dans des conditions réalistes, sans exclusion de valeurs atypiques et sans ajustement de paramètres supplémentaires, et cela avec une simple optimisation énergétique (avec garantie). L'utilisation de techniques d'optimisation avec garanties peut être exécutée sur des problèmes contenant des centaines d'acides aminés pour un très petit temps de calcul. Dans de rares cas, prendre en compte l'entropie conformationnelle permet d'améliorer les estimations, en contrepartie d'un temps de calcul non négligeable.

Conclusions et perspectives

Nous résumons ici les contributions apportées par cette thèse et présentons quelques perspectives.

Dans le cadre du design computationnel de protéines, nous avons combiné des techniques des réseaux de fonctions de coût et des champs de Markov afin de développer des algorithmes calculant une approximation de la fonction de partition avec une garantie déterministe.

L'algorithme Z^*_ε est basé sur un élagage des conformations d'énergie négligeables et fournit la fonction de partition à une approximation ε. Z^*_ε s'est montré plus performant en terme de temps que d'autres algorithmes exacts sur 349 modèles graphiques issus d'interactions entre protéines ainsi que sur quelques instances issues de challenges officiels. Ces résultats ont mis en évidence l'importance de l'élagage de conformations d'énergie négligeable. De plus, nous avons comparé la rapidité de Z^*_ε à estimer l'affinité de liaison, c'est à dire calculer trois fonctions de partition à ε près pour chaque complexe de protéines, face à une autre méthode, K^*, estimant l'affinité de liaison avec un calcul approximatif de la fonction de partition basé sur une combinaison de `DEE`/A^*. Encore une fois Z^*_ε a prouvé qu'il était plus rapide que son confrère K^* sur un jeu de données de 30 complexes de protéines combinant un total de 1003 mutants.

De plus, en s'appuyant sur un algorithme récemment développé pour l'optimisation, nous avons développé `#HBFS`, un algorithme de comptage ayant la faculté de fournir la fonction de partition avec une garantie déterministe connue et croissante au cours du temps. Nous avons montré sur des instances issues de modélisation de complexes de protéines que `#HBFS` est plus efficace en temps que son prédécesseur Z^*_ε, pour atteindre le même niveau de garantie.

Les deux algorithmes précédents ne prennent quasiment pas en compte la structure du graphe associé au modèle graphique. À l'aide d'une décomposition arborescente, nous avons pu développer un algorithme exact `BTDZ`, permettant d'exploiter les parties conditionnellement indépendantes du graphe pour ne pas avoir à les

recalculer. Comme espéré, nous avons montré que `BTDZ` calculait la fonction de partition exactement sur des instances avec une petite largeur d'arbre, là où son confrère `#HBFS` était incapable de le faire.

D'autre part, en utilisant un algorithme d'optimisation énergétique existant (issu des CFN) et ceux élaborés durant cette thèse, nous avons développé un outil de design computationnel `EasyEJayZ`, disponible en *open source*, simple d'utilisation, entièrement automatisé, depuis la modélisation du problème jusqu'à l'estimation de l'affinité de liaison de complexes de protéines et notamment de l'impact de mutations sur les changements d'affinité de liaison. Cet outil propose deux types de méthodes qui s'appuient sur des matrices d'énergies calculées à partir du programme de design computationnel PyRosetta. La première EasyE, très rapide, est basée sur la différence en énergie des conformations d'énergies minimales entre l'état lié et non-lié, calculées par l'algorithme d'optimisation HBFS [10]. La seconde, JayZ, est basée sur un calcul approché à $\varepsilon = 10^{-3}$ près des fonctions de partition de l'état lié et non-lié afin de capturer l'entropie conformationnelle à l'interface du complexe de protéines. Les fonctions de partition ont été calculées à partir de l'algorithme Z^*_ε développé pendant la thèse. Cette seconde méthode est théoriquement plus précise moyennant un temps de calcul plus long.

Afin de tester leur efficacité, nous avons évalué nos deux méthodes en estimant l'affinité de liaison sur 21 complexes de protéine, contenant un total de 1094 mutations puis en confrontant ces estimations aux données expérimentales disponibles dans une banque de données. Pour nous situer par rapport aux méthodes de l'état de l'art, nous nous sommes comparés aux estimations provenant de FoldX, un programme de design computationnel estimant l'affinité de liaison à partir d'une fonction d'énergie empirique, et GOBLIN, un outil de design computationnel estimant l'affinité à partir de fonctions de partition calculées par un algorithme sans garantie.

Globalement, les estimations fournies par notre première méthode, basées sur les énergies des conformations d'énergie minimale, ont une corrélation de (0.41) et une AUC (*Area under the Curve*) de la courbe ROC de (70%), plus élevées que FoldX (0.24 et 58%) et GOBLIN (0.16 et 65%) avec une erreur de prédiction (RMSE) qui reste similaire à celle des autres méthodes (inférieure à 2.20 kcal.mol^{-1} pour les trois méthodes).

De plus, notre seconde méthode est venue améliorer la plus mauvaise estimation, provenant du système le plus gros. Grâce à l'algorithme `HBFS`, capable d'énumérer un très grand nombre de conformations dans un temps raisonnable, ainsi qu'à la constante de normalisation fournie par Z^*_ε, nous avons pu calculer la contribution entropique des rotamères pour chaque acide aminé présent à l'interface du complexe. Par conséquent, nous avons pu identifier la cause de la sous-estimation de

la première méthode. Il en est ressorti que certaines contributions conformationnelles à l'affinité de liaison ne pouvait pas être pris en compte seulement avec la conformation d'énergie minimale.

Perspectives

Aujourd'hui, même si `#HBFS` est performant sur des modèles graphiques variés, il peine encore à résoudre certains modèles graphiques, y compris ceux avec une petite largeur d'arbre. C'est justement cela qui nous a motivés à développer `BTDZ`. Néanmoins pour le moment, `BTDZ` est un algorithme exact et il souffre lui aussi d'une explosion combinatoire assez rapide. Il serait intéressant de mixer la recherche arborescente `#HBFS` avec `BTDZ` afin d'avoir un algorithme pouvant résoudre un panel de problèmes bien plus vaste. Cet algorithme est en ce moment en cours de développement.

Par ailleurs, de la même manière que dans l'application de cette thèse, il serait intéressant d'étudier les contributions des acides aminés sur des interactions avec des peptides. En effet, comme l'interface d'une interaction protéine-protéine est souvent large, elle peut être très stable et la variation en entropie lors d'une simple mutation peut rester limitée. De plus, le grand nombre d'acides animés présent à l'interface rend le calcul de la fonction de partition très laborieux. De ce fait, si un peptide interagit avec une protéine, il sera plus aisé d'estimer sa contribution entropique conformationnelle totale lors de son interaction. Cependant, tout comme la disponibilité de résultats expérimentaux pour des mutations multiples à l'interface de complexes protéine-protéine, ceux concernant les interactions protéine-peptide restent également limités. De ce fait, une application directe à un problème de prédiction d'affinité protéine-peptide est plus attirante. En effet, si l'on avait une protéine interessante et que l'on voulait inhiber son interaction avec une autre protéine, alors il serait possible de créer *de novo* un peptide inhibiteur de l'interaction en utilisant notre outil de design. Cela peut être fait en partant d'une structure peptidique de base puis en remedollant entièrement le peptide, c'est à dire, en testant toutes les combinaisons possibles de mutations pour chaque acide aminé (20^n mutants possibles pour un peptide de n acides aminés). Enfin, on pourrait valider expérimentalement les meilleurs candidats.

De reste, notre outil de design computationnel pourrait être amélioré afin de prendre en entrée, non pas uniquement des complexes protéine-protéine ou protéine-peptide, mais aussi des interactions de protéines avec des ligands non-peptidiques (différents types de molécules organiques). Enfin, nos méthodes prennent en compte la flexibilité de la protéine seulement au niveau des chaînes latérales, il serait donc intéressant d'étendre notre outil actuel afin qu'il soit capable de prendre aussi en

compte différentes conformations du squelette pour ainsi modéliser sa flexibilité. Cela peut être accompli en améliorant les outils de calcul de la fonction de partition pour la méthode JayZ, comme énoncé précédemment, mais aussi par l'amélioration des outils d'optimisation pour la méthode EasyE. Plus important, si les améliorations de nos outils de calcul sont suffisantes, il serait même possible de considérer des conformations des complexes dans des états totalement différents (*multi-state design*).

Bibliographie

[1] Tanja Kortemme and David Baker. Computational design of protein–protein interactions. *Current opinion in chemical biology*, 8(1) :91–97, 2004.

[2] 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 (Oxford, England)*, 22(14) :e174–83, July 2006.

[3] Alexandre Zanghellini, Lin Jiang, Andrew M Wollacott, Gong Cheng, Jens Meiler, Eric A Althoff, Daniela Röthlisberger, and David Baker. New algorithms and an in silico benchmark for computational enzyme design. *Protein Science*, 15(12) :2785–2794, 2006.

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

[5] 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*, 212 :59–79, 2014.

[6] David Simoncini, David Allouche, Simon de Givry, Celine Delmas, Sophie Barbe, and Thomas Schiex. Guaranteed discrete energy optimization on large protein design problems. *Journal of chemical theory and computation*, 11(12) :5980–5989, 2015.

[7] 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*, 37(12) :1048–1058, 2016.

[8] Clément Viricel, David Simoncini, David Allouche, Simon de Givry, Sophie Barbe, and Thomas Schiex. Approximate counting with deterministic guarantees for affinity computation. In *Modelling, Computation and Optimi-*

zation in Information Systems and Management Sciences, pages 165–176. Springer, 2015.

[9] Clément Viricel, David Simoncini, Sophie Barbe, and Thomas Schiex. Guaranteed weighted counting for affinity computation : Beyond determinism and structure. In *International Conference on Principles and Practice of Constraint Programming*, pages 733–750. Springer International Publishing, 2016.

[10] 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.

[11] Kevin Murphy. An introduction to graphical models. *Rap. tech*, pages 1–19, 2001.

[12] Daphne Koller and Nir Friedman. *Probabilistic graphical models : principles and techniques*. MIT press, 2009.

[13] Rina Dechter. Reasoning with probabilistic and deterministic graphical models : Exact algorithms. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 7(3) :1–191, 2013.

[14] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128 :1–58, 2006.

[15] Brendan J Frey. Extending factor graphs so as to unify directed and undirected graphical models. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, pages 257–264. Morgan Kaufmann Publishers Inc., 2002.

[16] Ross Kindermann and J Laurie Snell. *Markov random fields and their applications*, volume 1. American Mathematical Society, 1980.

[17] Stan Z Li. *Markov random field modeling in image analysis*. Springer Science & Business Media, 2009.

[18] Kevin P Murphy. *Machine learning : a probabilistic perspective*. MIT press, 2012.

[19] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and valued CSPs : Frameworks, properties and comparison. *Constraints*, 4 :199–240, 1999.

[20] M C. Cooper. Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems*, 134(3) :311–342, 2003.

[21] Simon de Givry, Javier Larrosa, Pedro Meseguer, and Thomas Schiex. Solving Max-Sat as weighted CSP. In *Proc. of the Ninth International Conference on Principles and Practice of Constraint Programming*, LNCS, Kinsale, Ireland, October 2003. Springer Verlag.

[22] M. Cooper, S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki, and T. Werner. Soft arc consistency revisited. *Artificial Intelligence*, 174 :449–478, 2010.

[23] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[24] Ingo A Beinlich, Henri Jacques Suermondt, R Martin Chavez, and Gregory F Cooper. The alarm monitoring system : A case study with two probabilistic inference techniques for belief networks. In *AIME 89*, pages 247–256. Springer, 1989.

[25] Daniel Zelterman. Bayesian artificial intelligence, 2005.

[26] Simon JD Prince. *Computer vision : models, learning, and inference*. Cambridge University Press, 2012.

[27] Chaohui Wang, Nikos Komodakis, and Nikos Paragios. Markov random field modeling, inference & learning in computer vision & image understanding : A survey. *Computer Vision and Image Understanding*, 117(11) :1610–1627, 2013.

[28] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning : theory and practice*. Elsevier, 2004.

[29] Leslie G Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2) :189–201, 1979.

[30] C. Lecoutre, L Saïs, S. Tabary, and V. Vidal. Reasoning from last conflict(s) in constraint programming. *Artificial Intelligence*, 173 :1592,1614, 2009.

[31] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *ECAI*, volume 16, page 146, 2004.

[32] Judea Pearl. *Heuristics : intelligent search strategies for computer problem solving*. Addison-Wesley Pub. Co., Inc., Reading, MA, 1984.

[33] 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*, 4(2) :100–107, 1968.

[34] J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted CSP. In *Proc. of the 18th IJCAI*, pages 239–244, Acapulco, Mexico, August 2003.

[35] M C. Cooper and T. Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154(1-2) :199–227, 2004.

[36] J. Larrosa, S. de Givry, F. Heras, and M. Zytnicki. Existential arc consistency : getting closer to full arc consistency in weighted CSPs. In *Proc. of the 19th IJCAI*, pages 84–89, Edinburgh, Scotland, August 2005.

[37] Martin C Cooper, Simon de Givry, Marti Sanchez, Thomas Schiex, and Matthias Zytnicki. Virtual arc consistency for weighted csp. In *AAAI*, volume 8, pages 253–258, 2008.

[38] Christophe Lecoutre, Nicolas Paris, Olivier Roussel, and Sébastien Tabary. Propagating soft table constraints. In *Principles and Practice of Constraint Programming*, pages 390–405. Springer, 2012.

[39] Hiep Nguyen, Thomas Schiex, and Christian Bessiere. Dynamic virtual arc consistency. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 98–103. ACM, 2013.

[40] J. Larrosa. On arc and node consistency in weighted CSP. In *Proc. AAAI'02*, pages 48–53, Edmondton, (CA), 2002.

[41] Neil Robertson and Paul D Seymour. Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1) :49–64, 1984.

[42] Umberto Bertelé and Francesco Brioshi. *Nonserial Dynamic Programming*. Academic Press, 1972.

[43] Radu Marinescu, Rina Dechter, and Alexander T Ihler. And/or search for marginal map. In *UAI*, pages 563–572. Citeseer, 2014.

[44] 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.

[45] Nathalie Peyrard, Simon De Givry, Alain Franc, Stéphane Robin, Régis Sabbadin, Thomas Schiex, and Matthieu Vignes. Exact and approximate inference in graphical models : variable elimination and beyond. *arXiv preprint arXiv :1506.08544*, 2015.

[46] C. Terrioux and P. Jegou. Bounded backtracking for the valued constraint satisfaction problems. In *Proc. of the Ninth International Conference on Principles and Practice of Constraint Programming (CP-2003)*, 2003.

[47] S. de Givry, T. Schiex, and G. Verfaillie. Exploiting Tree Decomposition and Soft Local Consistency in Weighted CSP. In *Proc. of the National Conference on Artificial Intelligence, AAAI-2006*, pages 22–27, 2006.

[48] Seinosuke Toda. On the computational power of pp and $\oplus$p. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 514–519. IEEE, 1989.

[49] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2) :498–519, 2001.

[50] Martin J Wainwright, Tommi S Jaakkola, and Alan S Willsky. Tree-reweighted belief propagation algorithms and approximate ml estimation by

pseudo-moment matching. In *Workshop on Artificial Intelligence and Statistics*, volume 21, page 97. Society for Artificial Intelligence and Statistics Np, 2003.

[51] Tamir Hazan, Subhransu Maji, and Tommi Jaakkola. On sampling from the Gibbs distribution with random maximum a-posteriori perturbations. In *Advances in Neural Information Processing Systems*, pages 1268–1276, 2013.

[52] Stefano Ermon, Carla P Gomes, Ashish Sabharwal, and Bart Selman. Taming the curse of dimensionality : Discrete integration by hashing and optimization. *arXiv preprint arXiv :1302.6677*, 2013.

[53] Supratik Chakraborty, Daniel J Fremont, Kuldeep S Meel, Sanjit A Seshia, and Moshe Y Vardi. Distribution-aware sampling and weighted model counting for sat. *arXiv preprint arXiv :1404.2984*, 2014.

[54] Marc Thurley. sharpsat–counting models with advanced component caching and implicit bcp. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 424–429. Springer, 2006.

[55] Tian Sang, Paul Beame, and Henry Kautz. Heuristics for fast exact model counting. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 226–240. Springer, 2005.

[56] Rina Dechter. Bucket elimination : A unifying framework for reasoning. *Artificial Intelligence*, 113(1–2) :41–85, 1999.

[57] Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6) :772–799, 2008.

[58] Umut Oztok and Adnan Darwiche. A top-down compiler for sentential decision diagrams. In *Proceedings of the 24th International Conference on Artificial Intelligence. AAAI Press*, 2015.

[59] Tian Sang, Paul Beame, and Henry Kautz. Solving bayesian networks by weighted model counting. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, volume 1, pages 475–482, 2005.

[60] G. Shafer. An axiomatic study of computation in hypertrees. Working paper 232, University of Kansas, School of Business, Lawrence, 1991.

[61] Roope Kaivola, Rajnish Ghughal, Naren Narasimhan, Amber Telfer, Jesse Whittemore, Sudhindra Pandav, Anna Slobodová, Christopher Taylor, Vladimir A Frolov, Erik Reeber, et al. Replacing testing with formal verification in intel coretm i7 processor execution engine validation. In *CAV*, volume 9, pages 414–429. Springer, 2009.

[62] Adnan Darwiche. A logical approach to factoring belief networks. *KR*, 2 :409–420, 2002.

[63] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff : Engineering an efficient sat solver. In *38th Design Automation Conference (DAC'01)*, pages 530–535, June 2001.

[64] Yogesh S Mahajan, Zhaohui Fu, and Sharad Malik. Zchaff2004 : An efficient sat solver. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 360–375. Springer, 2004.

[65] Paul Beame, Russell Impagliazzo, Toniann Pitassi, and Nathan Segerlind. Memoization and dpll : Formula caching proof systems. In *Computational Complexity, 2003. Proceedings. 18th IEEE Annual Conference on*, pages 248–259. IEEE, 2003.

[66] Stephen M Majercik and Michael L Littman. Using caching to solve larger probabilistic planning problems. In *AAAI/IAAI*, pages 954–959, 1998.

[67] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Algorithms and complexity results for# sat and bayesian inference. In *Foundations of computer science, 2003. proceedings. 44th annual ieee symposium on*, pages 340–351. IEEE, 2003.

[68] João P Marques Silva and Karem A Sakallah. Grasp ? ? ?a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 220–227. IEEE Computer Society, 1997.

[69] Hantao Zhang. Sato : An efficient prepositional prover. *Automated Deduction ? ? ?CADE-14*, pages 272–275, 1997.

[70] Lintao Zhang, Conor F Madigan, Matthew H Moskewicz, and Sharad Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, pages 279–285. IEEE Press, 2001.

[71] Roberto J Bayardo Jr and Joseph Daniel Pehoushek. Counting models using connected components. In *AAAI/IAAI*, pages 157–162, 2000.

[72] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Value elimination : Bayesian inference via backtracking search. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, pages 20–28. Morgan Kaufmann Publishers Inc., 2002.

[73] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17(1) :229–264, 2002.

[74] Adnan Darwiche and Pierre Marquis. A perspective on knowledge compilation. In *IJCAI*, volume 1, pages 175–182, 2001.

[75] Scott Kirkpatrick. Optimization by simulated annealing : Quantitative studies. *Journal of statistical physics*, 34(5) :975–986, 1984.

[76] Radford M Neal. *Probabilistic inference using Markov chain Monte Carlo methods*. PhD thesis, Department of Computer Science, University of Toronto Toronto, Ontario, Canada, 1993.

[77] Radford M Neal. Annealed importance sampling. *Statistics and Computing*, 11(2) :125–139, 2001.

[78] Jianzhu Ma, Jian Peng, Sheng Wang, and Jinbo Xu. Estimating the partition function of graphical models using langevin importance sampling. In *AISTATS*, pages 433–441, 2013.

[79] Judea Pearl. *Reverend Bayes on inference engines : A distributed hierarchical approach*. Cognitive Systems Laboratory, School of Engineering and Applied Science, University of California, Los Angeles, 1982.

[80] Jonathan S Yedidia, William T Freeman, Yair Weiss, et al. Generalized belief propagation. In *NIPS*, volume 13, pages 689–695, 2000.

[81] Kevin P Murphy, Yair Weiss, and Michael I Jordan. Loopy belief propagation for approximate inference : An empirical study. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 467–475. Morgan Kaufmann Publishers Inc., 1999.

[82] Claude Berrou, Alain Glavieux, and Punya Thitimajshima. Near shannon limit error-correcting coding and decoding : Turbo-codes. 1. In *Communications, 1993. ICC'93 Geneva. Technical Program, Conference Record, IEEE International Conference on*, volume 2, pages 1064–1070. IEEE, 1993.

[83] Jonathan S Yedidia, William T Freeman, and Yair Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7) :2282–2312, 2005.

[84] Justin Domke. Learning graphical model parameters with approximate marginal inference. *IEEE transactions on pattern analysis and machine intelligence*, 35(10) :2454–2467, 2013.

[85] Martin J Wainwright, Tommi S Jaakkola, and Alan S Willsky. A new class of upper bounds on the log partition function. *Information Theory, IEEE Transactions on*, 51(7) :2313–2335, 2005.

[86] Martin J Wainwright and Michael I Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1-2) :1–305, 2008.

[87] Joris M. Mooij. libDAI : A free and open source C++ library for discrete approximate inference in graphical models. *Journal of Machine Learning Research*, 11 :2169–2173, August 2010.

[88] Qiang Liu and Alexander T Ihler. Bounding the partition function using holder's inequality. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 849–856, 2011.

[89] Rina Dechter. Mini-buckets : A general scheme for generating approximations in automated reasoning. In *Proc. of the* 16^{th} *IJCAI*, pages 1297–1303, 1997.

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

[91] Lieping Chen and Xue Han. Anti–pd-1/pd-l1 therapy of human cancer : past, present, and future. *The Journal of clinical investigation*, 125(9) :3384–3391, 2015.

[92] Fabrizio Chiti and Christopher M Dobson. Protein misfolding, functional amyloid, and human disease. *Annu. Rev. Biochem.*, 75 :333–366, 2006.

[93] Errol C Friedberg, Graham C Walker, Wolfram Siede, and Richard D Wood. *DNA repair and mutagenesis*. American Society for Microbiology Press, 2005.

[94] Bruce Alberts, Dennis Bray, Julian Lewis, Martin Raff, Keith Roberts, James D Watson, and AV Grimstone. Molecular biology of the cell (3rd edn). *Trends in Biochemical Sciences*, 20(5) :210–210, 1995.

[95] William Clegg. Crystal structure determination. *Oxford Chemistry Primers*, 60(1) :ALL–ALL, 1998.

[96] John D Roberts. *Nuclear magnetic resonance : applications to organic chemistry*. McGraw-Hill Book Company, Inc, 1959.

[97] 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*, 28(1) :235–242, 2000.

[98] Christopher M Dobson. Protein folding and misfolding. *Nature*, 426(6968) :884, 2003.

[99] Ken A Dill and Justin L MacCallum. The protein-folding problem, 50 years on. *science*, 338(6110) :1042–1046, 2012.

[100] John Moult, Krzysztof Fidelis, Andriy Kryshtafovych, Torsten Schwede, and Anna Tramontano. Critical assessment of methods of protein structure prediction : Progress and new directions in round xi. *Proteins : Structure, Function, and Bioinformatics*, 84(S1) :4–14, 2016.

[101] Katherine Henzler-Wildman and Dorothee Kern. Dynamic personalities of proteins. *Nature*, 450(7172) :964, 2007.

[102] 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*, 450(7171) :913, 2007.

[103] Gordon G Hammes, Stephen J Benkovic, and Sharon Hammes-Schiffer. Flexibility, diversity, and cooperativity : pillars of enzyme catalysis. *Biochemistry*, 50(48) :10422–10430, 2011.

[104] Cyrus Levinthal. Are there pathways for protein folding ? *Journal de chimie physique*, 65 :44–45, 1968.

[105] JM Haile. *Molecular dynamics simulation*, volume 18. Wiley, New York, 1992.

[106] Massimiliano Mangoni, Danilo Roccatano, and Alfredo Di Nola. Docking of flexible ligands to flexible receptors in solution by molecular dynamics simulation. *Proteins : Structure, Function, and Bioinformatics*, 35(2) :153–162, 1999.

[107] John R Desjarlais and Tracy M Handel. Side-chain and backbone flexibility in protein core design. *Journal of molecular biology*, 290(1) :305–318, 1999.

[108] John L Klepeis, Kresten Lindorff-Larsen, Ron O Dror, and David E Shaw. Long-timescale molecular dynamics simulations of protein structure and function. *Current opinion in structural biology*, 19(2) :120–127, 2009.

[109] Andrew R Leach. Ligand docking to proteins with discrete side-chain flexibility. *Journal of molecular biology*, 235(1) :345–356, 1994.

[110] Ian W Davis, W Bryan Arendall, David C Richardson, and Jane S Richardson. The backrub motion : how protein backbone shrugs when a sidechain dances. *Structure*, 14(2) :265–274, 2006.

[111] Mark A Hallen, Daniel A Keedy, and Bruce R Donald. Dead-end elimination with perturbations (deeper) : A provable protein design algorithm with continuous sidechain and backbone flexibility. *Proteins : Structure, Function, and Bioinformatics*, 81(1) :18–39, 2013.

[112] Roland L Dunbrack. Rotamer libraries in the 21 st century. *Current opinion in structural biology*, 12(4) :431–440, 2002.

[113] Ho Ki Fung, William J Welsh, and Christodoulos A Floudas. Computational de novo peptide and protein design : rigid templates versus flexible templates. *Industrial & Engineering Chemistry Research*, 47(4) :993–1001, 2008.

[114] Jeffrey C Moore and Frances H Arnold. Directed evolution of a para-nitrobenzyl esterase for aqueous-organic solvents. *Nature biotechnology*, 14(4) :458–467, 1996.

[115] Christopher A Voigt, Stephen L Mayo, Frances H Arnold, and Zhen-Gang Wang. Computational method to reduce the search space for directed protein evolution. *Proceedings of the National Academy of Sciences*, 98(7) :3778–3783, 2001.

[116] Carl Pabo. Molecular technology : designing proteins and peptides. *Nature*, 301(5897) :200–200, 1983.

[117] K Eric Drexler. Molecular engineering : An approach to the development of general capabilities for molecular manipulation. *Proceedings of the National Academy of Sciences*, 78(9) :5275–5278, 1981.

[118] Tim Harder, Wouter Boomsma, Martin Paluszewski, Jes Frellsen, Kristoffer E Johansson, and Thomas Hamelryck. Beyond rotamers : a generative, probabilistic model of side chains in proteins. *BMC bioinformatics*, 11(1) :306, 2010.

[119] Maxim V Shapovalov and Roland L Dunbrack. A smoothed backbone-dependent rotamer library for proteins derived from adaptive kernel density estimates and regressions. *Structure*, 19(6) :844–858, 2011.

[120] S C Lovell, J M Word, J S Richardson, and D C Richardson. The penultimate rotamer library. *Proteins*, 40(3) :389–408, August 2000.

[121] Christian B Anfinsen. Principles that govern the folding of protein chains. *Science*, 181(4096) :223–230, 1973.

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

[123] Themis Lazaridis and Martin Karplus. Discrimination of the native from misfolded protein models with an energy function including implicit solvation. *Journal of molecular biology*, 288(3) :477–487, 1999.

[124] John R Desjarlais and Tracy M Handel. De novo design of the hydrophobic cores of proteins. *Protein Science*, 4(10) :2006–2018, 1995.

[125] B Kuhlman and D Baker. Native protein sequences are close to optimal for their structures. *Proceedings of the National Academy of Sciences of the United States of America*, 97(19) :10383–8, September 2000.

[126] Pedro EM Lopes, Olgun Guvench, and Alexander D MacKerell. Current status of protein force fields for molecular dynamics simulations. *Molecular Modeling of Proteins*, pages 47–71, 2015.

[127] Wendy D Cornell, Piotr Cieplak, Christopher I Bayly, Ian R Gould, Kenneth M Merz, David M Ferguson, David C Spellmeyer, Thomas Fox, James W Caldwell, and Peter A Kollman. A second generation force field for the simulation of proteins, nucleic acids, and organic molecules. *Journal of the American Chemical Society*, 117(19) :5179–5197, 1995.

[128] David A Pearlman, David A Case, James W Caldwell, Wilson S Ross, Thomas E Cheatham, Steve DeBolt, David Ferguson, George Seibel, and Peter Kollman. Amber, a package of computer programs for applying molecular

mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules. *Computer Physics Communications*, 91(1) :1–41, 1995.

[129] Alex D MacKerell Jr, Donald Bashford, MLDR Bellott, Roland Leslie Dunbrack Jr, Jeffrey D Evanseck, Martin J Field, Stefan Fischer, Jiali Gao, H Guo, Sookhee Ha, et al. All-atom empirical potential for molecular modeling and dynamics studies of proteins. *The journal of physical chemistry B*, 102(18) :3586–3616, 1998.

[130] Bernard R Brooks, Charles L Brooks, Alexander D MacKerell, 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*, 30(10) :1545–1614, 2009.

[131] Chris Oostenbrink, Alessandra Villa, Alan E Mark, and Wilfred F Van Gunsteren. A biomolecular force field based on the free enthalpy of hydration and solvation : the gromos force-field parameter sets 53a5 and 53a6. *Journal of computational chemistry*, 25(13) :1656–1676, 2004.

[132] Zsuzsanna Dosztanyi, Veronika Csizmok, Peter Tompa, and Istvan Simon. The pairwise energy content estimated from amino acid composition discriminates between folded and intrinsically unstructured proteins. *Journal of molecular biology*, 347(4) :827–839, 2005.

[133] Kaushik Raha, Arjan J van der Vaart, Kevin E Riley, Martin B Peters, Lance M Westerhoff, Hwanho Kim, and Kenneth M Merz. Pairwise decomposition of residue interaction energies using semiempirical quantum mechanical methods in studies of protein- ligand interaction. *Journal of the American Chemical Society*, 127(18) :6583–6594, 2005.

[134] Andrew Leaver-Fay, Michael Tyka, Steven M Lewis, Oliver F Lange, James Thompson, Ron Jacak, Kristian Kaufman, P Douglas Renfrew, Colin A Smith, Will Sheffler, et al. Rosetta3 : an object-oriented software suite for the simulation and design of macromolecules. *Methods in enzymology*, 487 :545, 2011.

[135] Pablo Gainza, Kyle E Roberts, Ivelin Georgiev, Ryan H Lilien, Daniel A Keedy, Cheng-Yu Chen, Faisal Reza, Amy C Anderson, David C Richardson, Jane S Richardson, et al. Osprey : Protein design with ensembles, flexibility, and provable algorithms. *Methods Enzymol*, 2012.

[136] Carol A Rohl, Charlie EM Strauss, Kira MS Misura, and David Baker. Protein structure prediction using rosetta. *Methods in enzymology*, 383 :66–93, 2004.

[137] Lin Jiang, Eric A Althoff, Fernando R Clemente, Lindsey Doyle, Daniela Röthlisberger, Alexandre Zanghellini, Jasmine L Gallaher, Jamie L Betker,

Fujie Tanaka, Carlos F Barbas, et al. De novo computational design of retro-aldol enzymes. *science*, 319(5868) :1387–1391, 2008.

[138] 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*, 453(7192) :190, 2008.

[139] Daniel J Mandell, Evangelos A Coutsias, and Tanja Kortemme. Sub-angstrom accuracy in protein loop reconstruction by robotics-inspired conformational sampling. *Nature methods*, 6(8) :551–552, 2009.

[140] Dominik Gront, Daniel W Kulp, Robert M Vernon, Charlie EM Strauss, and David Baker. Generalized fragment picking in rosetta : design, protocols and applications. *PloS one*, 6(8) :e23294, 2011.

[141] 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*, 29(10) :1527–42, July 2008.

[142] 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*, 8(4) :e1002477–e1002477, 2012.

[143] Ivelin S Georgiev, Rebecca S Rudicell, Kevin O Saunders, Wei Shi, Tatsiana Kirys, Krisha McKee, Sijy O'Dell, Gwo-Yu Chuang, Zhi-Yong Yang, Gilad Ofek, et al. Antibodies vrc01 and 10e8 neutralize hiv-1 with high breadth and potency even with ig-framework regions substantially reverted to germline. *The Journal of Immunology*, 192(3) :1100–1106, 2014.

[144] Rebecca S Rudicell, Young Do Kwon, Sung-Youl Ko, Amarendra Pegu, Mark K Louder, Ivelin S Georgiev, Xueling Wu, Jiang Zhu, Jeffrey C Boyington, Xuejun Chen, et al. Enhanced potency of a broadly neutralizing hiv-1 antibody in vitro improves protection against lentiviral infection in vivo. *Journal of virology*, 88(21) :12669–12682, 2014.

[145] Kim T Simons, Charles Kooperberg, Enoch Huang, and David Baker. Assembly of protein tertiary structures from fragments with similar local sequences using simulated annealing and bayesian scoring functions. *Journal of molecular biology*, 268(1) :209–225, 1997.

[146] Kim T Simons, Ingo Ruczinski, Charles Kooperberg, Brian A Fox, Chris Bystroff, and David Baker. Improved recognition of native-like protein structures using a combination of sequence-dependent and sequence-independent features of proteins. *Proteins : Structure, Function, and Bioinformatics*, 34(1) :82–95, 1999.

[147] Tanja Kortemme and David Baker. A simple physical model for binding energy hot spots in protein–protein complexes. *Proceedings of the National Academy of Sciences*, 99(22) :14116–14121, 2002.

[148] Jeffrey J Gray, Stewart Moughon, Chu Wang, Ora Schueler-Furman, Brian Kuhlman, Carol A Rohl, and David Baker. Protein–protein docking with simultaneous optimization of rigid-body displacement and side-chain conformations. *Journal of molecular biology*, 331(1) :281–299, 2003.

[149] Tanja Kortemme, David E Kim, and David Baker. Computational alanine scanning of protein-protein interfaces. *Sci. STKE*, 2004(219) :pl2–pl2, 2004.

[150] Philip Bradley, Kira MS Misura, and David Baker. Toward high-resolution de novo structure prediction for small proteins. *Science*, 309(5742) :1868–1871, 2005.

[151] Tanja Kortemme, Lukasz A Joachimiak, Alex N Bullock, Aaron D Schuler, Barry L Stoddard, and David Baker. Computational redesign of protein-protein interaction specificity. *Nature structural & molecular biology*, 11(4) :371, 2004.

[152] Krishna Praneeth Kilambi, Kavan Reddy, and Jeffrey J Gray. Protein-protein docking with dynamic residue protonation states. *PLoS computational biology*, 10(12) :e1004018, 2014.

[153] Rebecca F Alford, Julia Koehler Leman, Brian D Weitzner, Amanda M Duran, Drew C Tilley, Assaf Elazar, and Jeffrey J Gray. An integrated framework advancing membrane protein modeling and design. *PLoS computational biology*, 11(9) :e1004398, 2015.

[154] 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*, 2017.

[155] John Edward Jones. On the determination of molecular fields. i. from the variation of the viscosity of a gas with temperature. *Proceedings of the Royal Society of London A : Mathematical, Physical and Engineering Sciences*, 106(738) :441–462, 1924.

[156] John Edward Jones. On the determination of molecular fields. ii. from the equation of state of a gas. *Proceedings of the Royal Society of London A : Mathematical, Physical and Engineering Sciences*, 106(738) :463–477, 1924.

[157] Themis Lazaridis and Martin Karplus. Effective energy function for proteins in solution. *Proteins : Structure, Function, and Bioinformatics*, 35(2) :133–152, 1999.

[158] 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*, 12(12) :6201–6212, 2016.

[159] Chen Yanover and Philip Bradley. Extensive protein and dna backbone sampling improves structure-based specificity prediction for c2h2 zinc fingers. *Nucleic acids research*, 39(11) :4564–4576, 2011.

[160] Tanja Kortemme, Alexandre V Morozov, and David Baker. An orientation-dependent hydrogen bonding potential improves prediction of specificity and structure for proteins and protein–protein complexes. *Journal of molecular biology*, 326(4) :1239–1259, 2003.

[161] Matthew J O'Meara, Andrew Leaver-Fay, Michael D Tyka, Amelie Stein, Kevin Houlihan, Frank DiMaio, Philip Bradley, Tanja Kortemme, David Baker, Jack Snoeyink, et al. Combined covalent-electrostatic model of hydrogen bonding improves structure prediction with rosetta. *Journal of chemical theory and computation*, 11(2) :609–622, 2015.

[162] Andrew Leaver-Fay, Matthew J O'Meara, Mike Tyka, Ron Jacak, Yifan Song, Elizabeth H Kellogg, James Thompson, Ian W Davis, Roland A Pache, Sergey Lyskov, et al. Scientific benchmarks for guiding macromolecular energy function improvement. *Methods in enzymology*, 523 :109, 2013.

[163] Maxim V Shapovalov and Roland L Dunbrack. A smoothed backbone-dependent rotamer library for proteins derived from adaptive kernel density estimates and regressions. *Structure*, 19(6) :844–858, 2011.

[164] Donald S Berkholz, Camden M Driggers, Maxim V Shapovalov, Roland L Dunbrack, and P Andrew Karplus. Nonplanar peptide bonds in proteins are common and conserved but not biased toward active sites. *Proceedings of the National Academy of Sciences*, 109(2) :449–453, 2012.

[165] HW Hellinga and FM Richards. Optimal sequence selection in proteins of known structure by simulated evolution. *Proceedings of the National Academy of Sciences*, 91(13) :5803–5807, 1994.

[166] Ken A Dill. Dominant forces in protein folding. *Biochemistry*, 29(31) :7133–7155, 1990.

[167] Bassil I Dahiyat and Stephen L Mayo. De novo protein design : fully automated sequence selection. *Science*, 278(5335) :82–87, 1997.

[168] Scott TR Walsh, Hong Cheng, James W Bryson, Heinrich Roder, and William F DeGrado. Solution structure and dynamics of a de novo designed three-helix bundle protein. *Proceedings of the National Academy of Sciences*, 96(10) :5486–5491, 1999.

[169] James W Bryson, John R Desjarlais, Tracy M Handel, and William F Degrado. From coiled coils to small globular proteins : Design of a native-like three-helix bundle. *Protein Science*, 7(6) :1404–1414, 1998.

[170] 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*, 302(5649) :1364–1368, 2003.

[171] Arnout RD Voet, Hiroki Noguchi, Christine Addy, David Simoncini, Daiki Terada, Satoru Unzai, Sam-Yong Park, Kam YJ Zhang, and Jeremy RH Tame. Computational design of a self-assembling symmetrical β-propeller protein. *Proceedings of the National Academy of Sciences*, 111(42) :15102–15107, 2014.

[172] Christine E Tinberg, Sagar D Khare, Jiayi Dou, Lindsey Doyle, Jorgen W Nelson, Alberto Schena, Wojciech Jankowski, Charalampos G Kalodimos, Kai Johnsson, Barry L Stoddard, et al. Computational design of ligand-binding proteins with high affinity and selectivity. *Nature*, 501(7466) :212–216, 2013.

[173] Jaafar N Haidar, Brian Pierce, Yong Yu, Weiwei Tong, Michael Li, and Zhiping Weng. Structure-based design of a t-cell receptor leads to nearly 100-fold improvement in binding affinity for pepmhc. *Proteins : Structure, Function, and Bioinformatics*, 74(4) :948–960, 2009.

[174] Brian G Pierce, Lance M Hellman, Moushumi Hossain, Nishant K Singh, Craig W Vander Kooi, Zhiping Weng, and Brian M Baker. Computational design of the affinity and specificity of a therapeutic t cell receptor. *PLoS Comput Biol*, 10(2) :e1003478, 2014.

[175] Daniel N Bolon and Stephen L Mayo. Enzyme-like proteins by computational design. *Proceedings of the National Academy of Sciences*, 98(25) :14274–14279, 2001.

[176] Sehat Nauli, Brian Kuhlman, and David Baker. Computer-based redesign of a protein folding pathway. *Nature structural & molecular biology*, 8(7) :602, 2001.

[177] Loren L Looger, Mary A Dwyer, James J Smith, and Homme W Hellinga. Computational design of receptor and sensor proteins with novel functions. *Nature*, 423(6936) :185, 2003.

[178] Gautam Dantas, Brian Kuhlman, David Callender, Michelle Wong, and David Baker. A large scale test of computational protein design : folding and stability of nine completely redesigned globular proteins. *Journal of molecular biology*, 332(2) :449–460, 2003.

[179] Shaun M Lippow and Bruce Tidor. Progress in computational protein design. *Current opinion in biotechnology*, 18(4) :305–311, 2007.

[180] Stuart A Sievers, John Karanicolas, Howard W Chang, Anni Zhao, Lin Jiang, Onofrio Zirafi, Jason T Stevens, Jan Münch, David Baker, and David Eisenberg. Structure-based design of non-natural amino acid inhibitors of amyloid fibrillation. *Nature*, 475(7354) :96, 2011.

[181] 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*, 5(2) :1186–1198, 2015.

[182] Pietro Sormanni, Francesco A Aprile, and Michele Vendruscolo. Rational design of antibodies targeting specific epitopes within intrinsically disordered proteins. *Proceedings of the National Academy of Sciences*, 112(32) :9902–9907, 2015.

[183] Jorge A Fallas, George Ueda, William Sheffler, Vanessa Nguyen, Dan E McNamara, Banumathi Sankaran, Jose Henrique Pereira, Fabio Parmeggiani, TJ Brunette, Duilio Cascio, et al. Computational design of self-assembling cyclic protein homo-oligomers. *Protein Expression and Purification*, 28 :29, 2016.

[184] Eva-Maria Strauch, Steffen M Bernard, David La, Alan J Bohn, Peter S Lee, Caitlin E Anderson, Travis Nieusma, Carly A Holstein, Natalie K Garcia, Kathryn A Hooper, et al. Computational design of trimeric influenza-neutralizing proteins targeting the hemagglutinin receptor binding site. *Nature Biotechnology*, 35(7) :667–671, 2017.

[185] Shigeki Kiyonaka, Ryou Kubota, Yukiko Michibata, Masayoshi Sakakura, Hideo Takahashi, Tomohiro Numata, Ryuji Inoue, Michisuke Yuzaki, and Itaru Hamachi. Allosteric activation of membrane-bound glutamate receptors using coordination chemistry within living cells. *Nature Chemistry*, 2016.

[186] Priyabrata Pattnaik. Surface plasmon resonance. *Applied biochemistry and biotechnology*, 126(2) :79–92, 2005.

[187] Adrián Velázquez-Campoy, Hiroyasu Ohtaka, Azin Nezami, Salman Muzammil, and Ernesto Freire. Isothermal titration calorimetry. *Current protocols in cell biology*, pages 17–8, 2004.

[188] Yutaka Tokiwa, Buenaventurada P Calabia, Charles U Ugwu, and Seiichi Aiba. Biodegradability of plastics. *International journal of molecular sciences*, 10(9) :3722–3742, 2009.

[189] Panagiotis L Kastritis and Alexandre MJJ Bonvin. On the binding affinity of macromolecular interactions : daring to ask why proteins interact. *Journal of The Royal Society Interface*, 10(79) :20120835, 2013.

[190] Gabor Mocz and Justin A Ross. Fluorescence techniques in analysis of protein–ligand interactions. *Protein-Ligand Interactions : Methods and Applications*, pages 169–210, 2013.

[191] Kurt S Thorn and Andrew A Bogan. Asedb : a database of alanine mutations and their effects on the free energy of binding in protein interactions. *Bioinformatics*, 17(3) :284–285, 2001.

[192] MD Shaji Kumar and M Michael Gromiha. Pint : protein–protein interactions thermodynamic database. *Nucleic acids research*, 34(suppl_1) :D195–D198, 2006.

[193] Iain H Moal and Juan Fernández-Recio. Skempi : a structural kinetic and energetic database of mutant protein interactions and its use in empirical models. *Bioinformatics*, 28(20) :2600–2607, 2012.

[194] Zhihai Liu, Yan Li, Li Han, Jie Li, Jie Liu, Zhixiong Zhao, Wei Nie, Yuchen Liu, and Renxiao Wang. Pdb-wide collection of binding data : current status of the pdbbind database. *Bioinformatics*, 31(3) :405–412, 2014.

[195] Thom Vreven, Iain H Moal, Anna Vangone, Brian G Pierce, Panagiotis L Kastritis, Mieczyslaw Torchala, Raphael Chaleil, Brian Jiménez-García, Paul A Bates, Juan Fernandez-Recio, et al. Updates to the integrated protein–protein interaction benchmarks : docking benchmark version 5 and affinity benchmark version 2. *Journal of molecular biology*, 427(19) :3031–3041, 2015.

[196] Xueling Wu, Tongqing Zhou, Jiang Zhu, Baoshan Zhang, Ivelin Georgiev, Charlene Wang, Xuejun Chen, Nancy S Longo, Mark Louder, Krisha McKee, et al. Focused evolution of hiv-1 neutralizing antibodies revealed by structures and deep sequencing. *Science*, 333(6049) :1593–1602, 2011.

[197] Timothy A Whitehead, Aaron Chevalier, Yifan Song, Cyrille Dreyfus, Sarel J Fleishman, Cecilia De Mattos, Chris A Myers, Hetunandan Kamisetty, Patrick Blair, Ian A Wilson, et al. Optimization of affinity, specificity and function of designed influenza inhibitors using deep sequencing. *Nature biotechnology*, 30(6) :543–548, 2012.

[198] William P Robins, Shah M Faruque, and John J Mekalanos. Coupling mutagenesis and parallel deep sequencing to probe essential residues in a genome or gene. *Proceedings of the National Academy of Sciences*, 110(9) :E848–E857, 2013.

[199] Josiah Willard Gibbs. *A method of geometrical representation of the thermodynamic properties of substances by means of surfaces*. Connecticut Academy, 1873.

[200] Terrell L Hill. *Cooperativity theory in biochemistry : steady-state and equilibrium systems*. Springer Science & Business Media, 2013.

[201] Nathaniel W Silver, Bracken M King, Madhavi NL Nalam, Hong Cao, Akbar Ali, GS Kiran Kumar Reddy, Tariq M Rana, Celia A Schiffer, and Bruce Tidor. Efficient computation of small-molecule configurational binding entropy and free energy changes by ensemble enumeration. *Journal of chemical theory and computation*, 9(11) :5098–5115, 2013.

[202] Giulio Rastelli, Alberto Del Rio, Gianluca Degliesposti, and Miriam Sgobba. Fast and accurate predictions of binding free energies using mm-pbsa and mm-gbsa. *Journal of computational chemistry*, 31(4) :797–810, 2010.

[203] Tingjun Hou, Junmei Wang, Youyong Li, and Wei Wang. Assessing the performance of the mm/pbsa and mm/gbsa methods. 1. the accuracy of binding free energy calculations based on molecular dynamics simulations. *Journal of chemical information and modeling*, 51(1) :69–82, 2010.

[204] Irina Massova and Peter A Kollman. Combined molecular mechanical and continuum solvent approach (mm-pbsa/gbsa) to predict ligand binding. *Perspectives in drug discovery and design*, 18(1) :113–135, 2000.

[205] Peter A Kollman, Irina Massova, Carolina Reyes, Bernd Kuhn, Shuanghong Huo, Lillian Chong, Matthew Lee, Taisung Lee, Yong Duan, Wei Wang, et al. Calculating structures and free energies of complex molecules : combining molecular mechanics and continuum models. *Accounts of chemical research*, 33(12) :889–897, 2000.

[206] Christopher Jarzynski. Nonequilibrium equality for free energy differences. *Physical Review Letters*, 78(14) :2690, 1997.

[207] Johan Åqvist, Victor B Luzhkov, and Bjørn O Brandsdal. Ligand binding affinities from md simulations. *Accounts of chemical research*, 35(6) :358–365, 2002.

[208] Holger Gohlke, Christina Kiel, and David A Case. Insights into protein–protein binding by binding free energy calculation and free energy decomposition for the ras–raf and ras–ralgds complexes. *Journal of molecular biology*, 330(4) :891–913, 2003.

[209] David L Beveridge and FM DiCapua. Free energy via molecular simulation : applications to chemical and biomolecular systems. *Annual review of biophysics and biophysical chemistry*, 18(1) :431–492, 1989.

[210] Surjit B Dixit and Christophe Chipot. Can absolute free energies of association be estimated from molecular mechanical simulations ? the biotin-streptavidin system revisited. *The Journal of Physical Chemistry A*, 105(42) :9795–9799, 2001.

[211] Christophe Chipot. Frontiers in free-energy calculations of biological systems. *Wiley Interdisciplinary Reviews : Computational Molecular Science*, 4(1) :71–89, 2014.

[212] Daan Frenkel and Berend Smit. *Understanding molecular simulation : from algorithms to applications*, volume 1. Academic press, 2001.

[213] Raphael Guerois, Jens Erik Nielsen, and Luis Serrano. Predicting changes in the stability of proteins and protein complexes : a study of more than 1000 mutations. *Journal of molecular biology*, 320(2) :369–387, 2002.

[214] Joost Schymkowitz, Jesper Borg, Francois Stricher, Robby Nys, Frederic Rousseau, and Luis Serrano. The foldx web server : an online force field. *Nucleic acids research*, 33(suppl 2) :W382–W388, 2005.

[215] Yves Dehouck, Jean Marc Kwasigroch, Marianne Rooman, and Dimitri Gilis. Beatmusic : prediction of changes in protein–protein binding affinity on mutations. *Nucleic acids research*, 41(W1) :W333–W339, 2013.

[216] Jeffrey R Brender and Yang Zhang. Predicting the effect of mutations on protein-protein binding interactions through structure-based interface profiles. *PLoS Comput Biol*, 11(10) :e1004494, 2015.

[217] Minghui Li, Marharyta Petukh, Emil Alexov, and Anna R Panchenko. Predicting the impact of missense mutations on protein–protein binding affinity. *Journal of chemical theory and computation*, 10(4) :1770–1780, 2014.

[218] Minghui Li, Franco L Simonetti, Alexander Goncearenco, and Anna R Panchenko. Mutabind estimates and interprets the effects of sequence variants on protein–protein interactions. *Nucleic acids research*, page gkw374, 2016.

[219] Simon Marillet, Pierre Boudinot, and Frédéric Cazals. High-resolution crystal structures leverage protein binding affinity predictions. *Proteins : structure, function, and bioinformatics*, 84(1) :9–20, 2016.

[220] Hetunandan Kamisetty, Arvind Ramanathan, Chris Bailey-Kellogg, and Christopher James Langmead. Accounting for conformational entropy in predicting binding free energies of protein-protein interactions. *Proteins : Structure, Function, and Bioinformatics*, 79(2) :444–462, 2011.

[221] Christina Kiel and Luis Serrano. Structure-energy-based predictions and network modelling of rasopathy and cancer missense mutations. *Molecular systems biology*, 10(5) :727, 2014.

[222] Tzvia Selzer, Shira Albeck, and Gideon Schreiber. Rational design of faster associating and tighter binding protein complexes. *Nature Structural & Molecular Biology*, 7(7) :537–541, 2000.

[223] Ruben Abagyan and Maxim Totrov. Biased probability monte carlo conformational searches and electrostatic calculations for peptides and proteins. *Journal of molecular biology*, 235(3) :983–1002, 1994.

[224] Yves Dehouck, Dimitri Gilis, and Marianne Rooman. Design of modified proteins using knowledge-based approaches. In *AIP Conference Proceedings*, volume 1456, pages 139–147. AIP, 2012.

[225] Yves Dehouck, Aline Grosfils, Benjamin Folch, Dimitri Gilis, Philippe Bogaerts, and Marianne Rooman. Fast and accurate predictions of protein stability changes upon mutations using statistical potentials and neural networks : Popmusic-2.0. *Bioinformatics*, 25(19) :2537–2543, 2009.

[226] Joël Janin, Kim Henrick, John Moult, Lynn Ten Eyck, Michael JE Sternberg, Sandor Vajda, Ilya Vakser, and Shoshana J Wodak. Capri : a critical assessment of predicted interactions. *Proteins : Structure, Function, and Bioinformatics*, 52(1) :2–9, 2003.

[227] Feng Gao, Mattia Bonsignori, Hua-Xin Liao, Amit Kumar, Shi-Mao Xia, Xiaozhi Lu, Fangping Cai, Kwan-Ki Hwang, Hongshuo Song, Tongqing Zhou, et al. Cooperation of b cell lineages in induction of hiv-1-broadly neutralizing antibodies. *Cell*, 158(3) :481–491, 2014.

[228] Vered Domankevich, Yarden Opatowsky, Assaf Malik, Abraham B Korol, Zeev Frenkel, Irena Manov, Aaron Avivi, and Imad Shams. Adaptive patterns in the p53 protein sequence of the hypoxia-and cancer-tolerant blind mole rat spalax. *BMC evolutionary biology*, 16(1) :177, 2016.

[229] Michael Gribskov, Andrew D McLachlan, and David Eisenberg. Profile analysis : detection of distantly related proteins. *Proceedings of the National Academy of Sciences*, 84(13) :4355–4358, 1987.

[230] Peng Xiong, Chengxin Zhang, Wei Zheng, and Yang Zhang. Bindprofx : Assessing mutation-induced binding affinity change by protein interface profiles with pseudo-counts. *Journal of molecular biology*, 429(3) :426–434, 2017.

[231] Frédéric Cazals, Flavien Proust, Ranjit P Bahadur, and Joël Janin. Revisiting the voronoi description of protein–protein interfaces. *Protein Science*, 15(9) :2082–2092, 2006.

[232] Sébastien Loriot and Frédéric Cazals. Modeling macro–molecular interfaces with intervor. *Bioinformatics*, 26(7) :964–965, 2010.

[233] Benjamin Bouvier, Raik Grünberg, Michael Nilges, and Frédéric Cazals. Shelling the voronoi interface of protein–protein complexes reveals patterns of residue conservation, dynamics, and composition. *Proteins : structure, function, and bioinformatics*, 76(3) :677–692, 2009.

[234] Frederic Cazals, Harshad Kanhere, and Sébastien Loriot. Computing the volume of a union of balls : a certified algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 38(1) :3, 2011.

[235] David Eisenberg, Morgan Wesson, and Mason Yamashita. Interpretation of protein folding and binding with atomic solvation parameters. *Chem. Scr. A*, 29 :217–221, 1989.

[236] Panagiotis L Kastritis, João PGLM Rodrigues, Gert E Folkers, Rolf Boelens, and Alexandre MJJ Bonvin. Proteins feel more than they see : fine-tuning of binding affinity by properties of the non-interacting surface. *Journal of molecular biology*, 426(14) :2632–2652, 2014.

[237] Panagiotis L Kastritis, Iain H Moal, Howook Hwang, Zhiping Weng, Paul A Bates, Alexandre MJJ Bonvin, and Joël Janin. A structure-based benchmark for protein–protein binding affinity. *Protein Science*, 20(3) :482–491, 2011.

[238] Glenn M Torrie and John P Valleau. Nonphysical sampling distributions in monte carlo free-energy estimation : Umbrella sampling. *Journal of Computational Physics*, 23(2) :187–199, 1977.

[239] Christian Bartels and Martin Karplus. Probability distributions for complex systems : adaptive umbrella sampling of the potential energy. *The Journal of Physical Chemistry B*, 102(5) :865–880, 1998.

[240] Johannes Kästner. Umbrella sampling. *Wiley Interdisciplinary Reviews : Computational Molecular Science*, 1(6) :932–942, 2011.

[241] Peter Kollman. Free energy calculations : applications to chemical and biochemical phenomena. *Chemical reviews*, 93(7) :2395–2417, 1993.

[242] Devleena Shivakumar, Joshua Williams, Yujie Wu, Wolfgang Damm, John Shelley, and Woody Sherman. Prediction of absolute solvation free energies using molecular dynamics free energy perturbation and the opls force field. *Journal of chemical theory and computation*, 6(5) :1509–1519, 2010.

[243] Alexander Benedix, Caroline M Becker, Bert L de Groot, Amedeo Caflisch, and Rainer A Böckmann. Predicting free energy changes using structural ensembles. *Nature methods*, 6(1) :3–4, 2009.

[244] A R Leach and A P Lemon. Exploring the conformational space of protein side chains using dead-end elimination and the A* algorithm. *Proteins*, 33(2) :227–39, November 1998.

[245] R F Goldstein. Efficient rotamer elimination applied to protein side-chains and related spin glasses. *Biophysical journal*, 66(5) :1335–40, May 1994.

[246] Ivelin Georgiev and Bruce R Donald. Dead-end elimination with backbone flexibility. *Bioinformatics*, 23(13) :i185–i194, 2007.

[247] Adegoke A Ojewole, Jonathan D Jou, Vance G Fowler, and Bruce R Donald. Bbkˆ*(branch and bound over kˆ*) : A provable and efficient ensemble-based algorithm to optimize stability and binding affinity over large sequence spaces. In *International Conference on Research in Computational Molecular Biology*, pages 157–172. Springer, 2017.

[248] Pablo Gainza, Kyle E Roberts, Ivelin Georgiev, Ryan H Lilien, Daniel A Keedy, Cheng-Yu Chen, Faisal Reza, Amy C Anderson, David C Richardson,

Jane S Richardson, et al. Osprey : protein design with ensembles, flexibility, and provable algorithms. *Methods in enzymology*, 523 :87, 2013.

[249] J. Larrosa. Boosting search with variable elimination. In *Principles and Practice of Constraint Programming - CP 2000*, volume 1894 of *LNCS*, pages 291–305, Singapore, September 2000.

[250] Robert Clay Prim. Shortest connection networks and some generalizations. *Bell Labs Technical Journal*, 36(6) :1389–1401, 1957.

[251] Ronald L Graham and Pavol Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1) :43–57, 1985.

[252] Robert Endre Tarjan. *Data structures and network algorithms*. SIAM, 1983.

[253] DA Case, V Babin, Josh Berryman, RM Betz, Q Cai, DS Cerutti, TE Cheatham Iii, TA Darden, RE Duke, H Gohlke, et al. Amber 14, 2014.

[254] M. Sanchez, D. Allouche, S. de Givry, and T. Schiex. Russian doll search with tree decomposition. In *Proc. IJCAI'09*, pages 603–608, San Diego (CA), USA, 2009.

[255] William H Landschulz, Peter F Johnson, and Steven L McKnight. The leucine zipper : a hypothetical structure common to a new class of dna binding proteins. *Science*, 240(4860) :1759–1765, 1988.

[256] David C Chan, Deborah Fass, James M Berger, and Peter S Kim. Core structure of gp41 from the hiv envelope glycoprotein. *Cell*, 89(2) :263–273, 1997.

[257] Christopher W Wood, Marc Bruning, Amaurys Á Ibarra, Gail J Bartlett, Andrew R Thomson, Richard B Sessions, R Leo Brady, and Derek N Woolfson. Ccbuilder : an interactive web-based tool for building, designing and assessing coiled-coil protein assemblies. *Bioinformatics*, 30(21) :3029–3035, 2014.

[258] Sherlyn Jemimah, K Yugandhar, and M Michael Gromiha. Proximate : a database of mutant protein–protein complex thermodynamics and kinetics. *Bioinformatics*, page btx312, 2017.

[259] Sidhartha Chaudhury, Sergey Lyskov, and Jeffrey J Gray. Pyrosetta : a script-based interface for implementing molecular modeling algorithms using rosetta. *Bioinformatics*, 26(5) :689–691, 2010.

[260] Hahnbeom Park, Philip Bradley, Per Greisen, 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*, 12(12) :6201–6212, 2016. PMID : 27766851.

[261] Wuyuan Lu, Izydor Apostol, MA Qasim, Nicholas Warne, Richard Wynn, Wen Lei Zhang, Stephen Anderson, Yi Wen Chiang, Eleanor Ogin, Irvin Rothberg, et al. Binding of amino acid side-chains to s 1 cavities of serine proteinases. *Journal of molecular biology*, 266(2) :441–461, 1997.

[262] Elizabeth H Kellogg, Andrew Leaver-Fay, and David Baker. Role of conformational sampling in computing mutation-induced changes in protein structure and stability. *Proteins : Structure, Function, and Bioinformatics*, 79(3) :830–838, 2011.

[263] Daniel Far Dourado and Samuel Coulbourn Flores. A multiscale approach to predicting affinity changes in protein–protein interfaces. *Proteins : Structure, Function, and Bioinformatics*, 82(10) :2681–2690, 2014.

[264] Daniel Far Dourado and Samuel Coulbourn Flores. Modeling and fitting protein-protein complexes to predict change of binding energy. *Scientific reports*, 6, 2016.

[265] Minghui Li, Franco L Simonetti, Alexander Goncearenco, and Anna R Panchenko. Mutabind estimates and interprets the effects of sequence variants on protein–protein interactions. *Nucleic acids research*, page gkw374, 2016.

Résumé

Cette thèse porte sur deux sujets intrinsèquement liés : le calcul de la constante de normalisation d'un champ de Markov et l'estimation de l'affinité de liaison d'un complexe de protéines. Premièrement, afin d'aborder ce problème de comptage #P complet, nous avons développé Z_ε^*, basé sur un élagage des quantités de potentiels négligeables. Il s'est montré plus performant que des méthodes de l'état de l'art sur des instances issues d'interaction protéine-protéine. Par la suite, nous avons développé `#HBFS`, un algorithme avec une garantie *anytime*, qui s'est révélé plus performant que son prédécesseur. Enfin, nous avons développé `BTDZ`, un algorithme exact basé sur une décomposition arborescente qui a fait ses preuves sur des instances issues d'interaction intermoléculaire appelées "superhélices". Ces algorithmes s'appuient sur des méthodes issues des modèles graphiques : cohérences locales, élimination de variable et décompositions arborescentes. A l'aide de méthodes d'optimisation existantes, de Z_ε^* et des fonctions d'énergie de Rosetta, nous avons développé un logiciel open source estimant la constante d'affinité d'un complexe protéine-protéine sur une librairie de mutants. Nous avons analysé nos estimations sur un jeu de données de complexes de protéines et nous les avons confronté à deux approches de l'état de l'art. Il en est ressorti que notre outil était qualitativement meilleur que ces méthodes.

Abstract

This thesis is focused on two intrinsically related subjects : the computation of the normalizing constant of a Markov random field and the estimation of the binding affinity of protein-protein interactions. First, to tackle this #P-complete counting problem, we developed Z_ε^*, based on the pruning of negligible potential quantities. It has been shown to be more efficient than various state-of-the-art methods on instances derived from protein-protein interaction models. Then, we developed `#HBFS`, an anytime guaranteed counting algorithm which proved to be even better than its predecessor. Finally, we developed `BTDZ`, an exact algorithm based on tree decomposition. `BTDZ` has already proven its efficiency on intances from coiled coil protein interactions. These algorithms all rely on methods stemming from graphical models : local consistencies, variable elimination and tree decomposition. With the help of existing optimization algorithms, Z_ε^* and Rosetta energy functions, we developed a package that estimates the binding affinity of a set of mutants in a protein-protein interaction. We statistically analyzed our estimation on a database of binding affinities and confronted it with state-of-the-art methods. It appears that our software is qualitatively better than these methods.