Arrangements of circles on a sphere : Algorithms and applications to molecular models represented by a union of balls

Sebastien Loriot

Arrangements de Cercles sur une Sphère: Algorithmes et Applications aux Modèles Moléculaires Representés par une Union de Boules

THÈSE

présentée et soutenue publiquement le 02 Décembre 2008

pour l'obtention du

Doctorat de l'Université de Bourgogne

(spécialité mathématiques)

par

Sébastien Loriot

Composition du jury

Rapporteurs :	Richard Lavery	Université de Lyon 7 Institut de Biologie et Chimie des Protéines
	Jack Snoeyink	The University of North Carolina at Chapel Hill Department of Computer Science
Directeurs de thèse :	Frédéric Cazals	INRIA Sophia-Antipolis - Méditerranée Algorithme-Biologie-Structure
	Frédéric Chazal	INRIA Saclay - Ile-de-France Geometrica
Examinateurs :	John Maddocks	Ecole Polytechnique Fédérale de Lausanne Laboratory for Computation and Visualization in Mathematics and Mechanics
	Robert Roussarie	Université de Bourgogne Institut de Mathématiques de Bourgogne

Table des matières

Remerciements

Je remercie mes deux directeurs de thèse pour le temps et l'énergie qu'ils m'ont consacré. Je tiens à exprimer ma gratitude aux membres du jury pour avoir relu avec attention mon manuscrit. Je veux également mentionner l'importance des membres, actuels et anciens, de chacune des équipes au sein desquelles j'ai eu le plaisir d'évoluer : l'IMB, Geometrica et ABS à l'INRIA Sophia-Antipolis, ainsi que le laboratoire de Michael Levitt à Stanford. Enfin, je tiens à remercier plus particulièrement certaines personnes : Julie et Frédéric pour leur aide précieuse aussi bien scientifique que pour le travail de rédaction de ce manuscrit (surtout Frédéric qui a beaucoup souffert pendant trois ans de mes *chinoiseries verbales*); Frédéric pour m'avoir encouragé à suivre le master MIGS grâce à quoi tout a commencé; Mes parents pour leur soutien tout au long de mes études; Gaëlle pour avoir accepté de devenir ma femme malgré mes absences.

Chapitre 1

Introduction

Dans chaque organisme vivant, l'acide désoxyribonucléique (ADN) est le support de l'information codant son développement. La structure en double hélice de nucléotides de l'ADN a été proposée par James D. Watson et Francis H. C. Crick en 1953 [192], d'après les résultats expérimentaux de Rosalind Franklin[1]. Parmi les macromolécules responsables du fonctionnement de chaque organisme vivant, les protéines sont codées par l'ADN. La première structure protéique déterminée [2] en 1958 par John Kendrew et al. [116] fut celle de la myoglobine. En décrivant la *forme* de cette protéine, ils observèrent : *Perhaps the most remarkable features of the molecule are its complexity and its lack of symmetry. The arrangement seems to be almost totally lacking in the kind of regularities which one instinctively anticipates, and it is more complicated than has been predicated by any theory of protein structure. Though the detailed principles of construction do not yet emerge, we may hope that they will do so at a later stage of the analysis.* De nos jours, malgré d'importants progrès des techniques en biologie structurale [3], le comportement des protéines garde toujours une part de mystère. Nous espérons que le travail présenté contribuera à l'enrichissement des connaissances liées aux biomolécules et à leurs fonctions.

1.1 Introduction à la Structure des Protéines

Une protéine est décrite selon quatre niveaux hiérarchiques : la structure primaire, la structure secondaire, la structure tertiaire et la structure quaternaire.

1.1.1 Une Structure Primaire : Construction à Base de Vingt Acides Aminés

Une protéine est composée d'une ou plusieurs *chaînes polypeptidiques*. Comme observé par Frederick Sanger [169] qui, en 1955 détermina la première séquence protéique (celle de l'insuline), chaque chaîne d'une protéine est une succession d'acides aminés, dont les types physico-chimiques sont différents (comme illustré par la figure 1.2). Chaque acide aminé est composé d'une partie commune (NH_2-C_αH-COOH), ainsi qu'une partie spécifique, appelée la *chaîne latérale*, qui est liée au carbone α (C_α). La liaison covalente entre l'oxygène d'un premier acide aminé et l'azote du suivant (appelée *liaison peptidique* entre les deux acides aminés) résulte d'une réaction d'hydrolyse.

[1] Cette dernière disparue en 1958, ne put être nominée pour le prix Nobel de médecine de 1962, qui fut attribué à James Watson, Francis Crick et Maurice Wilkins *for their discoveries concerning the molecular structure of nucleic acids and its significance for information transfer in living material.*

[2] Le prix Nobel de chimie de 1962 a été attribué à Max F. Perutz et John C. Kendrew *for their studies of the structures of globular proteins.*

[3] Le prix Nobel de chimie de 2002 a été attribué à Kurt Wüthrich *for his development of nuclear magnetic resonance spectroscopy for determining the three-dimensional structure of biological macromolecules in solution.* Le prix Nobel de chimie de 1982 a été attribué à Aaron Klug, *for his development of crystallographic electron microscopy and his structural elucidation of biologically important nucleic acid-protein complexes*

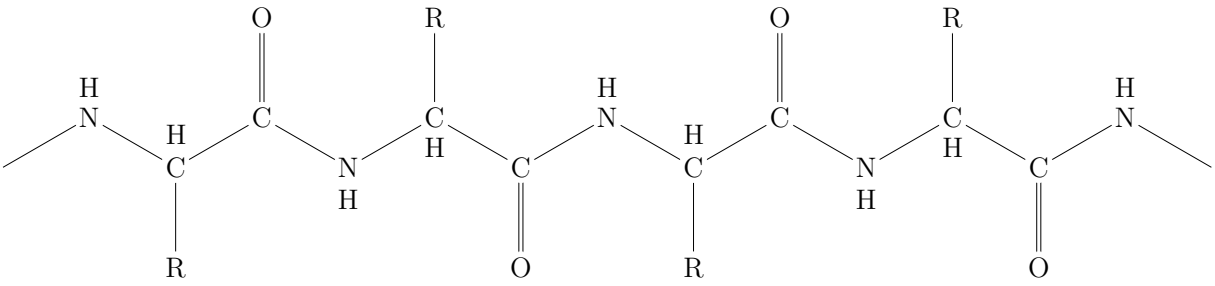


FIG. 1.1 – Une partie du squelette d'une chaîne polypeptidique d'une protéine. Les chaînes latérales sont simplement représentées par R.

Le *squelette* de chaque chaîne, est formé de la répétition du motif NH-C_αH-CO (illustré sur la figure 1.1). Par convention, chaque chaîne est lue depuis son premier azote (N) jusqu'à son oxygène terminal (O). Un acide aminé dans une protéine est appelé un *résidu*.

Les atomes du squelette du C_α d'un résidu au C_α du suivant, définissent un *groupe peptidique*. Une propriété remarquable d'un tel groupe est la coplanarité de ses atomes. De plus, au sein d'un groupe peptidique, la longueur des liaisons et les angles de liaisons sont similaires dans toutes les protéines.

Si l'on considère le C_α commun à deux groupes peptidiques, l'angle de rotation autour de la liaison N-C_α, ainsi que celui autour de la liaison C_α-C définissent les *angles de torsion* ϕ et ψ respectivement[4]. Ces deux angles de torsion sont les deux degrés de liberté, au niveau des résidus, qui définissent la structure spatiale du squelette de chaque chaîne d'une protéine (la flexibilité au niveau des chaînes latérales est représentée à l'aide d'angles de torsion préfixés χ).

1.1.2 La Structure Secondaire : Deux Motifs Principaux de Stabilité

Dans une protéine, les résidus d'une même chaîne sont souvent organisés à l'aide de motifs dits de structure secondaires, auxquels sont associés une suite périodique d'angles ϕ and ψ. Les deux motifs les plus connus (définis par Linus Pauling et al. en 1951 [155, 156]) sont le feuillet β et l'hélice α.

Une hélice α dans une protéine est constituée d'un ensemble de résidus adjacents dont le squelette forme une hélice régulière, parcourant 3,6 résidus par tour. Cette structure est stabilisée par des liaisons hydrogènes entre l'atome d'oxygène du résidu n, et l'atome d'azote du résidu $n+4$ (la longueur de cette liaison est d'environ 2,8 Å). Les chaînes latérales sont bien entendu positionnées à l'extérieur de l'hélice (comme illustré par la figure 1.3). On notera qu'une hélice α dans une protéine peut être droite ou gauche, en fonction du sens de rotation autour de son axe (la plus observée est l'hélice droite).

Un feuillet β se compose de plusieurs parties du squelette d'une chaîne (une telle partie étant appelé un *segment*). Deux segments adjacents dans l'espace sont liés entre eux par des liaisons hydrogènes entre leurs atomes d'azote et d'oxygène, comme représenté sur les figures 1.4 et 1.5. Un feuillet β peut être qualifié de parallèle (figure 1.4) ou d'antiparallèle (figure 1.5) en fonction de l'orientation de ses segments : un feuillet β est dit parallèle si deux segments voisins sont parcourus dans le même sens, sinon il est dit antiparallèle.

L'organisation de chaque chaîne polypeptidique repose principalement sur les hélices α et les feuillets β, mais il existe des structures plus compliquées. Toutes ces structures posent des contraintes sur les angles de torsion (ϕ, ψ). Pour chaque C_α, en représentant les valeurs de ϕ et ψ comme un point, on observe des régions dédiées pour les résidus impliqués dans un feuillet β ou dans une hélice α. La figure 1.6 présente une telle représentation, dite de Ramachandran [162].

1.1.3 La Structure Tertiaire : un Niveau d'Organisation Global

Lorsque chaque chaîne polypeptidique a adopté une organisation locale en motifs de structure secondaire, la chaîne, de manière globale, se replie pour former la structure tertiaire. Cette dernière est

[4] dans le cas où le C_α appartient à une proline, ϕ est constant

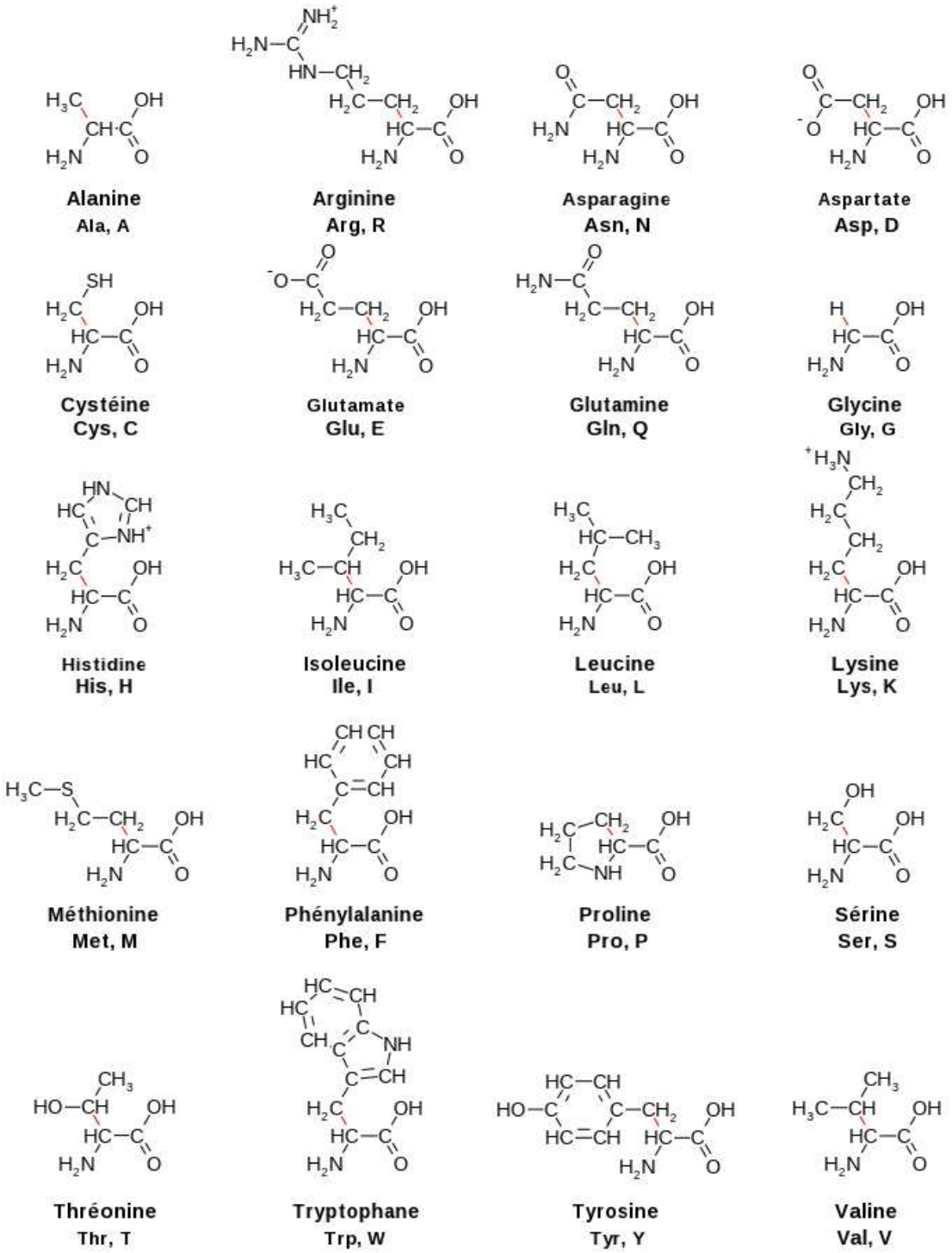


FIG. 1.2 – Liste des principaux acides aminés utilisés pour construire les protéines. Un code d'une lettre ainsi qu'un code de trois lettres sont associés à chaque acide aminé.

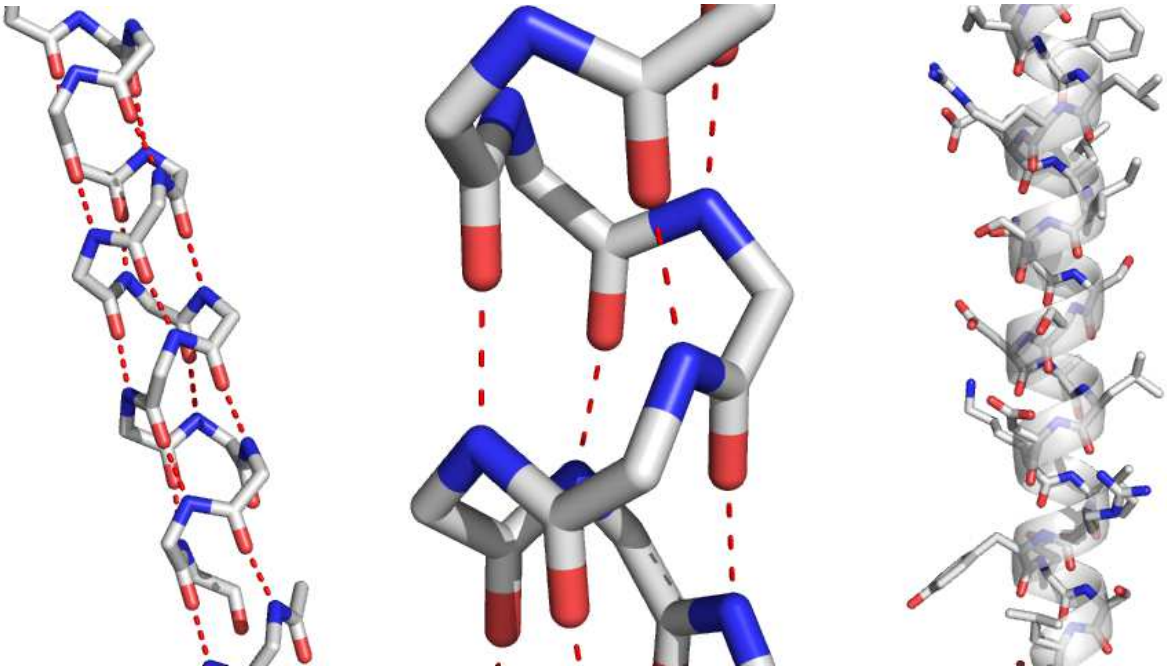


FIG. 1.3 – Hélices α dans les protéines. De gauche à droite: une hélice α parfaite, les liaisons hydrogènes sont représentées par des tirets et les chaînes latérales sont omises; un agrandissement de l'image de gauche; une hélice α extraite d'une protéine avec ses chaînes latérales. Les atomes de carbones, azotes et oxygènes sont colorés en blanc, bleu et rouge respectivement. Les atomes d'hydrogène ne sont pas représentés.

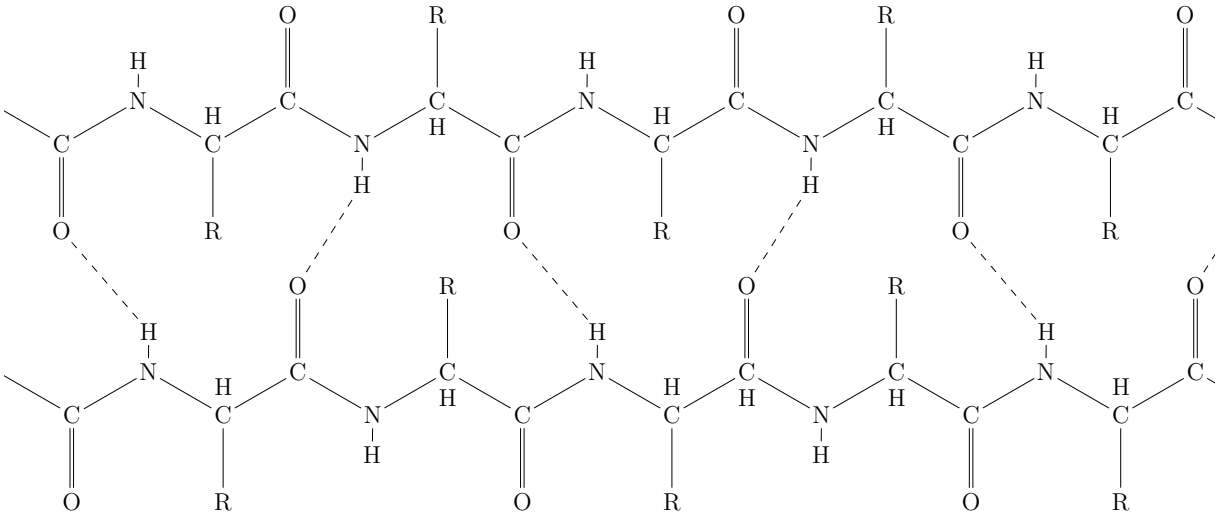


FIG. 1.4 – Un feuillet *beta* parallèle. Les chaînes latérales sont remplacées par un simple R. Les liaisons hydrogènes sont représentées par des tirets.

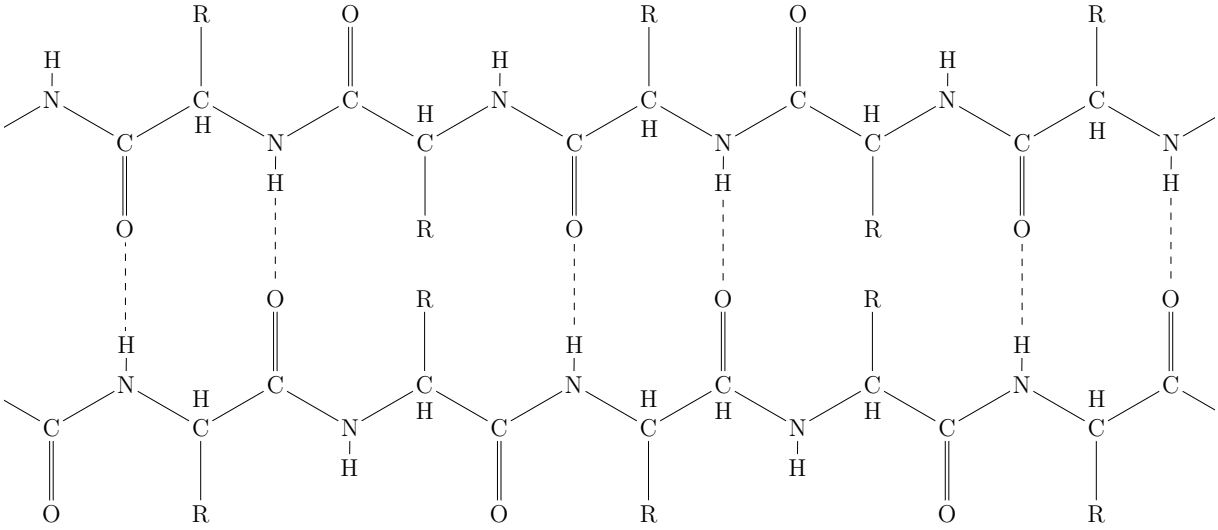


FIG. 1.5 – Un feuillet *beta* antiparallèle. Les chaînes latérales sont remplacées par un simple R. Les liaisons hydrogènes sont représentées par des tirets.

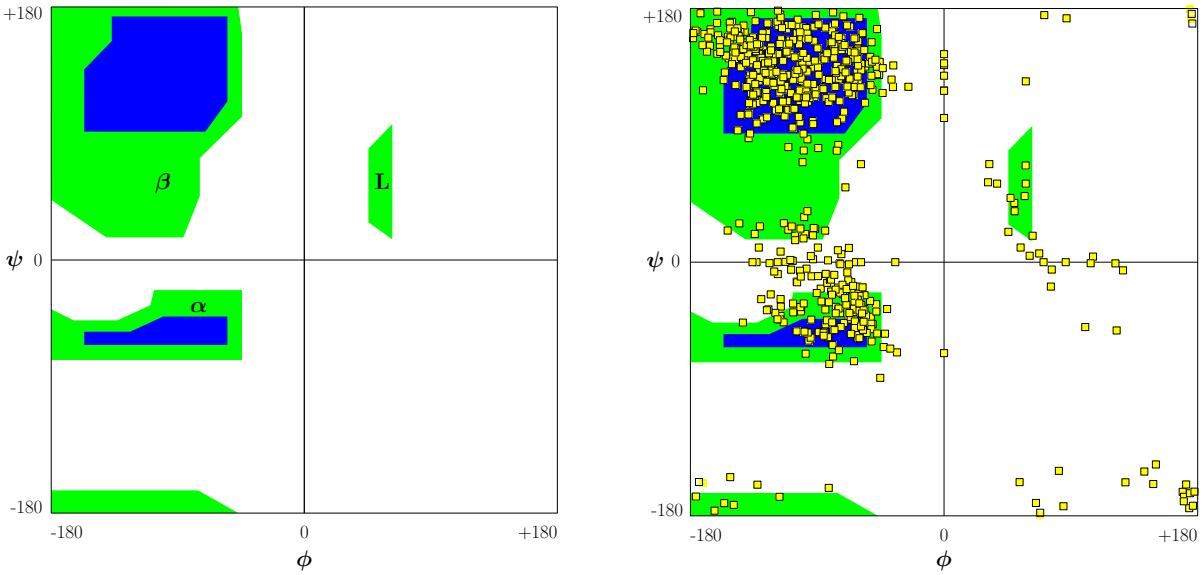


FIG. 1.6 – Représentations de Ramachandran [162]. Gauche : Les régions les plus probables et limites pour les angles (ϕ, ψ) sont colorées respectivement en vert et en bleu. Les régions correspondant aux valeurs de (ϕ, ψ) pour une hélice droite, un feuillet β et une hélice gauche sont approximativement les régions nommées $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ et $\mathbf{L}$ respectivement. Droite : Valeurs de (ϕ, ψ) pour l'allantoicase de la levure (code PDB `1SG3`).

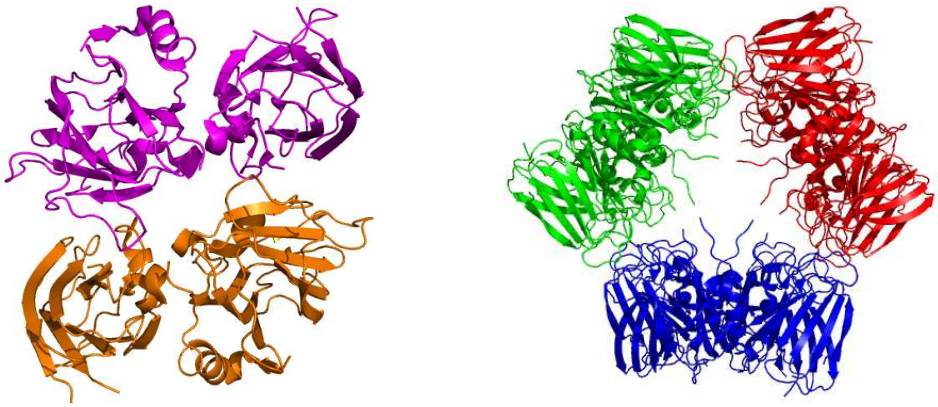


FIG. 1.7 – Symétries dans les protéines. Gauche : l'unité asymétrique du cristal de l'allantoicase de la levure (PDB code 1SG3) est un dimère (une couleur par sous-unité). Droite : l'unité biologique est un trimère des dimères (une couleur par dimère). La transformation D_3 est utilisée pour reconstruire l'unité biologique depuis le protomère.

essentielle pour la fonctionnalité des protéines. En 1957, une expérience a montré que le processus de repliement d'une protéine peut être inversé, et est conduit principalement par des réactions chimiques [175]. Les ribonucléases de pancréas de boeuf (des enzymes digestives), cessent leurs activités lorsqu'elles sont dépliées (état obtenu en augmentant délicatement la température). Quand ces protéines retrouvent leurs configurations repliées, l'activité digestive est de nouveau observée. *The lack of regularity* de la structure tertiaire d'une protéine est très importante pour la diversité de ses interactions, et donc pour ses fonctions. On peut observer la présence de *domaines* au sein des structures tertiaires. Ces domaines relativement réguliers sont composés d'enchaînement de motifs de structure secondaire (α-domaine, β-domaine, $(\alpha+\beta)$-domaine et (α/β)-domaine), et ont été observé avec précision dans les protéines [130]. Le livre de Brändén et Tooze fournit plus de détails sur ce sujet [33]. Kendrew et al., après être parvenus à déterminer la structure haute résolution de la myoglobine [117] en 1960, ont observé que les chaînes latérales hydrophobiques sont plus dans le coeur des protéines, alors que les hydrophiliques sont plutôt sur le bord [118]. Cette propriété, avec la perte d'entropie due au repliement, souvent nommée l'effet hydrophobique, est considérée comme un paramètre clé du repliement des protéines [74].

1.1.4 La Structure Quaternaire : un Assemblage Fonctionnel

L'assemblage de plusieurs chaînes polypeptidiques par des liaisons majoritairement non covalentes définit le dernier niveau de la structure d'une protéine. Chaque chaîne d'un tel assemblage est appelée une sous-unité. Une protéine composée de plus d'une sous-unité est appelée un *oligomère*.

Les sous-unités d'une protéine peuvent être toutes identiques ou différentes. Un *protomère* d'une protéine est le plus petit sous-ensemble de sous-unités différentes nécessaires pour construire cette protéine (si toutes les sous-unités sont différentes, le protomère est alors la protéine elle-même). La figure 1.7 présente l'exemple de l'allantoicase de la levure qui est un hexamère.

Les sous-unités sont généralement organisées de manière symétriques, utilisant exclusivement des rotations axiales. Une transformation C_n est utilisée pour arranger n copies du protomère, en utilisant des rotations successives d'angle $\frac{2\pi}{n}$ autour d'un axe donné. Une transformation D_n est utilisée pour arranger $2n$ copies du protomère, en utilisant une rotation axiale d'angle π, suivie par des rotations successives d'angle $\frac{2\pi}{n}$ autour d'un axe perpendiculaire au premier. Des transformations d'ordre supérieures sont parfois observées et utilisent des rotations autour des axes de symétrie d'un polyèdre régulier donné.

Plusieurs raisons ont été avancées pour expliquer l'existence de la structure quaternaire des protéines, elles sont toutes liées à la taille (en nombre d'acides aminés) de la protéine [190].

Premièrement, le processus de repliement semble plus simple (en s'appuyant le principe *diviser pour régner*). Deuxièmement, pour une question d'entretien d'un tel assemblage, il est plus efficace de remplacer

une partie défectueuse plutôt que l'ensemble. La dernière raison que nous citerons ici, est le comportement symétrique que permet un tel assemblage.

1.1.5 Techniques Expérimentales pour la Détermination des Structures Protéiques

Cristallographie aux rayons X. Le principe de cette méthode est d'utiliser la diffraction d'un faisceau de rayons X induite par un cristal de protéines pour obtenir une carte d'électron-densité.

La première limitation de cette approche vient de la constitution du cristal. Ce dernier doit être suffisamment large, sans défauts et *pur*. Le volume du cristal est composé à 50 pour cent de molécules d'eau, pour satisfaire les propriétés physico-chimiques et palier la forme globulaire des protéines. La cristallisation est fonction de plusieurs paramètres (concentration en sel, température, pression ...), et plusieurs semaines peuvent être nécessaires pour obtenir un cristal assez gros. Un fois le cristal obtenu, il est souvent cryogénisé pour limiter les déplacements au sein du cristal mais aussi pour améliorer la résistance du cristal aux rayons X. Les rayons X les mieux adaptés sont produits généralement à l'aide d'un synchrotron. La qualité d'un cristal (et donc de la carte électron-densité) est déterminée par la qualité de la diffraction. La position de chaque atome est estimée d'après la carte électron-densité, de sorte que la structure finale corresponde aux densités. Une structure est considérée comme étant à haute résolution quand la position de ses atomes est correcte à moins de 2Å. La structure déterminée expérimentalement n'est pas (en général) l'entité biologique de la protéine, mais l'*unité asymétrique du cristal*, qui représente la plus petite entité répétée pour construire le cristal.

Une telle méthode permet de déterminer la structure d'une protéine d'au plus 500 acides aminés.

Spectroscopie par résonance magnétique nucléaire (RMN). L'idée de cette méthode est d'utiliser le moment magnétique de certains noyaux atomiques (le *spin*). Une protéine est plongée dans un champ magnétique constant, puis soumise à un second champ que l'on fait ensuite osciller par étapes. Ce second champ induit un changement du spin des noyaux, et la désactivation de ce champ entraîne l'émission de radiations mesurables par les noyaux atomiques retrouvant leurs positions initiales qui sont mesurables. La fréquence de ces radiations dépend à la fois du type d'atome et de l'environnement atomique de chaque noyau. La structure d'une protéine est obtenue principalement par l'étude des radiations émises par le changement de spin de ses atomes d'hydrogène. De la variation des paramètres du second champ magnétique, on peut estimer (i) la distance entre les atomes d'hydrogène liés au même atome ou à deux atomes liés de manière covalente, et (ii) la distance entre deux atomes d'hydrogène qui sont proches dans l'espace, mais pas du type précédemment décrit.

La conclusion de l'étude des résultats de cette méthode est une liste de contraintes de distances. Ceci implique que plusieurs structures peuvent satisfaire à ces contraintes, et donc que plusieurs modèles sont proposés.

Un des avantages de cette méthode vient du fait que les protéines sont en solution. La composition de la solution peut donc être ajustée (de même que la température, la pression ou le pH) de manière à observer le comportement d'une protéine dans différents environnements (physiologiques ou non). Néanmoins, selon l'ajustement de ces paramètres, mais aussi en fonction de la protéine étudiée, l'interprétation du signal peut être difficile voire impossible. De plus, pour les besoins de certaines expériences, il est parfois nécessaire d'utiliser des isotopes radioactifs des atomes de carbone, d'azote et d'hydrogène.

Cette méthode permet de déterminer la structure d'une protéine d'au plus 250 acides aminés.

La cryo-microscopie électronique (Cryo-EM). Cette méthode se base sur l'observation directe des protéines en utilisant un microscope très puissant. Étant donné une molécule dans son environnement naturel, l'échantillon est souvent cryogénisé en utilisant de l'azote liquide (ou de l'éthane), puis observé à l'aide d'un microscope électronique. La différence principale avec un microscope classique est que l'échantillon n'est pas *illuminé* par de la lumière, mais bombardé par des électrons (la plus faible longueur d'onde des électrons permet un meilleur grossissement). Le faisceau quant à lui est dirigé par une lentille électrostatique ou électromagnetique (un anneau). La sortie obtenue est une carte d'electron-densité déduite des collisions élastiques et inélastiques (produisant du bruit) avec les atomes de l'échantillon. Lorsque cela est possible (le faisceau d'électrons endommage l'échantillon), plusieurs images à différentes positions autour de l'échantillon peuvent être utilisées pour faciliter l'étape de reconstruction du modèle. Trois principales approches sont utilisées pour reconstruire la structure d'une molécule, de la plus haute à la

plus faible résolution : (i) En *cristallographie électronique*, l'échantillon est organisé en un cristal selon deux dimensions (d'une taille strictement inférieure à celle requise pour la cristallographie aux rayons X). L'étape de reconstruction utilise, comme pour la cristallographie aux rayons X, le principe de diffraction et la symétrie du réseau cristallin. La résolution maximum est d'environ 5Å. (ii) En *reconstruction par particule isolée*, l'échantillon contient plusieurs copies de la même molécule. La reconstruction utilise le signal des différentes copies, moyennant les différentes structures. La résolution maximum est d'environ 10Å. (iii) En *tomographie électronique*, l'échantillon utilisé contient une seule entité de la molécule considérée. L'utilisation du signal de différentes copies pour améliorer le rapport signal sur bruit est donc impossible dans ce cas. Le principe de cette approche est d'augmenter le nombre de vues, en faisant tourner l'échantillon autour d'un axe perpendiculaire au faisceau d'électrons. La résolution maximum est d'environ 20Å.

Cette méthode permet de déterminer des assemblages de protéines de grande taille, c'est à dire d'au moins 300 acides aminés.

1.1.6 Bases de Données de Structures Protéiques

La première base de données disponible a été créée en 1971 par le *Brookhaven National Laboratory* (USA), et portait le nom de **Protein Data Bank**. En 1999, cette base de données a été transférée aux membres du *Research Collaboratory for Structural Bioinformatics*[5] (RCSB). En 2003, RCSB avec le *Macromolecular Structure Database at the EMBL's European Bioinformatics Institute*[6] (PDBe), et la *Protein Data Bank Japan*[7] (PDBj) ont décidé de travailler en collaboration pour donner naissance à la *Worldwide Protein Data Bank*[8] (wwPDB) [27], une base de données internationale de structures de macromolécules disponible gratuitement et librement pour toute la communauté. En 2006, le groupe *Biological Magnetic Resonance Data Bank*[9] (BRMB) a rejoint le projet. Cette structure permet de regrouper les efforts de chacun de ces groupes, et a aussi un rôle de standardisation. Chaque fichier dans la base de données est disponible dans au moins un format de fichier commun (le format PDB). La qualité de l'ensemble des fichiers de la base de données ainsi que leur consistance sont garanties (de même que la non redondance des entrées). La consistance des fichiers est particulièrement importante pour le développement de logiciels (robustesse lors la lecture des fichiers), mais aussi pour l'évolution du format : ces décisions sont prises par la communauté et non par un seul groupe. En septembre 2008, 42026, 6506 et 134 structures de protéines obtenues respectivement par cristallographie aux rayons X, RMN et cryo-EM étaient disponibles dans la base de données.

La base de données *Structural Classification of Proteins*[10] (SCOP) [148, 13] fournit une classification des protéines, en fonction de leurs structures et de leurs évolutions relatives. Toutes les structures disponibles dans la wwPDB sont considérées. Une autre base de données similaire fournit une classification hiérarchique des structures protéiques, en se basant sur quatre niveaux: *Class, Architecture, Topology and Homologous superfamily*[11] (CATH) [154, 104].

En cristallographie aux rayons X, l'unité asymétrique du cristal n'est en général pas l'unité biologique. L'unité asymétrique peut contenir plusieurs copies de l'unité biologique, ou plusieures opérations de symétries cristallographiques doivent être appliquées pour reconstruire l'unité biologique à partir de l'unité asymétrique. Même si la matrice de transformation est présente dans les fichiers de structures (pour le format PDB, il faut regarder le champ `REMARK 350`), le *Protein Quaternary Structure server*[12] (PQS) [109] fournit pour chaque entrée de la wwPDB obtenu par cristallographie aux rayons X, les coordonnées de la probable structure quaternaire.

[5] `http://www.rcsb.org/pdb`
[6] `http://www.ebi.ac.uk/msd/`
[7] `http://www.pdbj.org/`
[8] `http://www.wwpdb.org/`
[9] `http://www.bmrb.wisc.edu/`
[10] `http://scop.mrc-lmb.cam.ac.uk/scop/`
[11] `http://www.cathdb.info`
[12] `http://pqs.ebi.ac.uk`

Type d'atome	rayon en Å
Oxygène	1,40
Azote trigonal	1,65
Azote tetraèdrique	1,50
Carbone tetraèdrique	1,87
Carbone trigonal	1,76
Sulfure	1,85
Eau	1,40

TAB. 1.1 – Rayons étendus pour le modèle de VdW de protéines, tirés de [57].

1.2 Quelques Enjeux de la Biologie Structurale Computationelle pour les Protéines

Dans cette section, sont présentés des problèmes dans le domaine de la biologie structurale computationelle, un domaine qui fait appel aux mathématiques et à l'informatique appliqués à la biologie structurale. Les points particuliers qui sont évoqués ici motivent le travail présenté dans ce manuscrit mais donnent aussi de futures directions de recherche. Nous commencerons par quelques rappels de notions essentielles pour l'étude des protéines, puis nous introduirons le problème du repliement et celui de l'amarrage de protéines.

1.2.1 Pré-requis : Modèles Moléculaires en Biologie Structurale Computationelle

Un atome est composé d'un noyau entouré par un nuage d'électrons. Idéalement, la physique d'une protéine serait la mieux décrite par la mécanique quantique. Néanmoins, de part le coût des calculs [13], la description des protéines se base essentiellement sur la physique newtonienne. Les prochains paragraphes présentent des quantités géométriques permettant de manipuler les protéines dans un tel modèle.

Root mean square deviation La structure d'une protéine in silico est principalement décrite à l'aide d'un point par atome (ou un ensemble de positions par atome dans le cas de données provenant de la RMN). Une mesure très utilisée pour quantifier la différence entre deux structures de la même séquence protéique est le *root mean square deviation* (RMSD). Si on considère une protéine de n atomes comme un point de $\mathbb{R}^{3n}$, le RMSD entre deux protéines est la distance Euclidienne entre les deux protéines divisée par $\sqrt{n}$. Le RMSD est souvent calculé après une étape de minimisation entre les structures, en utilisant des transformations rigides pour amener une protéine proche de la seconde. La valeur calculée est alors nommée le *least-RMSD*. Le C_α-RMSD signifie que le calcul du RMSD est limité aux carbones α des protéines.

Union de boules Pour faciliter à la fois les simulations et la visualisation des protéines, un atome est souvent représenté par une boule, dont le rayon dépend de la nature de l'atome mais aussi de son environnement chimique : c'est le modèle de *Van der Waals* (VdW). La position des atomes d'hydrogène étant difficile à déterminer, selon les applications, ces atomes sont souvent omis. Des jeux de rayons pour les atomes *lourds* (i.e. pas d'hydrogène) sont disponibles pour tenir compte de cette seconde approximation. Une boule/sphère atomique désignera dans la suite la boule/sphère correspondant à l'atome dans le modèle de VdW. Pour plus d'informations sur les rayons utilisés, une vue d'ensemble est disponible dans [100]. La table 1.1 présente aussi un jeu de rayon tiré de cristaux de structures [57].

Un modèle gros grain est parfois utilisé pour représenter une protéine. Chaque résidu est approché par un nombre réduit de boules (dont les rayons sont plus grands [128]). Ce modèle fournit une vue

[13] on ne peut actuellement traiter que des systèmes de petite taille (environ 50 atomes)

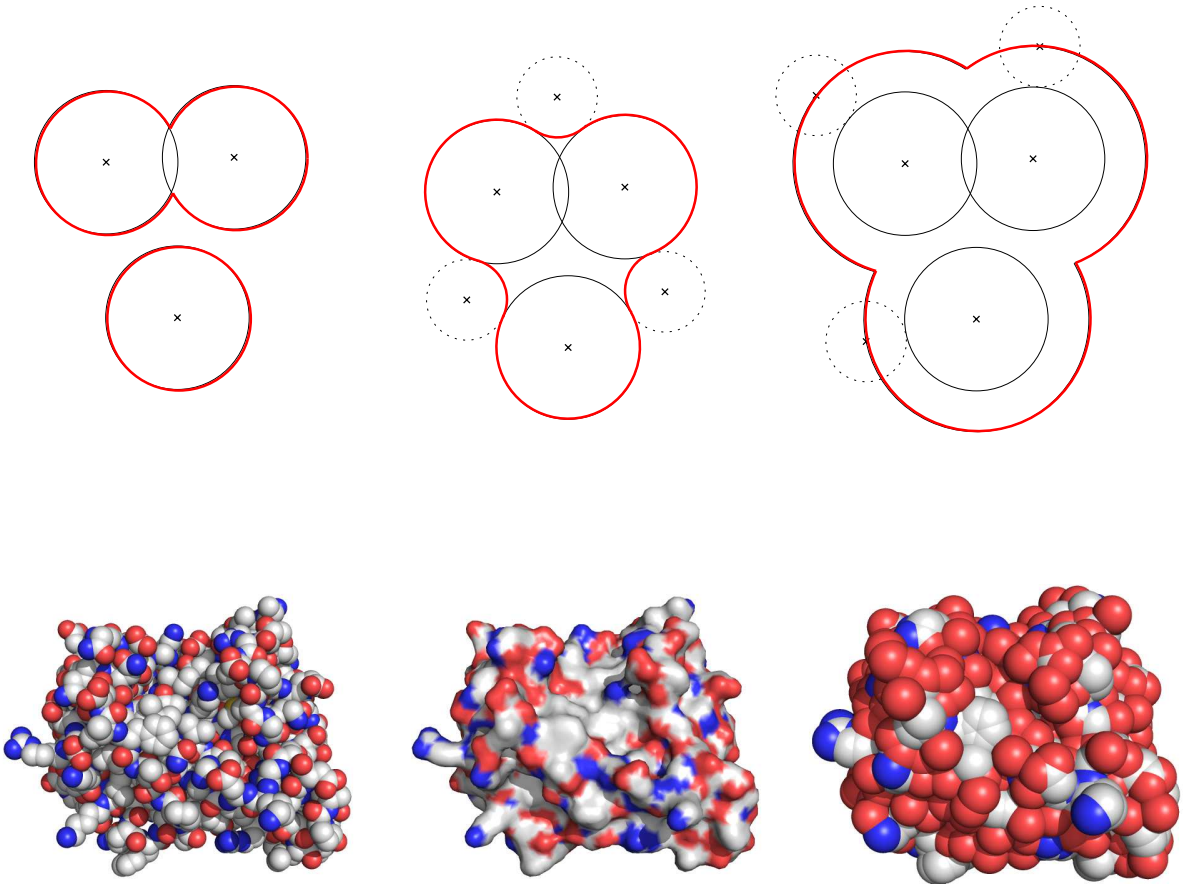


FIG. 1.8 – Première ligne : Illustrations en deux dimensions des surfaces moléculaires. Les atomes sont dessinés avec des cercles noirs, une boule de solvant par un cercle en pointillés et les différentes surfaces sont en rouge. Seconde ligne : Illustrations en trois dimensions des surfaces moléculaires. Les atomes de carbone, azote et oxygène sont colorés en blanc, bleu et rouge respectivement. De gauche à droite : la surface de Van der Waals, la surface de Connolly et la surface accessible au solvant.

d'ensemble de la structure d'une protéine (assez grossier pour ignorer certains niveaux de flexibilité au niveau des chaînes latérales), et est utile pour réduire le nombre de primitives à manipuler.

Surfaces et volumes S'appuyant sur le modèle de VdW, plusieurs surfaces (et donc volumes délimités par ces surfaces) ont été définies. La *surface de VdW* d'une molécule est définie comme le bord de l'union de ses boules. Une molécule d'eau est en général représentée par une unique boule (nommée une *boule de solvant*), qui est celle correspondant à son atome d'oxygène dans le modèle de VdW. Son rayon est généralement 1,4 Å. La *Surface Accessible au Solvant* (SAS) d'une molécule est définie comme le bord de l'union ses boules, dont les rayons ont été augmentés du rayon d'une boule de solvant. Cette surface délimite le volume dans lequel le centre d'une boule de solvant peut être placé, de sorte que l'intersection de cette boule avec au moins une des boules de la molécule est non vide. La *surface de Connolly* (souvent appelée *surface moléculaire*) d'une molécule, est définie comme le bord du volume qui n'est pas accessible à une boule de solvant, c'est à dire le volume qui ne peut pas être couvert par une boule de solvant sans pénétrer une boule de la molécule. Une manière de tracer cette surface est de considérer la surface couverte par une boule de solvant que l'on ferait rouler sur la surface de VdW d'une molécule.

Ces définitions sont illustrées sur la figure 1.8.

1.2.2 Pré-requis : Quantités Géométriques Classiques pour l'Étude des Protéines

Biophysique

Surface exposée A partir du modèle de Van der Waals, Lee et Richards [126] ont introduit la notion de surface accessible au solvant (SAS). La définition peut être restreinte soit à un atome soit à un groupe d'atomes (sous-unité, résidu, ...). Lee et Richards ont utilisé cette quantité pour étudier le problème du repliement des protéines. Le changement d'accessibilité d'un résidu entre l'état replié et déplié d'une protéine a été mesuré. La SAS dans l'état déplié est calculée à partir d'une configuration où le résidu était placé entre deux résidus d'alanine. Chothia s'est intéressé à l'hypothèse d'un effet hydrophobe qui aurait un rôle important dans le repliement des protéines [56]. Il a en effet observé une corrélation linéaire entre l'aire de la SAS ($SASA$) des acides aminés et leur énergie libre (mesurée) de transfert de l'eau dans un solvant organique. Cette relation a aussi été observée plus tard [188, 87]. En 1986, Eisenberg et McLachlan [83] ont donné une estimation de la contribution de chaque atome d'une protéine à l'énergie libre de solvatation en utilisant l'accessibilité de l'atome au solvant. Cette mesure a été utilisée pour estimer la stabilité d'une protéine dans l'eau. Vallone et al. [188] ont observé que leurs mesures expérimentales concordaient avec cette estimation. Pour une revue complète sur les surfaces moléculaires (définitions, applications et algorithmes), jusqu'en 1996, le lecteur peut consulter [63].

Surfaces de contact Les contacts géométriques au sein d'une molécule ont été étudiées à différents niveaux (entre protéines, résidus ou atomes). L'aire de la surface accessible au solvant perdue (BSA) introduite par Chothia et Janin [59] pour des complexes protéine-protéine est définie comme l'aire de la surface accessible perdue lors de la formation du complexe : $BSA(A \cup B) = SASA(A) + SASA(B) - SASA(A \cup B)$. Notons que cette formule suppose que les deux molécules sont assemblé à l'aide de transformations rigides, c'est à dire pas de déformations. Chothia et Janin ont utilisé cette quantité pour étudier la stabilité de trois complexes protéine-protéine pour modéliser la contribution hydrophobique induite par la perte d'accessibilité. La BSA a depuis été un paramètre central de nombreuses études de la structure des interfaces des complexes macromoléculaires [45, 160]. Abagyan et Totrov [1] ont introduit plusieurs mesures à différentes échelles pour évaluer la différence entre deux structures de la même protéine. Ces mesures sont basées sur une matrice d'aire de contact entre des paires de résidus. Pour calculer ces coefficients, il faut déjà calculer la $SASA$ de tous les résidus pris séparément. Ensuite étant donné un résidu i, pour chaque résidu j, il faut mesurer la perte de $SASA$ i induite par j. La matrice qui résulte de ces mesures n'est pas symétrique car la SAS perdue par i n'est pas la même que celle perdue par j. Un moyen proposé pour palier ce fait est de prendre la moyenne des valeurs. Ils ont observé que ces mesures sont plus sensibles que les classiques RMSD et cela permet aussi d'avoir une représentation graphique de la préservation des contacts. Une autre définition des surfaces de contact a été employée pour modéliser le site d'amarrage de la quinone-B [180]. La surface de contact entre un atome a et un atome b est définie comme la surface de a inclue dans b et plus *proche* de b (ce qui est revient à définir la surface de contact comme l'intersection de la surface de a avec la cellule de b dans le diagramme de puissance associé aux atomes de la protéine). Mc Conkey et al. [141] ont donné une autre méthode pour calculer ces coefficients de contact en utilisant la tessellation induite par le diagramme de puissance. Ces coefficients ont ensuite été impliqués dans la définition d'une fonction empirique de contacts non-covalents pour détecter le repliements natif de plusieurs protéines [142]. Un autre application de ces coefficients que l'on peut citer est la prédiction des configurations des chaînes latérales, en utilisant en plus un critère de complémentarité de surface [87]. L'aire de contact entre atomes (ACA) a été définie comme suit [70] : pour deux atomes a et b, l'ACA est la $SASA$ perdue lorsque l'on ajoute b aux voisins covalents de a, à laquelle on ajoute la quantité correspondante sur la surface de b. L'ACA est corrélée à l'énergie d'interaction électrostatique.

Volumes Plusieurs études impliquant des volumes *moléculaires* ont été réalisées. Comme Poupon l'écrit dans sa revue [159], le volume associé aux atomes a entre autres été utilisé de manière intensive pour caractériser la compacité d'une molécule. Par exemple, Gerstein et Chothia [99] ont observé que le coeur des protéines est plus compact que la partie en contact avec l'eau. Dans une grande partie des travaux

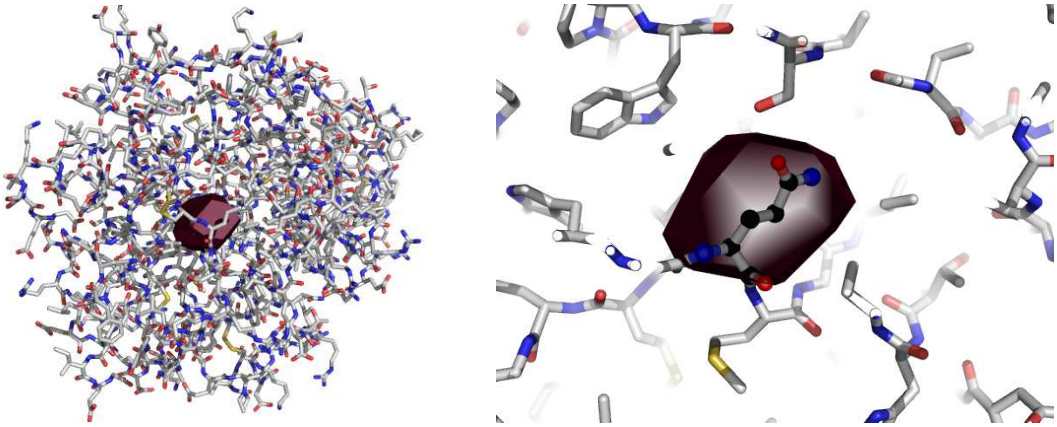


FIG. 1.9 – La cellule de Voronoï d'un résidu de glutamine (numéro 209) dans une représentation gros grain de la thrombine `E192Q-BPTI` (code PDB `1BTH`). La protéine est représentée avec des bâtons. Gauche : La cellule de Voronoï dans la protéine; Droite : Le résidu de glutamine à l'intérieur de sa cellule de Voronoï. (Images fournies par Julie Bernauer [29])

réalisés, le volume associé à un atome (ou un résidu) correspond à celui d'un polyèdre associé à l'atome. Plus précisément, deux options ont souvent été choisies : (i)le volume associé à un atome est défini comme le volume de sa cellule dans le diagramme de Voronoï [17] des centres des atomes; ou (ii)le volume associé à un atome est défini comme le volume de sa cellule dans le diagramme de puissance [16] des centres des atomes, avec comme poids les rayons de Van der Waals correspondant au carré. Un exemple d'une cellule du diagramme de Voronoï des centroïdes des chaînes latérales est présenté sur la figure 1.9.

Ces approches ont plusieurs inconvénients. En particulier, un atome sur le bord d'une modèle est associé à une cellule infinie. Pour éviter ce problème, des molécules d'eau fictives ont été ajoutées autour du modèle (soit sur une grille régulière, soit en plaçant les boules de solvant tangentes à trois atomes du bord du modèle). Une autre solution, utilisant le diagramme de puissance consiste à définir le volume associé à un atome comme le volume de sa cellule dans le diagramme, mais restreint à sa boule de VdW. Pour finir, nous indiquons que l'énergie libre de solvatation d'un composé dans l'eau a été estimée en utilisant le volume d'une union de boules et des termes d'interactions à plusieurs corps [111].

Algorithmes

Aire Nous présentons une sélection d'algorithmes, en insistant sur les aspects géométriques sous-jacents. Si un logiciel implémentant une méthode donnée est disponible, ce dernier est cité entre parenthèses. On peut séparer les méthodes en celles qui se basent sur une formule analytique, et celles qui s'appuient sur des approximations. Les méthodes basées sur des approximations utilisent une discrétisation de l'espace ambiant (une grille) ou des sphères atomiques (échantillonnées avec des points) pour fournir une estimation de l'aire. La qualité de l'estimation dépend bien entendu de la densité de l'échantillonnage.

Si on considère les méthodes basées sur une formule analytique, quelque soit le type de surface considérée (VdW ou SAS), son aire est calculée en utilisant l'aire de chaque morceau du bord de surface des boules qui contribue à la définition du bord de l'union des boules.

Nous allons décrire deux classes d'algorithme. La première classe utilise les propriétés du diagramme de puissance [16]. Dans ce paragraphe, une cellule sous entendra une cellule du diagramme de puissance des points pondérés correspondant aux atomes. Les parties de la surface d'une boule atomique qui contribuent au bord de l'union des boules sont situées à l'intérieur de sa cellule (les parties en dehors étant couvertes par d'autres boules). Une stratégie qui semble naturelle est de calculer l'intersection de chaque sphère atomique avec sa cellule. Cette étape nécessite une liste de voisins. Ces voisins peuvent être récupérés en utilisant par exemple une grille [189, 141]. En suivant la méthode non-naïve pour construire une cellule à l'aide de la polarité, les voisins redondants peuvent être filtrés [95] (GETAREA). Une autre stratégie possible est d'utiliser les propriétés du complexe dual [78]. En effet les boules qui contribuent au

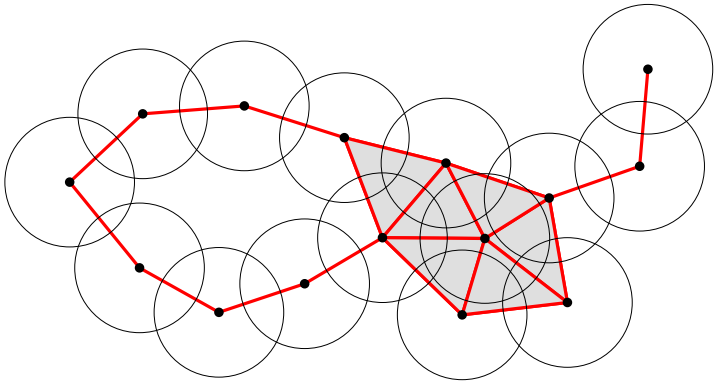


FIG. 1.10 – Le complexe dual d'une union de disques, aussi connu sous le nom de α-complexe ($\alpha = 0$). Le complexe dual est inclus dans l'union des disques, et une cavité de l'union des boules est inclue dans une cavité du complexe dual. Les disques qui contribuent au bord de l'union des disques peuvent être retrouvés à l'aide des simplexes non *intérieur* [78].

bord se lisent dans le complexe dual [7] (voir figure 1.10). Une autre alternative, utilisant une structure de données partitionnant l'espace, la contribution des boules au bord peut être reconstruite par une exploration extensive des contacts entre des paires et des triplés de boules [170] (MSMS).

Une seconde classe d'algorithmes se concentre sur les cercles d'intersection sur une sphère donnée afin de construire des cycles d'arcs de cercles qui bordent les parties convexes qui contribuent au bord de l'union des boules. Quelques exemples qui maintiennent ces boucles sont [185], [186] (SurfRace, FastSurf). Pour terminer, le bord de l'union des boules correspond aux faces exposées dans l'arrangement des cercles d'intersection sur chaque sphère [106].

Volume Plusieurs méthodes ont été développées pour calculer le volume d'une molécule entière. Connolly [62] a montré comment calculer le volume bordé par la surface de Connolly en utilisant une décomposition du volume en parties disjointes, avec une formule analytique pour chacune de ces parties. Gibson et Scheraga ont utilisé la formule d'inclusion-exclusion pour calculer le volume d'une union de boules [101]. Edelsbrunner [79] a amélioré la réduction des termes de la formule d'inclusion-exclusion, en montrant que le volume d'une union de boules d'un ensemble B est donné par :

$$\mathrm{vol}(\cup B) = \sum_{X \subseteq B} -1^{\mathrm{card}(X)-1} \mathrm{vol}(\cap X)$$

où X est défini pour chaque simplexe du complexe dual de B, comme les boules correspondants aux vertexes du simplexe. Edelsbrunner et Attali [15] ont montré plus tard que la formule reste correcte pour n'importe quel complexe simplicial abstrait dont les simplexes sont des ensembles indépendants de boules, et dont le plongement canonique a le même bord, et est dans le même espace que le complexe dual. De plus, toutes les formules d'inclusion-exclusion minimales sont encodées dans un tel complexe simplicial abstrait. En particulier, cela implique que l'on a besoin de formules pour calculer le volume d'intersection d'au plus quatre sphères. Une solution alternative qui utilise la tessellation induite par le diagramme de puissance associé à une molécule a été développé par Mc Conkey et al. [141], en utilisant les formules de volume de cônes et celles de polyèdre pour calculer le volume de la restriction de chaque boule à sa cellule.

1.2.3 Le problème du repliement

Comprendre et être capable de prédire comment chaque chaîne d'une protéine se replie pour acquérir sa structure tertiaire est un des principaux challenges en biologie structurale computationelle, connu sous le nom de problème *repliement*. George D. Rose a décrit ce problème [165] comme suit : *For a protein molecule, function follows form. Unreeled, it is just a long string of amino acids. But wad it up just the*

right way, into a bundle known as its native conformation, and it becomes the stuff of life. La structure de certaines protéines étant pour le moment impossible à obtenir expérimentalement, trouver un algorithme fiable et efficace au problème du repliement est d'autant plus important.

Le *Critical Assessment of Techniques for Protein Structure Prediction* (CASP) est une expérience mondiale qui se tient tous les deux ans depuis 1994, et qui donne la possibilité aux chercheurs de tester la qualité de leurs méthodes de prédiction sans à priori. Des structures tertiaires encore non publiées sont soumises aux participants. Ces derniers peuvent alors soumettre au serveur, un nombre limité de leurs meilleures prédictions. Les résultats sont publiés dans un journal, ce qui donne une sorte d'état de l'art des logiciels et méthodes disponibles (CASP7 [145] est la session la plus récente au moment de la rédaction de ce manuscrit).

Les méthodes utilisées peuvent être séparées en deux classes, les méthodes qui utilisent uniquement la séquence de la protéine pour déterminer sa structure (dites modélisation *de novo*), et les méthodes qui utilisent en plus de la séquence de la protéine, des bases de données de protéines repliées (dites modélisation par *homologie*). Une décomposition en deux sous-classes du dernier type de méthode est souvent observée : (i)*L'enfilage* utilise la façon dont des protéines homologues sont repliées, et dont les structures ont été déterminées expérimentalement. Se basant sur un alignement des séquences, le repliement de la protéine avec le plus de correspondances est choisie comme modèle. La structure prédite peut alors comporter un certain nombre de *trous* lorsque des parties de la séquence sont trop différentes. Cette méthode est basée sur l'hypothèse qu'il n'existe qu'un nombre limité de schéma de repliement dans la nature [58]. (ii) *L'appariement de segments* utilise un découpage de la séquence de la protéine en petits segments, et chaque segment est recherché dans une librairie de fragments de protéines dont le repliement est connu. Cette méthode est aussi basée sur l'hypothèse qu'une même séquence se replie de la même manière.

Dans la suite, on se concentre sur la première classe de méthode, la modélisation *de novo*.

Modélisation de novo des protéines Le principe des méthodes de modélisation *de novo*, est d'utiliser des simulations pour prédire la structure de protéines. Cette méthode est actuellement efficace pour des protéines de petite taille.

Plusieurs approches ont été développées pour prédire le repliement de protéines. Les méthodes de Monte Carlo utilise un échantillonage aléatoire des directions de mouvements. Un score d'interaction est calculé d'après une énergie potentielle d'interaction entre les atomes. Une direction de mouvement est acceptée quand l'énergie d'interaction est inférieure à un certain seuil. La dynamique moléculaire utilise une énergie potentielle pour intégrer les équations du mouvement de Newton. Ces deux méthodes utilisent une fonction potentielle qui sont des approximations de la réalité. Une de ces fonctions d'énergie (ENCAD [131]) est donnée sur la figure 1.11.

Initié en octobre 2001, le projet Folding@home[14] [178], propose une architecture pour construire l'ordinateur partagé le plus puissant dédié aux simulations de dynamique moléculaire. S'appuyant sur le principe du calcul en parallèle, n'importe qui peut installer sur son ordinateur personnel un écran de veille (un client) qui permet de lancer des simulations. Chaque client calcule de petits travaux, et les résultats sont communiqués à des serveurs centraux. L'algorithme utilisé suppose que le problème du repliement suit une chaîne de Markov : quand le système passe une barrière d'énergie, le système passe à l'état suivant. Tous les clients participent à l'exploration extensive des configurations possibles, en utilisant de la dynamique moléculaire d'après un état de départ commun mais avec des paramètres différents. Dès qu'un des clients franchit une barrière d'énergie (l'énergie est inférieure à un certain seuil), les serveurs en sont informés et répercutent l'information à tous les clients afin que ces derniers relancent de nouveaux calculs à partir du nouvel état.

Avec une puissance de calcul impressionnante [15], une simulation de repliement complète de petites protéines est possible en quelques jours. Néanmoins, certaines restrictions de l'architecture (pas de mémoire partagée rapide entre les clients) sont limitantes pour étendre ce principe à des simulations plus générales.

[14] `folding.stanford.edu`

[15] En septembre 2008, plus de 320000 processeurs étaient disponibles, incluant les processeurs des ordinateurs (tout système d'exploitation), les processeurs de cartes graphiques et même ceux des Playstation3.

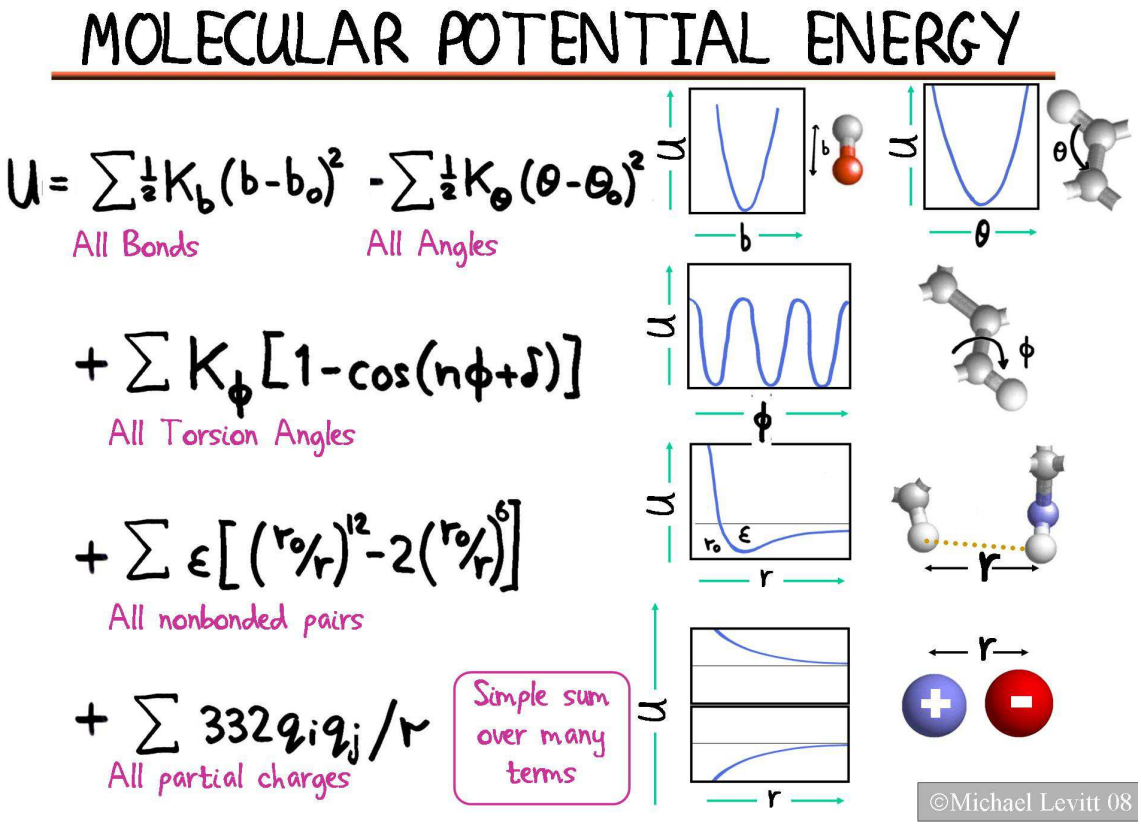


FIG. 1.11 – The total potential energy of any molecule is the sum of terms allowing the bond stretching, bond angle bending, bond twisting, Van der Waals interactions and electrostatics. Many properties of biomolecules can be simulated with such empirical energy function. (Image fournie par Michael Levitt [129])

Évaluation de structure et raffinement Nous allons maintenant nous intéresser à deux problèmes liés au repliement des protéines. (i) On considère un ensemble de conformations correspondant à différentes prédictions d'une même protéine. La structure déterminée expérimentalement (appelée la *structure native*) est supposée inconnue. Ce premier problème cherche à extraire les conformations qui sont proches de la structure native, et à attribuer un score en fonction de leur similarité à la structure native. (ii) Étant donné une conformation proche de la structure native, ce second problème vise à raffiner cette conformation de sorte à ce qu'elle soit encore plus proche de la structure native (proche au sens du RMSD à la structure native).

Ce dernier problème est connu sous le nom de *raffinement de structures*. Cette technique est utilisée pour affiner une structure prédite par un algorithme qui fournit un repliement grossier.

La plupart des méthodes qui traitent ces problèmes se basent sur une fonction d'énergie (souvent utilisable pour la dynamique moléculaire), qui sont de deux types : celles qui sont dérivées d'approximations de la physique, et celles qui sont dérivées de propriétés de structures de protéines de bases de données de structure expérimentale à haute résolution (dites empiriques).

Une vaste littérature traitant de ces problèmes est disponible, et le lecteur peut se référer à [179, 150, 136] pour plus de détails. L'introduction de certains articles sur le sujet sont aussi des sources de choix [191, 177]. Néanmoins, nous allons nous intéresser à certains travaux qui ouvrent des voies où la géométrie permettrait certaines améliorations.

Fogolari et al. se sont intéressés à l'extraction d'une fonction empirique d'après un jeu de protéines à haute résolution, en utilisant une représentation gros grains et en adaptant la notion d'interaction [94]. Confirmant l'efficacité des fonctions empiriques pour le problème de l'évaluation de repliement de structures, leur fonction est capable de retrouver la forme native et celles proches dans presque tous les cas présentés. La qualité de la base de données utilisée pour définir la fonction empirique est évidemment importante comme l'ont observé Liu et al. [138].

En utilisant une minimisation d'énergie dans le vide, Summa et Levitt [182] ont comparé l'efficacité de différentes fonctions physiques pour le raffinement de structure. De plus, ils ont proposé une fonction mélangeant le potentiel d'ENCAD [131], en remplaçant les termes d'interaction non-liés par une fonction empirique (utilisant les distances deux à deux, et par type de résidu [168]). Les performances de leur nouvelle approche s'est montré efficace vis à vis des fonctions physiques d'énergie disponibles gratuitement dans GROMACS [137].

Alors que de nombreux travaux utilisent des propriétés géométriques (principalement des quantités impliquant une paire d'atomes), peu ont abordé le problème avec la notion de voisinage.

Ngan et al. [151] ont dérivé une fonction empirique en utilisant le rayon du cercle circonscrit à trois résidus (avec une représentation gros grain), et la nature du triplé de résidus. Dans un autre travail, utilisant le complexe dual [16] de tous les atomes lourds, Li and Liang [135] ont défini une fonction empirique d'après les triplés d'atomes correspondant à un 2-simplexe dans le complexe dual. Krishnamoorthy et al. [123] ont quant à eux utilisé la distribution des tetraèdres de Delaunay, avec un modèle gros grain. Les tétrahèdres sont classés selon les quatre types de résidus, mais aussi selon l'adjacence le long du squelette de la protéine des résidus du tétraèdre. Pour apporter une solution au problème de stabilité en utilisant une fonction empirique basée sur Delaunay, l'approche précédente a été étendue plus tard par Bandyopadhyay et al. [21], en utilisant la notion de simplexes presque-Delaunay [20]. Pour chaque atome d'une protéine, Summa et al. ont défini la notion de *microenvironment*, comme étant l'ensemble des atomes à distance inférieure à r de cet atome (meilleurs résultats entre 4,5 et 4,8 Å), et qui sont à plus de s résidus le long du squelette (meilleures valeurs entre un et trois). Un atome central peut être de vingt types différents, et un microenvironment compte le nombre d'atomes parmi quatre classes.

L'association de fonctions potentielles dérivées de la physique avec des fonctions empiriques semble être une approche complémentaire intéressante. Le voisinage de chaque atome décrit probablement des propriétés fines des protéines repliées. Le microenvironment est un concept particulièrement intéressant mais présente plusieures limitations. Comme il s'appuie sur un compte d'atome (plutôt que sur une quantité continue), il est incapable de faire la différence entre deux atomes du même type à distance $\frac{r}{2}$

[16] Le complexe dual est un sous ensemble de la triangulation régulière, qui est équivalent au alpha complexe ($\alpha = 0$) défini pour un ensemble de points pondérés [79]. Une sphère/boule peut être considérée comme une point pondéré en utilisant son centre comme point, et son rayon au carré comme poids.

et r de l'atome central. De plus la coopération des atomes n'est pas vraiment prise en compte : le fait qu'un ensemble d'atomes soit distribué autour d'un atome ou juste d'un côté n'est pas encodé.

1.2.4 Le problème de l'amarrage

La fonction d'une protéine est déterminée par sa possibilité à interagir avec d'autres partenaires. Étant donné les structures des partenaires, un algorithme d'amarrage est une procédure qui prédit l'assemblage de ces partenaires. Un tel assemblage est appelé un *complexe*. Il y a trois grandes familles de complexes : les complexes protéine-protéine, les complexes protéine-médicament et les complexes protéine-acide nucléique. On remarquera qu'une protéine peut avoir une structure tertiaire légèrement différente en fonction qu'elle est déterminée expérimentalement dans un complexe (dite *forme complexée*) ou libre. En effet, la protéine peut se déformer globalement ou même se replier localement lors de l'association. Pour illustrer le problème de l'amarrage, nous considérons dans la suite le cas des complexes protéine-protéine et des complexes protéine-médicament (et ce bien que les complexes protéine-acide nucléique soient aussi fondamentaux).

Complexes protéine-médicament Les complexes protéine-médicament font partie des molécules les plus importantes pour les compagnies pharmaceutiques. En effet, les protéines sont les récepteurs de leurs médicaments, et une bonne compréhension de tous les processus liés à ce type d'assemblage est la clé pour fournir des traitements plus efficaces. Étant donné une protéine particulièrement intéressante (une cible thérapeutique), trouver toutes les régions de cette protéine susceptibles d'accueillir un médicament est un problème fondamental. Les atomes impliqués de la protéine sont considérés comme appartenant à une *poche* (cf. figure 1.12). Ce problème est en effet intéressant pour les fabricants de médicaments car, étant donnée une certaine poche, la connaissance des médicaments qui peuvent s'y loger et interagir avec les atomes de la poche fournit des informations sur l'action de ce médicament sur l'organisme (i.e. les effets primaires et éventuellement secondaire). Un procédure expérimentale, appelée le *filtrage*, consiste à prendre une protéine et essayer de faire interagir différents médicaments d'une bibliothèque in vitro. Ce processus est long et donc onéreux. Si l'on dispose de la structure de la protéine sélectionnée précédemment, et que l'on connaît l'emplacement de la poche, on peut alors effectuer un filtrage virtuel pour éliminer les médicaments qui ne peuvent pas ou ont de faibles chances d'interagir (en utilisant des critères géométriques ou chimiques), afin de se concentrer sur les molécules les plus prometteuses. De plus, les protéines homologues connues sont parfois utilisées afin de détecter de potentielles interactions et ainsi prévoir certains effets secondaires. Plusieurs algorithmes ont été développés pour calculer les poches dans les protéines. Pour détecter une poche totalement enfuie dans une protéine (i.e. une cavité), l'utilisation du complexe dual est clairement une bonne solution [79]. En effet, une union de boules et le complexe dual correspondant ont le même type d'homotopie. De plus, le complexe dual est inclus dans l'union de boules associée, et une cavité de l'union de boules est inclue dans une cavité du complexe dual (comme illustré sur la figure 1.10). Quand une poche n'est pas une cavité, plusieurs approches ont été développées, nous renvoyons le lecteur à une revue sur le sujet [124] pour plus de détails. Les algorithmes d'amarrage de médicament sur une protéine ont beaucoup utilisé le principe de la clé et de la serrure, n'utilisant que des transformations rigides lors de l'association. Néanmoins, les efforts sont maintenant dirigés de sorte à prendre en compte la flexibilité [181].

Tester les algorithmes d'amarrage Les complexes protéine-protéine sont centraux dans la majeure partie des activités d'un organisme vivant (réponse immunitaire, transmission des signaux, réparation de l'ADN, ...). L'étude et la compréhension de ces assemblages sont donc importants au vue des applications biologiques et médicales. En effet, de nombreuses maladies sont liées à un mauvais assemblage dû à une mutation de la protéine. Par exemple, la mutation d'un acide glutamique en une valine dans la séquence d'une chaîne de l'hémoglobine (un tétramère), se traduit par une perte d'élasticité de cette protéine (responsable du transport de l' oxygène) et d'un changement de sa forme quand elle libère l'oxygène (ce qui conduit à l'occlusion de vaisseaux et à une ischémie). Cette maladie est connue sous le nom de drépanocytose.

La plupart des expériences d'amarrage suppose que les deux partenaires donnés interagissent. Le problème de prédire si deux protéines interagissent est relativement difficile. Actuellement, trouver un

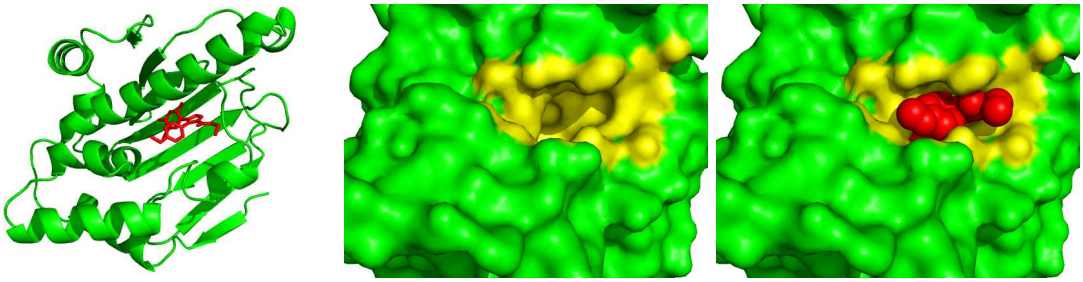


FIG. 1.12 – La protéine chaperonne HSP90 et un petit inhibiteur (PDB code `2BSM`). Une seule unité du dimère du domaine N-terminal est représentée. De gauche à droite : la protéine est représentée en mode cartoon, le ligand avec des bâtons; la surface de Connolly en jaune correspond à la localisation approximative de la poche qui accueille l'inhibiteur; en utilisant la représentation de Van der Waals, le ligand remplit la poche.

critère capable de répondre à cette question n'est pas trivial[17], mais est possible si les régions d'interaction des protéines sont connues [167]. Nous nous plaçons maintenant dans le cas de procédure d'amarrage où les deux partenaires ont été observés expérimentalement.

Un algorithme d'amarrage se décompose généralement en deux phases : une phase d'exploration qui consiste à explorer les positions relatives des deux partenaires, et une phase évaluation qui consiste à évaluer la qualité du complexe proposé. Pour rendre compte de la qualité des algorithmes d'amarrage (et augmenter la qualité de leurs prédictions), une procédure de test objective est nécessaire. L'expérience *Critical Assessment of PRediction of Interactions*[18] (CAPRI) est l'équivalente de CASP mais pour la prédiction de complexes. Lorsqu'un groupe de biologie structurale dispose de la structure d'un nouveau complexe, il a la possibilité de le soumettre au comité de CAPRI avant de le publier. Plusieurs complexes (deux ou trois) forment [19] une *manche*. Lorsque plusieurs manches ont été effectuées, une réunion est organisée pour discuter les performances et limitations des algorithmes de prédiction. La troisième réunion de CAPRI a été organisée en avril 2007 [115]. Les principales conclusions de cette réunion [127] sont que les algorithmes sont meilleurs lorsque au moins un des deux partenaires est fourni dans sa forme complexé (le cas où les deux partenaires sont en forme complexé n'est généralement pas traité car il n'est pas biologiquement significatif). Les partenaires de la plupart des complexes proposés étaient en forme libre, et la modélisation des changements de configuration reste une tâche difficile pour les algorithmes de prédiction : Sur certains complexes, aucun groupe n'est parvenu à des prédictions mieux classées que *acceptable*. Nous invitons le lecteur à parcourir le *prediction report* [127] pour une discussion plus détaillée des résultats et connaître comment les prédictions sont jugées.

Évaluer les prédictions Un des axes de recherche dans CAPRI est la définition de fonctions pour évaluer la qualité des complexes prédits (dites *fonctions de score*). Dans cet optique, CAPRI propose une manche d'évaluation qui suit chaque manche de prédiction. En plus des dix prédictions par complexe, les groupes prédicteurs sont autorisés à soumettre leurs cent meilleures prédictions par complexe. Après une étape de mélange, les groupes *évaluateurs* sont invités à indiquer les dix meilleures prédictions parmi toutes celles fournies par les groupes prédicteurs. Une situation qui justifie le besoin d'améliorer les fonctions de score est la suivante : un groupe évaluateur a choisi parmi les meilleures prédictions, celle d'un autre groupe, alors que la qualité de cette prédiction a été classée dans une catégorie inférieure à celle d'une ses propres prédictions (faites en tant que groupe prédicteur). Les algorithmes actuels sont donc peut être capables de prédire des complexes de très bonne qualité, mais les fonctions de score ne sont pas capables de les détecter.

La bibliographie des algorithmes d'amarrage et des fonctions de score est importante. Pour plus de

[17] en particulier si un complexe prédit n'a jamais été observé expérimentalement

[18] `http://www.ebi.ac.uk/msd-srv/capri/`

[19] Le nombre de complexes dont la structure est connue est relativement faible. En septembre 2008, le site web de RCSB en rapportait 2141 sur les 52,959 structures disponibles.

détails sur les algorithmes d'amarrage nous orientons le lecteur vers, bien entendu les références du prediction report [127] de CAPRI, mais aussi une revue plus ancienne [107]. Ce dernier travail présente aussi les précédents développements des fonctions de score. Le critère principal utilisé est la complémentarité géométrique. Mais même si elle est nécessaire pour la stabilité du complexe, elle n'est pas suffisante. La nature chimique des interactions et aussi essentielle. La complémentarité géométrique est maintenant utilisée comme une première étape de filtrage des complexes mal appareillés. Les recherches actuelles sur les fonctions de score sont maintenant tournées vers l'utilisation d'une combinaison pondérée de plusieurs termes (incluant des paramètres géométriques) et plus particulièrement des termes énergétiques [157].

Cependant, les propriétés géométriques pourraient être des sources d'amélioration. Plusieurs travaux [114, 64], utilisant la définition d'interface comme l'ensemble des atomes qui perdent de l'accessibilité au solvant, ont étudié le nombre de résidus impliqués à l'interface, le nombre de liaisons hydrogènes, la compacité des résidus à l'interface ou encore l'aire de la surface accessible au solvant perdue. Utilisant le diagramme de Voronoï d'une représentation en gros grain des résidus d'un complexe, le volume des cellules à l'interface a été utilisé comme un des paramètres pour une méthode empirique d'évaluation [29]. Une fonction empirique qui se base sur le complexe dual d'un modèle gros grain de complexes a été utilisée comme fonction de score [134]. Pour définir cette fonction, ils ont utilisé pour chaque paire de pseudo-atomes connectés par un 1-simplexe du complexe dual, la taille de l'étoile de chaque pseudo-atome dans le dual complexe. La définition de l'interface des complexes en utilisant une filtration du diagramme de puissance entre les deux partenaires (en plus de permettre l'étude de la courbure, la taille, la connectivité de l'interface) a montré que certains atomes peuvent être considérés comme à l'interface mais ne perdent pas d'accessibilité [43].

Les études géométriques ont principalement été restreintes à la surface des protéines perdant de l'accessibilité. Cette remarque pose la question du possible rôle d'une seconde couche d'atomes/résidus proches de la surface d'interaction. De plus, on a beaucoup parlé d'atomes qui deviennent couverts dans un complexe. Une autre question intéressante à étudier serait le rôle des interactions à plusieurs corps, en considérant en particulier quels sont les types d'atomes qui couvrent un certain type d'atome à l'interface.

1.3 Contributions et Vue d'Ensemble de la Thèse

Si l'on s'intéresse aux aires des contacts géométriques, les contacts deux à deux ont principalement été étudiés. Une raison possible est peut être le manque de méthodes facilement implémentables pour franchir le cap des paires. Pour ce faire, on a besoin d'avoir pour chaque atome, le décomposition de la surface de sa sphère en faces. Une face est définie comme la composante connexe de la surface de la sphère atomique qui est couverte par exactement les mêmes boules atomiques (ou éventuellement aucune). Une construction de géométrie algorithmique qui répond parfaitement à ces critères est l'arrangement sur la surface de chaque sphère atomique, des cercles provenant de l'intersection avec les sphères atomiques voisines. Pour chaque face, il convient aussi de rapporter la liste des atomes dont la boule couvre cette face. On remarquera qu'une région couverte par k boules correspond a un contact d'ordre $k+1$. Cette construction permet donc de traiter le problème de l'encodage des contacts à plusieurs corps. Elle généralise aussi les algorithmes calculant les classiques SAS ou surface de VdW (et leurs aires), ainsi que ceux mentionnés précédemment pour les aires des surfaces de contact. Une telle méthode permet donc d'être utilisée pour les questions posées au sujet des fonctions empiriques de potentiel, ainsi que les fonctions de score, pour le repliement et l'amarrage.

Dans le chapitres qui suivent, nous présentons une solution exacte au problème du calcul de l'arrangement de cercles sur une sphère, avec une liste de boules par face (appelée *liste couvrante*). La description se décompose comme suit. Le chapitre 3 se consacre à la description de l'adaptation de l'algorithme de Bentley-Ottmann pour calculer tout les points d'intersection d'un ensemble de cercles sur une sphère. Le chapitre 4 décrit une extension de cet algorithme pour construire explicitement l'arrangement ainsi que les listes couvrantes. Les primitives algébriques et géométriques nécessaires pour ces algorithmes font partie du concept du noyau sphérique, qui est développé dans le chapitre 5. Côté application, le chapitre 6 présente une application pour un algorithme d'amarrage avec boucles flexibles, où l'arrangement et les listes couvrantes sont utilisés pour améliorer les prédictions. Pour finir, le chapitre 7 détaille l'interface des implémentations des algorithmes et concepts introduits dans les chapitres 3, 4 et 5.

1.3.1 Chapitre 3: Trouver tous les Points d'Intersection d'un Ensemble de Cercles sur une Sphère

Motivations et travaux antérieurs

Historique En 1976, Shamos et Hoey [176] ont décrit un algorithme pour déterminer si au moins deux segments parmi n dans le plan s'intersectent (en temps $O(n \log n)$ et $O(n)$ mémoire), ainsi qu'une application pour détecter si deux polygones simples se coupent. En réponse à une question posée dans cet article, Bentley et Ottmann [23] en 1979 ont proposé un algorithme sensible à la sortie pour calculer les k points d'intersection d'un ensemble de n segments dans le plan. L'algorithme fonctionne en temps $O((n+k)\log n)$ et a besoin d'un espace mémoire de taille $O(n+k)$ (cette taille a été améliorée à $O(n)$ sans changer la complexité en temps par Brown en 1981 [34]). Le premier algorithme calculant les k points d'intersection d'un ensemble de n segments dans le plan en temps linéaire en k a été donné par Chazelle [49] en 1986. La complexité est $O(n(\log^2 n/\log\log n)+k)$ en temps, et $O(n+k)$ en mémoire. Cet algorithme a été amélioré à la complexité optimale en temps $O(n\log n+k)$ par Edelsbrunner et Chazelle [50] en 1992. En 1995, Balaban a publié un algorithme avec la même complexité en temps mais a amélioré la taille mémoire nécessaire à $O(n)$. Une version incrémentale a aussi été fournie par Chazelle et al. [53] avec un complexité de $O(n^2)$ en temps et mémoire. Nous citons aussi les travaux de Clarkson et Shor [60], et Mulmuley [147], qui ont décrit des algorithmes randomisés pour ce problème avec une complexité de $O(n\log n+k)$ en temps et $O(n)$ en mémoire. Pour ce qui est des droites, Edelsbrunner et Guibas ont introduit l'idée d'un balayage topologique (remplaçant la droite de balayage par une courbe de balayage) [80, 81] qui fonctionne en temps $O(n^2)$ et qui a besoin de $O(n)$ mémoire pour calculer l'arrangement d'un ensemble de n droites dans le plan. Cet algorithme utilise le fait que chaque paire de droites se coupent une fois. (Rafalin et al. ont proposé une extension pour traiter directement les cas dégénérés [161]). Beaucoup de propriétés des arrangements de différents objets géométriques ont été étudiées (complexité d'une face, complexité d'une zone,...) et le lecteur est invité à consulter [2, 4, 14] pour plus de détails. Les algorithmes d'intersection d'objets géométriques sont passés en revue dans [98], et quelques algorithmes parallèles sont présentés dans le chapitre 4.1 de [8] (et aussi dans [10] par exemple). Pour plus d'informations sur les arrangements en général, la revue la plus récente sur le sujet est [92].

Problèmes de robustesse pour balayer des cercles sur une sphère Les implémentations récentes d'algorithmes géométriques font généralement la distinction entre d'un côté les prédicats et les constructions, et de l'autre la partie combinatoire d'un algorithme (celle qui fait appel aux prédicats et construction).

Un *prédicat* est une fonction de test qui retourne un élément d'un ensemble fini de valeurs (par exemple, *est-ce que deux courbes c et c' se coupent?*). Une *construction* est une fonction qui construit de nouveaux objets (par exemple, *calcule le point d'intersection entre les courbes c et c'*). La robustesse d'un algorithme géométrique est donc basé sur le comportement attendu des prédicats et constructions vis à vis de la partie combinatoire, mais aussi de comment la partie combinatoire a été réalisée (c'est à dire comment elle traite les cas dégénérés). Les problèmes liés à l'évaluation des prédicats et constructions sont introduits dans la section 1.3.3, et traités dans le chapitre 5. Le problème qui nous intéresse ici est la partie combinatoire, et le traitement des cas dégénérés (par exemple quand deux cercles sont tangents). Pour ce faire, on s'appuie principalement sur deux stratégies. La première est de traiter explicitement ces cas. La seconde est d'utiliser des perturbations, qui permettent d'écrire la partie combinatoire de l'algorithme sous l'hypothèse que les données d'entrée sont en *position générale* (i.e. non dégénérées). Il y a deux types de perturbations efficaces: (i) les perturbations explicites contrôlées qui consistent à perturber les données d'entrée explicitement et à garantir la position générale (utilisé par exemple dans [106]). (ii) les perturbations implicites [82] qui consistent à appliquer une perturbation symbolique des données d'entrée, en utilisant un polynôme en ϵ qui s'annule avec ϵ. Un cas dégénéré est détecté à l'intérieur d'un prédicat en utilisant les entrées non perturbées. Dans un tel cas, l'évaluation du prédicat doit se faire avec les données perturbées en utilisant une valeur assez petite de ϵ. Pour ce qui est de trouver l'arrangement exact (ou simplement les points d'intersection), aucun de ces schémas de perturbations n'est utile. Les perturbations explicites changent les données d'entrée, alors que les perturbations implicites induisent un changement cohérent des décisions prises pendant le déroulement de l'algorithme, le plongement restant

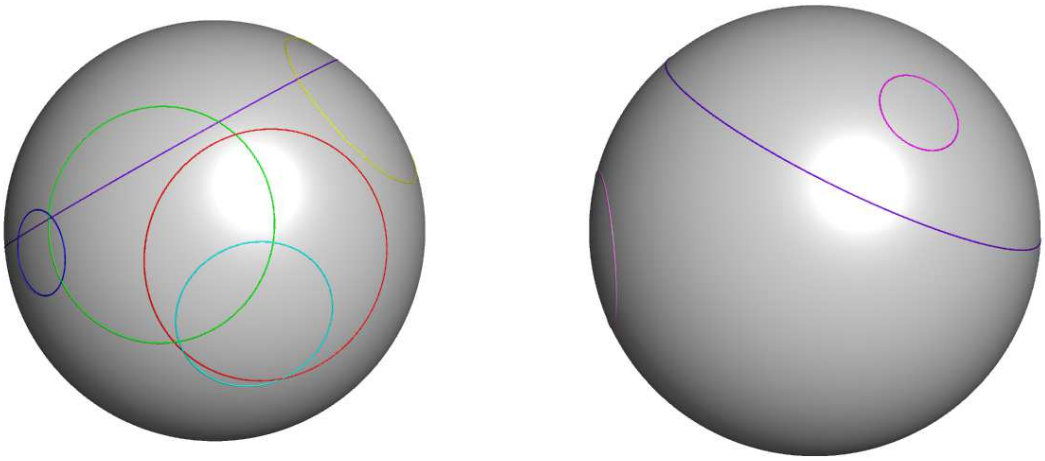


FIG. 1.13 – Huit cercles sont dessinés sur une sphère. Le chapitre 3 présente un algorithme pour calculer les points d'intersection de ces cercles (la scène est affichée de deux points de vue opposés).

le même. Dans le cas de deux cercles tangents, un prédicat d'intersection va indiquer soit que les cercles sont disjoints soit qu'ils se coupent en deux points, alors que la construction ne fournira qu'un seul point.

La sphère contre le plan Dans sa version originale, l'algorithme de Bentley-Ottmann [23] évoque déjà comment traiter le cas des cercles dans le plan. En théorie, la transition de la sphère au plan ne nécessite qu'une simple projection stéréographique. Côté implémentation, ce n'est pas clair. En effet, les opérations impliquées dans l'algorithme de balayage sont les mêmes dans le plan et sur la sphère, hormis les cas dégénérés spécifiques à la sphère. Comme on le verra dans le chapitre 3, ces cas dégénérés sont faciles à prendre en compte et ne pénalise pas la complexité du temps de calcul. De plus la projection stéréographique présente aussi des cas dégénérés lorsqu'un cercle passe par le centre de projection (pour les détails voir la section 3.1).

Contributions

Étant donné une collection de cercles sur une sphère, le travail présenté dans le chapitre 3 est dédié à la détermination de l'ensemble des points de la sphère qui sont communs à au moins deux cercles. L'algorithme introduit utilise une structure de données originale pour la gestion des cas dégénérés. La représentation des intersections permet de permuter les arcs de l'ordre vertical de manière combinatoire, ainsi que de maintenir la taille de la queue de priorité linéaire. Les temps de calculs et la mémoire nécessaire sont concernés par ce dernier aspect, comme le montrent les résultats de la section expérimentale 5.5. Pour finir, notre algorithme permet la construction de l'arrangement d'un ensemble de cercles sur une sphère et des listes couvrantes du chapitre 4.

1.3.2 Chapitre 4: Construction de l'Arrangement Induit par un Ensemble de Cercles sur une Sphère, accompagné des Listes Couvrantes

Motivations et travaux antérieurs

Applications Soit S_0 une sphère et une liste de cercles sur cette surface. L'*arrangement* de ces cercles sur S_0 est la partition de S_0 induite par ces cercles, en régions dont l'intérieur est connexe (voir figure 1.13). En général, la plupart des applications impliquant des sphères ou des boules (voir aussi l'introduction du chapitre 5 à la section 1.3.3) se basent sur de simples tests géométriques, et en particulier l'intersection. Néanmoins, certaines d'entre elles, peuvent tirer partie d'informations plus fines comme l'arrangement. Plusieurs applications ont déjà été développées à l'aide de cette construction, pour représenter les directions dans lesquelles un objet peut se déplacer. Utilisant un point par objet, chaque

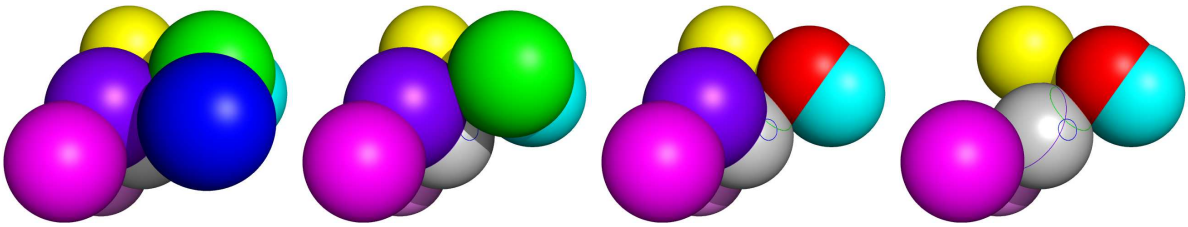


FIG. 1.14 – Huit sphères pénètrent une sphère grise. Le chapitre 4 présente un algorithme qui calcule l'arrangement sur la sphère grise des cercles d'intersection. De plus, pour chaque face, la liste des sphères dont la boule correspondante couvre cette face est aussi fournie.

région de l'arrangement décrit un ensemble de directions soit interdites soit autorisées. Par exemple, un arrangement d'arcs de grands cercles sur une sphère unité a été employé dans des problèmes de moulage, pour définir les ensembles de directions selon lesquelles un objet polyédrique peut être extrait de son moule [5].

Un tel arrangement s'est aussi montré utile pour la planification de l'assemblage de pièces mécaniques pour définir des directions sans collisions [164](sous-section 3). En radiochirurgie, un arrangement d'arcs de grands cercles ont permit de proposer une méthode pour améliorer et accélérer le traitement des tumeurs du cerveau, en définissant les régions interdites où le faisceau pourrait endommager le cerveau du patient [174]. En biologie structurale computationelle, les arrangements perturbés de cercles sur une sphère atomique ont été utilisés pour calculer le bord d'une molécule représentée par une union de boules [106].

Applications en biologie structurale computationelle Dans la section 1.2.2, l'importance des contacts deux à deux a été mise en avant. On suppose maintenant que chaque cercle sur S_0 provient de l'intersection de S_0 avec une sphère S_i, et la boule bordée par cette sphère est notée B_i. La liste de couverture d'une face de l'arrangement est la liste des boules qui contiennent la face, et la multiplicité d'une face est la taille de sa liste couvrante (voir figure 1.14). Ce qui suit introduit un problème de biologie structurale computationelle qui est directement concerné par cette extension. On considère le problème de l'encodage des contacts à plusieurs corps, entre un atome et ses atomes voisins. En représentant une molécule comme une collection de boules, l'importance des surfaces moléculaire a été rappelée précédemment. Pour un atome donné d'une molécule, considérons l'arrangement induit par les cercles d'intersection avec ses atomes voisins. La contribution de cet atome à la surface moléculaire (par exemple la SAS) correspond aux faces de multiplicité zéro. La figure 1.15 représente l'aire cumulée (sur tous les atomes d'un complexe) en fonction de la multiplicité. On notera en particulier que l'aire de la SAS correspond à environ 4,5% de l'aire totale. Ce chiffre montre que l'étude des faces de multiplicité > 1 peut fournir de précises informations, et l'arrangement de cercles est un des moyens permettant l'étude des interactions entre plusieurs atomes.

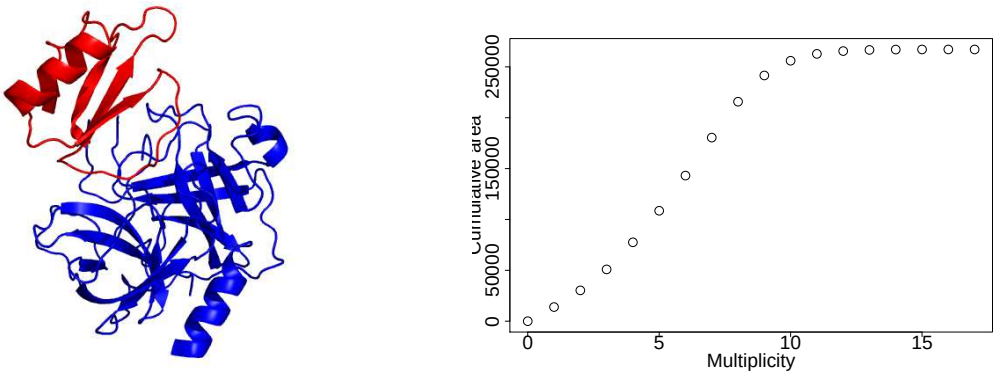


FIG. 1.15 – Modéliser les contacts atomiques à plusieurs corps. Gauche : le squelette d'un complexe protéine-protéine (l'alpha-chymotrypsine-eglin c, code PDB `1ACB`), formé de deux partenaires; Droite : aire de la surface cumulée, sur tous les atomes du complexe, en fonction de la multiplicité des faces. La multiplicité zéro correspond à l'aire de la SAS et ne représente que 4,5% de l'aire cumulée.

Un autre problème utilisant les arrangements de cercles, en rapport avec les algorithmes d'amarrage avec boucle flexible est introduit dans la section 1.3.4, et décrit dans le chapitre 6.

Algorithme pour calculer un arrangement de cercles sur une sphère Un algorithme calculant une carte des trapèzes sur une sphère induite par un ensemble de cercles sur une sphère a été décrit [106, 86]. Les cas dégénérés sont supprimés en utilisant des perturbations explicites. Le premier algorithme complet et exact calculant une carte planaire induite par les courbes d'intersection d'un ensemble de quadriques sur l'une d'entre elle, disons Q, est décrit dans [26]. Cette implémentation réduit le problème à un calcul d'arrangement de courbes algébriques dans le plan en utilisant une projection. De plus, il est aussi montré comment à partir de cette carte planaire, on peut reconstruire l'arrangement sur Q (une étape qui peut être évitée en le calculant directement sur Q)[20]. S'inspirant de tous ces travaux, un algorithme de balayage générique pour les surfaces a été proposé [25]. Étant donné un ensemble de courbes sur une surface, l'arrangement de ces courbes est obtenue grâce à cet algorithme, à condition de fournir un ensemble de primitives qui remplissent une liste de pré-requis de l'algorithme. Pour le moment, pour le cas de cercles sur une sphère, des primitives pour traiter le cas des grands cercles seulement sont disponibles[21] Pour finir, nous mentionnons aussi un algorithme qui permet de calculer une décomposition volumique induite par un arrangement de quadriques. Cette stratégie ne repose pas sur des arrangements sur des surfaces, et aucune implémentation n'est disponible.

Contributions

Le chapitre 4 présente comment construire l'arrangement de cercles sur une sphère, ainsi qu'une représentation compacte des listes couvrantes (appelée arbre couvrant). Notre travail apporte donc trois contributions.

(i) Nous présentons le premier algorithme qui est capable de calculer de manière exacte n'importe quel arrangement de cercles sur une sphère. Il est basé sur l'adaptation sphérique de l'algorithme de Bentley-Ottmann du chapitre 3, qui traite explicitement les cas dégénérés.

(ii) La première description d'une extension de l'algorithme de balayage pour calculer un arrangement de segments dans le plan a été faite en 1982 dans [152]. Cette extension retourne des cycles orientés d'arêtes qui décrivent des régions simplement connexes du plan. Dans le chapitre 4, nous décrivons comment construire un structure en demi-arêtes étendue (autorisant les faces non simplement connexes)

[20] Néanmoins, cette stratégie est suffisante pour calculer les points d'intersection sur Q (problème du chapitre 3), mais un effort supplémentaire est nécessaire pour construire l'arrangement.

[21] En utilisant le noyau sphérique de CGAL présenté dans le chapitre 5, nous avons commencé l'écriture des primitives qui permettraient de manipuler tous les types de cercles avec cet algorithme

pour définir l'arrangement, une construction qui se fait à la volée pendant le balayage, en utilisant le principe d'Union-Find.

(iii) En utilisant les propriétés du balayage, une méthode pour construire l'arbre couvrant des faces de l'arrangement de manière compacte est aussi présentée.

La seule alternative, est l'algorithme calculant la carte des trapèzes à partir de perturbations explicites contrôlées [106, 86]. Notre méthode cumule plusieurs avantages. Premièrement, le calcul de l'arrangement est exact, ce qui permet de garantir la robustesse. Même si l'exactitude n'est pas un pré-requis des applications que nous visons en biologie structurale computationelle, d'autres applications peuvent en avoir besoin. De plus, l'exactitude de l'arrangement est garantie avec un petit surcoût (sans changer les entrées), comme on peut le constater dans la section 5.5. Deuxièmement, utiliser un algorithme de balayage d'une sphère fournit une méthode pour construire directement l'arrangement (plutôt que la carte des trapèzes [31]), ainsi que des listes de couvertures.

1.3.3 Chapitre 5: Opérations Algébriques et Géométriques Nécessaires pour Manipuler des Cercles et des Sphères en 3D

Motivations et travaux antérieurs

Primitives courbes et linéaires en géométrique algorithmique. Jusqu'à récemment, les algorithmes géométriques se limitaient aux objets linéaires dans l'espace affine Euclidien (points, segments, triangles,...), les objets courbes étant approchés par des éléments linéaires. Néanmoins, manipuler directement des objets courbes permet d'éviter cette étape de discrétisation, évitant d'augmenter le nombre d'objet à manipuler et permettant de meilleures complexités pour les algorithmes. Pour ces raisons, manipuler directement des objets courbes est important en géométrie algorithmique. Les primitives les plus simples parmi les objets courbes sont les cercles, les arcs de cercles et les sphères qui sont centraux pour soit représenter soit approcher des objets plus compliqués.

Si l'on s'intéresse aux objets uni dimensionnels, les arcs de cercles dans le plan sont impliqués dans le design et la production de circuits imprimés et intégrés [37], et sont aussi utilisés pour approcher des courbes [77, 6].

Les boules et les sphères sont probablement encore plus importantes, en particulier pour approcher des objets 3D. Étant donné un objet en 3D connu, des boules peuvent être utilisées pour fournir des approximations hiérarchiques, pour par exemple effectuer des tests de collision rapide [3], ou faire de la visualisation à différentes échelles [166].

Étant donné un nuage de points en 3D échantillonnant un objet inconnu, les boules sont centrales pour l'inférence des propriétés géométriques et topologiques de l'objet inconnu d'après le nuage. A propos de la topologie, les boules peuvent en effet être utilisées pour approcher l'axe médian (à l'aide des centres) et l'objet (par l'union des boules), ce qui est une idée principale de l'algorithme du *power crust* [11]. Pour certains compacts, l'union des boules centrées sur les points du nuage permet de retrouver la topologie du compact sous certaines conditions d'échantillonage [47]. D'un côté plus géométrique, un certain nombre de quantités peuvent être déduites grâce aux boules centrées sur les point du nuage de points. Par exemple, la mesure du bord d'un nuage de points, grâce à laquelle les points singuliers et les arêtes vives d'un modèle échantillonné peuvent être estimés [48]. Cette inférence nécessite de calculer le volume d'intersection d'une boule et de sa cellule de Voronoï, ce qui implique des constructions géométriques sur une sphère [141, 18]. Si on considère un polyèdre 3D, l'offset de ce polyèdre peut être défini comme une union de boules (l'offset est équivalent à la somme de Minkowski entre le polyèdre et une sphère). Le volume occupé par des objects en 3D peut être approximée par une union de boules [163]. Pour plus de références sur les représentations qui utilisent des sphères, le lecteur peut consulter le chapitre 11.12 de Visionbib[22].

Pour finir, les sphères sont aussi centrales en modélisation moléculaire avec le modèle de Van der Waals, comme rappelé précédemment.

Prédicats et Constructions en géométrie algorithmique Les implémentations récentes des algorithmes géométriques distinguent les prédicats et les constructions. Comme les erreurs d'arrondi dû

[22] http://www.visionbib.com/bibliography/describe482.html

au calcul flottant sont souvent sources d'instabilité [120], la robustesse des algorithmes géométriques se base sur l'évaluation exacte des prédicats [196]. Ce problème se ramène en général à l'évaluation du signe d'une expression polynomiale sur les données d'entrée, ce qui peut être réalisée efficacement en utilisant des techniques de filtrage arithmétiques [132]. En dehors de cette stratégie classique, une autre solution consiste à garder en mémoire des objets géométriques intermédiaires, afin de pouvoir les utiliser plus tard afin d'évaluer des prédicats. Dans le cas des objets courbes, la première stratégie consiste typiquement à utiliser un encodage implicite d'un nombre algébrique par un polynôme (s'annulant en ce nombre), et fait appel à des calculs de résultants pour l'évaluation de prédicats. La seconde stratégie s'appuie sur une représentation explicite mais exacte des nombres algébriques [65, 125], et nécessite l'utilisation de calculs multi-précision jusqu'à une borne de séparation pour l'évaluation de prédicats [35, 133]. Ce procédé permet de réutiliser les calculs intermédiaires et rend la tâche du design des prédicats (difficile pour des objets algébriques) plus facile. Les travaux sur les constructions filtrées [96, 88] devrait réconcilier les deux stratégies. L'implémentation d'un tel cadre nécessite encore un peu de travail pour dépasser les performances des prédicats n'utilisant pas de constructions intermédiaires.

La bibliothèque CGAL, noyaux et classes de traits Le but du projet à code ouvert CGAL est de promouvoir la recherche en géométrie algorithmique, de sorte que les algorithmes de référence soient disponibles sous la forme d'implémentation robuste et de qualité pour la recherche académique et les applications industrielles. Le lecteur est invité à consulter le site du projet CGAL ainsi qu'à la littérature pour plus de détails sur le projet [44, 93]. Les algorithmes géométriques sont génériques, et une classe géométrique est instanciée à l'aide d'une *classe de traits* (offrant l'ensemble minimum de fonctionnalités requises). Un tel paramètre *patron* est supposé fournir un jeu de fonctionnalités, avec des signatures précises, décrit et documenté comme un *concept*.
Dans la bibliothèque CGAL les objets géométriques de taille constante, ainsi que des prédicats et constructions généraux sur ces objets sont souvent regroupés dans des *noyaux*. Comme les noyaux peuvent être utilisés directement comme classe de traits par plusieurs classes CGAL, les noyaux de CGAL sont tous documentés comme des concepts C++(c'est à dire qu'une implémentation d'un concept donné est un *modèle* du concept).

Cercles et sphères: des primitives aux algorithmes Dans la continuité des algorithmes dédiés aux primitives linéaires, un courant récent en géométrie algorithmique a été la spécification d'algorithmes et programmes tournés vers les courbes et les surfaces. Les travaux se sont essentiellement concentrés sur les arrangements en 2D et 3D, ce qui implique des primitives tel que le calcul d'intersections ou la position relative entre deux objets.

Dans le cas du plan, MAPC a été la première bibliothèque géométrique permettant de manipuler des courbes algébriques génériques [121], mais les cas dégénérés n'étaient pas traités. La bibliothèque prototype EXACUS permet de calculer des arrangements de coniques et de cubiques dans le plan [85]. La première version d'un noyau pour des objets non-linéaires (cercles et arcs de cercle en 2D) a été disponible à partir de la version 3.2 de CGAL [158]. Le paquet arrangement de CGAL peut être utilisé avec ce noyau pour calculer des arrangements d'arcs de cercle dans le plan. Hormis ceci, ce paquet peut aussi manipuler des coniques et des courbes de Bézier [193].

Pour ce qui est du cas 3D, Andrade et Stolfi ont proposé des opérations pour manipuler des arcs de cercle, implementées en Modula-3 [12]. Les objets sont représentés à l'aide des coordonnées de Plücker et les opérations sont limitées à une sphère donnée (alors que nous utilisons une définition générale des arcs de cercle en 3D).

Contributions

Le travail présenté dans le chapitre 5 est la première implémentation complète et efficace des fonctionnalités essentielles pour manipuler de manière exacte des sphères, cercles et arcs de cercle en 3D, et ce en répondant aux attentes des applications mentionnées. Le but était de fournir un paquet prêt à être utilisé. En suivant les meilleures pratiques, une séparation nette a été faite entre l'algèbre et les aspects géométriques d'un côté, et le concept du noyau et son implémentation de l'autre. Le concept du noyau sphérique est illustré par une application pour calculer l'arrangement exact de cercles sur une

sphère. Comme label de qualité, la partie basique du paquet (avec les sphères, cercles et arcs de cercle généraux en 3D) a été examinée et acceptée par le comité éditorial de CGAL [38]. La seconde partie du paquet, dédiée aux objets sur une sphère de référence, suivra le même processus dans un second temps. En particulier, la diffusion de notre code va permettre d'autre applications. Par exemple, une classe de traits dédiée à tous les types de cercles sur une sphère, requise par l'algorithme générique de balayage de courbes sur une surface, pourra être développée [25].

1.3.4 Chapitre 6: Utilisation de l'Arrangement de Cercles pour la Sélection de Conformations Diverses, Appliqué à un Algorithme d'Amarrage avec Boucle Flexible.

Motivations et travaux antérieurs

Ensembles pour la modélisation moléculaire Les interactions entre protéines sont essentielles pour tous les processus biologiques, mais la prédiction de complexes nécessitant de tenir compte de la flexibilité des partenaires est délicate, comme l'atteste l'expérience menée par CAPRI où le nombre de prédictions classées *medium* ou *high* est faible (par opposition au nombre de prédictions classées *incorrect* et *acceptable*) [115]. Comme les protéines sont par nature flexibles, elles subissent des changements de conformations au cours du temps, ou de manière équivalente, on peut considérer qu'il existe à un instant donné un ensemble de conformations en équilibre. Au cours de l'exploration de l'espace des conformations, les conformations occupent des régions caractérisées par des énergies libres faibles. Pour les protéines de taille moyenne qui subissent des mouvements de faible amplitude sur une échelle de temps d'une dizaine de nanosecondes, les changements de conformation peuvent être étudiés avec la dynamique moléculaire. Pour des cas plus compliqués où la flexibilité s'applique à de larges parties du squelette d'une protéine ou lorsque l'amplitude du mouvement est importante, un ensemble fini de conformations pré-calculées (appelé des *conformers*) peut être utilisé simultanément. Cette représentation est particulièrement appropriée lorsque l'on considère l'amarrage de macromolécules. Dans le cas de l'assemblage de protéines, on veut prédire la meilleure complémentarité géométrique des deux partenaires, ce qui nécessite une exploration des positions et orientations des deux partenaires, mais aussi de leurs espaces de conformation. Dans l'interprétation de Monod-Wyman-Changeux [144], les protéines en forme libre sont considérées comme des ensembles de conformers en équilibre thermodynamique. Lorsque les partenaires s'associent, l'équilibre est déplacé vers celui de la structure complexée observée. L'implémentation de cette stratégie peut se faire à un niveau global (i.e. la protéine) [105], ou local (i.e. les chaînes latérales) [39], ou intermédiaire (i.e. boucles ou domaines) [22].

Génération et sélections de conformers : énergie ou géométrie Représenter la flexibilité à l'aide d'un ensemble de conformers est informatiquement possible (en temps de calculs), si sa taille n'est pas trop importante. Il est donc essentiel que ce nombre réduit de conformers soit le plus représentatif de l'espace des conformations disponible pour chaque fragment de molécule. Plus généralement, les conformations étant intéressantes pour un certains nombre d'applications, quel critère (géométrique ou énergétique) faut-il utiliser pour les générer/sélectionner? D'un point de vue statistique, l'énergie devrait être le critère de choix pour la génération d'ensembles représentatifs de l'équilibre thermodynamique entre les conformations. Néanmoins, ce critère est en général pour diverses raisons inutilisable.

Premièrement, l'exploration exhaustive de l'espace de conformations pour de gros systèmes ou des systèmes se déformant avec de larges amplitudes de mouvements, n'est pas possible. Pour que les calculs restent raisonnables, les méthodes qui traitent ce cas favorisent des calculs géométriques, et les calculs d'énergie sont faits dans des étapes postérieures [67, 73]. Deuxièmement, lorsque les conformers sont utilisés pour modéliser une région d'une protéine, l'énergie associé à chaque conformer varie en fonction de son environnement. Dans le cas de l'amarrage par exemple, l'énergie de chaque copie dépend de ses interactions avec le second partenaire (interactions électrostatiques ou de Van der Waals, énergie de désolvatation,...). L'attribution d'un poids à chaque conformer comme s'il était seul ne tient donc pas compte en général de sa probabilité d'apparition. Troisièmement, il est possible que le paysage énergétique associé à une protéine soit plutôt plat, avec des barrières énergétiques faibles entre les conformers (difficile

de séparer les conformations). Dans ce cas, il est important d'être capable d'échantillonner exhaustivement l'espace disponible pour les éléments flexibles.

On notera aussi que la génération d'ensembles variés est une stratégie de choix pour simuler des processus compliqués. Par exemple, des ensembles variés générés à l'aide de potentiel en parapluie répulsif ont récemment été utilisés pour étudier les échanges de domaines [140].

Contributions

Soit un ensemble $\mathcal{C} = \{C_1, \ldots, C_n\}$ de n conformers (rotamers, boucles de protéines, protéines complètes), chacun d'entre eux étant représenté par une collection de boules, chaque boule bordée par une sphère. Ce modèle est relativement général, les boules pouvant être celles du modèle de (VdW) pour représenter les atomes, ou modéliser un résidu (on considère des modèles tout-atomes ou gros grains pour les protéines). Comme nous l'avons vu, l'échantillonage de l'espace des conformations disponible est un pré-requis important. Dans le chapitre 6, nous voulons résoudre le problème suivant :

Étant donné un ensemble pré-calculé de n conformers et un entier $s < n$, trouver une *sélection* de s conformers qui maximise un critère géométrique donné.

Afin de définir le type de critère géométrique que nous visons, on notera que l'union des boules de tous les conformers de la sélection définit un volume, dont la partition par les sphères bordant les boules est appelée un *arrangement volumique*. De la même façon, la décomposition de chaque sphère par les cercles d'intersection avec les autres sphères définit un *arrangement surfacique*. Les figures 1.16 et 1.17 sont des illustrations en deux dimensions de ces arrangements. A l'aide de ces arrangements, nous étudions plusieurs problèmes d'optimisation géométrique dont la sortie est une sélection. Ces problèmes ont pour but de maximiser *l'occupation spatiale* de la sélection, selon différents critères. Par exemple, on peut vouloir trouver s conformers qui maximisent (i) le volume occupé par ces conformers, (ii) l'aire de la surface moléculaire (MSA) de l'union des conformers (voir figure 1.19).

Pour illustrer ces propos, considérons la figure 1.18(i), qui représente 40 conformations de boucles flexibles d'un complexe. Certaines de ces boucles sont comme on peut le voir redondantes, et on voudrait filtrer cet ensemble pour sélectionner un sous-ensemble varié. Une telle sélection, générée par notre algorithme est représentée sur les illustrations 1.18 (ii) et 1.19.

D'un point de vue application, nous replaçons le problème de la sélection de conformations dans le contexte de la prédiction de complexes en utilisant un algorithme d'amarrage avec plusieurs copies de boucles flexibles. Sur les systèmes testés, on observe que l'utilisation des boucles choisies avec notre stratégie peut améliorer de manière significative les résultats de ces algorithmes d'amarrage.

1.3.5 Chapitre 7: Implémentation des Algorithmes

Motivations et travaux antérieurs

Le projet CGAL a officiellement démarré en 1996 sous l'influence d'un consortium de plusieurs équipes de recherche en Europe et en Israël (dont notre groupe à l'INRIA) et a été soutenu par la communauté Européenne durant trois ans. Le format choisi fut celui d'un projet à code ouvert, afin de permettre à d'autres chercheurs de participer. GEOMETRYFACTORY [23], une start-up lancées en janvier 2003, commercialise des licences et développe des outils spécifiques basés sur CGAL.

Durant la même période, le *Computational Geometry Impact Task Force Report*, coordonné par Bernard Chazelle, a donné certaines recommandations [51, 52]. Deux d'entre elle étaient la production de logiciels *utilisables* et utiles, et le besoin de créer une structure qui permettra de mettre en avant les implémentations réalisées dans le monde académique. CGAL s'efforce de suivre ces deux recommandations.

CGAL suit des standards de haute qualité qui sont garanties par différents moyens. Premièrement, un comité éditorial, crée en 2001 et comprenant actuellement 12 membres, est chargé de prendre les décisions techniques et de coordonner la promotion de CGAL. Le comité évalue les propositions de nouveaux paquets, et organise un processus de révision similaire à celui d'un article dans un journal. Ce processus garantie la cohérence, l'homogénéité et la qualité de la bibliothèque, mais aussi récompense les auteurs des paquets

[23] `http://www.geometryfactory.com/`

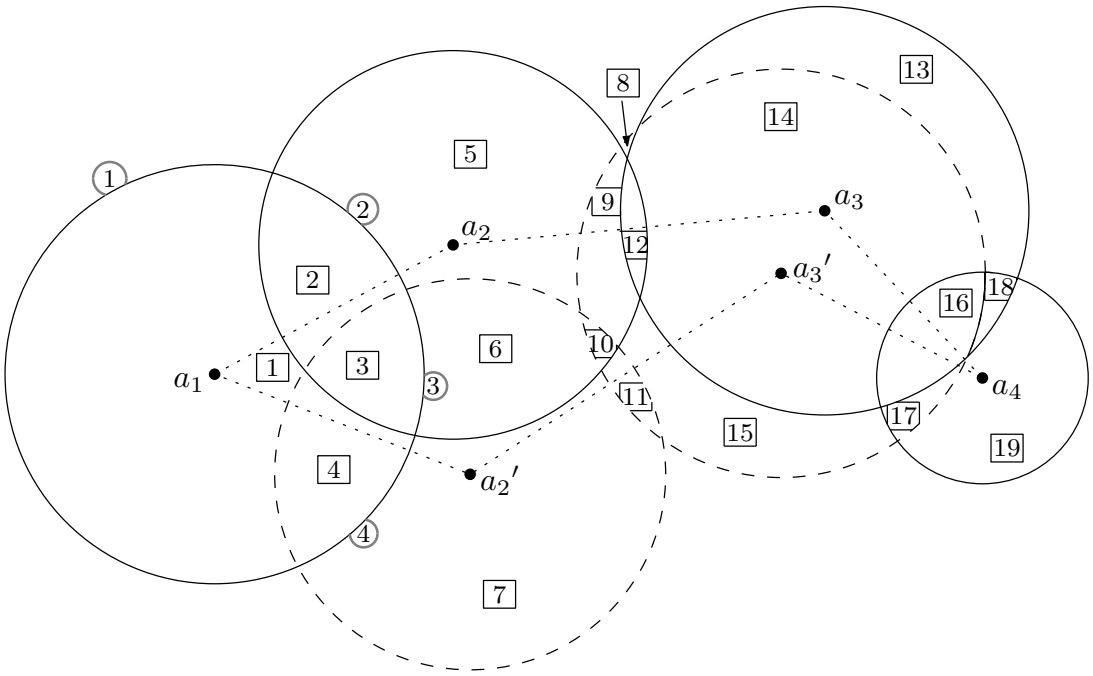


FIG. 1.16 – Exemple de deux conformers 2D, chacun comprenant quatre boules, la première et la quatrième étant communes. Le volume (2D) occupé par deux conformers est décomposé en 19 cellules (numéros encadrés). Les numéros encerclés représentent les morceaux de la surface de la boule centré en a_1, lorsque l'on considère un arrangement en surface de l'ensemble des sphères.

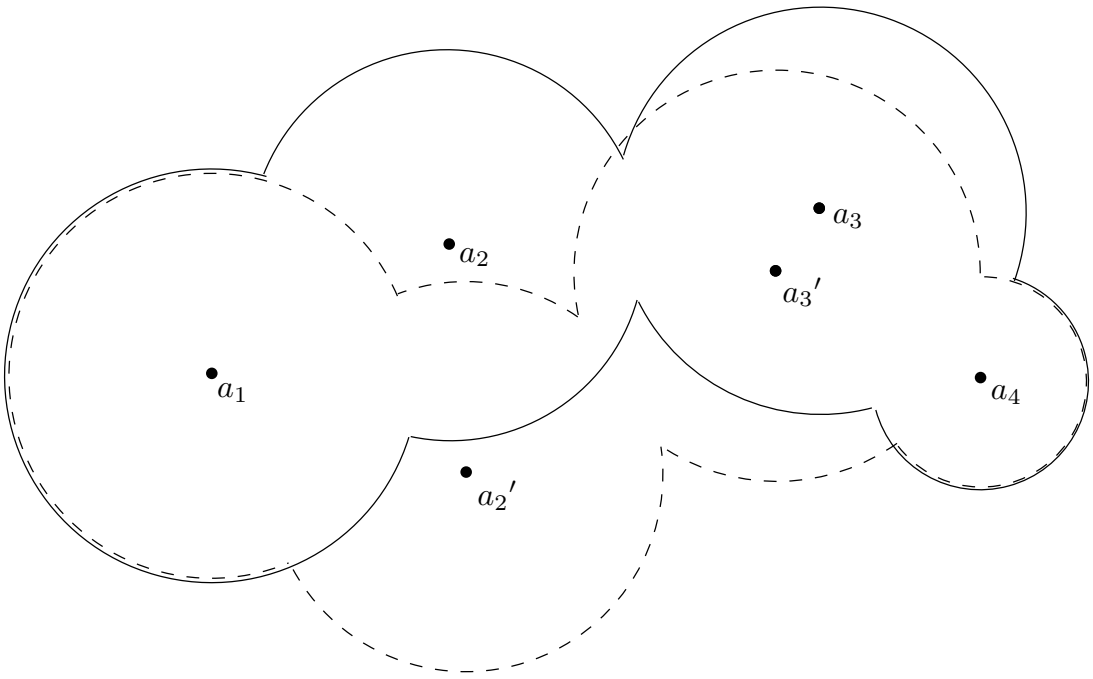


FIG. 1.17 – Le bord de l'union des boules des deux conformers de la figure 1.16, représentés respectivement en pointillés et trait continu.

FIG. 1.18 – Sélection d'un sous-ensemble varié de conformations parmi un ensemble représentant une boucle flexible d'une thrombine, complexé avec un inhibiteur de la trypsine de pancréas de boeuf (code PDB `1BTH`). De gauche à droite : (i) le squelette du récepteur en mode cartoon avec 40 conformers d'une boucle; (ii) le squelette avec 10 conformers sélectionnés par notre algorithme, appelé `Greedy`; (iii) le squelette avec 10 conformers sélectionnés par un algorithme standard de classification hiérarchique, appelé `HClust`.

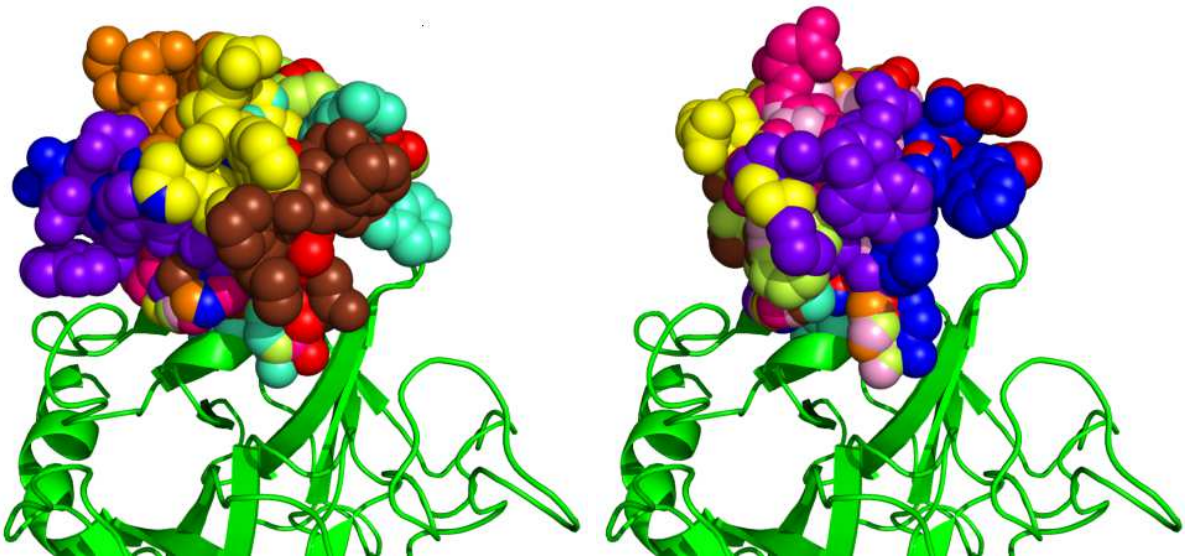


FIG. 1.19 – La suite de la figure 1.18: Représentation avec les boules de Van der Waals des 10 conformers présentés sur les illustrations 1.18 (ii) et 1.18 (iii). On remarque que les conformers du premier ensemble sont bien *séparés*, alors que ceux du second sont entremêlés.

intégrés dans CGAL. Deuxièmement, une documentation détaillée composée d'un guide de l'utilisateur et d'un manuel de références pour chaque paquet est disponible (pour la version 3.3.1 de septembre 2007 3000 pages pour 57 paquets). Troisièmement, des tests sont exécutés chaque nuit sur les versions internes pour éviter l'apparition de bogues.

L'arrangement de cercles sur une sphère ayant des applications en biologie structurale computationelle, nous mentionnons l'existence du projet BALL[24] [122] qui a pour but de fournir un cadre unifié et générique pour développer des logiciels dans ce domaine.

Contributions

Le chapitre 7 donne une ébauche de l'implementation du noyau sphérique CGAL, et détaille les differents niveaux de l'implementation du paquet arrangement de cercles sur une sphère. Nous donnons aussi un exemple qui montre comment le paquet arrangement s'interface avec le noyau sphérique pour calculer la décomposition des surfaces d'un ensemble de sphères. La partie du noyau sphérique non dédiée aux objets sur une sphère de référence et le noyau algébrique seront disponibles dans la prochaine version de CGAL(prévue à la fin de l'année 2008). La partie restante, ainsi que les primitives pour tout les types de cercles pour le nouveau paquet CGAL arrangement sur une surface [25] ont besoin d'être travaillés afin de pouvoir être présentés au comité éditorial.

[24] `http://www.ball-project.org/`

Chapitre 2

Introduction

In living organisms, deoxyribonucleic acid (DNA) is known to store the information required for development. The nucleotide double helix structure of DNA was first proposed by James D. Watson and Francis H. C. Crick in 1953 [192], based on the experimental work of Rosalind Franklin[25]. Among macromolecules responsible for many functions in a living organism, proteins are coded by DNA. The first complete structure of a protein [26] (myoglobin) was determined in 1958 by John Kendrew et al. [116]. They were able to give the first description of the *shape* of a protein, and noticed that: *Perhaps the most remarkable features of the molecule are its complexity and its lack of symmetry. The arrangement seems to be almost totally lacking in the kind of regularities which one instinctively anticipates, and it is more complicated than has been predicated by any theory of protein structure. Though the detailed principles of construction do not yet emerge, we may hope that they will do so at a later stage of the analysis.* Currently, even though huge progress in structural biology experimental techniques [27] has been achieved by thousands of researchers, protein behavior still keeps secrets. We hope that the methods developed in this work will contribute to improve the understanding of biomolecules and their functions in organisms.

2.1 Introduction to Protein Structure

Protein structure is described at four different hierarchical levels: the primary structure, the secondary structure, the tertiary structure and the quaternary structure.

2.1.1 The Primary Structure: Building from a Basis of Twenty Amino-acids

A protein is made of several *polypeptide chains*. Like the first complete protein sequence determined by Frederick Sanger in 1955 [169] (the insulin), each chain of a protein is a succession of amino acids belonging to 20 different physico-chemical types (refer to Fig. 2.2 for an illustration). Each amino acid is composed of a common template (NH_2-C_αH-COOH) and a specific *side chain* bound to the carbon α atom (C_α). The *peptide bond* linking two amino acids is formed from an hydrolysis reaction leading to a covalent bond between an oxygen atom of the first amino acid and the nitrogen atom of the second one.

[25] As she died in 1958, Rosalind Franklin could not be eligible for the 1962 Nobel prize in Medicine, which was awarded to James Watson, Francis Crick and Maurice Wilkins *for their discoveries concerning the molecular structure of nucleic acids and its significance for information transfer in living material.*

[26] Max F. Perutz and John C. Kendrew were awarded the 1962 Nobel Prize in Chemistry, *for their studies of the structures of globular proteins.*

[27] Aaron Klug was awarded the 1982 Nobel Prize in Chemistry, *for his development of crystallographic electron microscopy and his structural elucidation of biologically important nucleic acid-protein complexes.* Kurt Wüthrich was awarded the 2002 Nobel Prize in Chemistry, *for his development of nuclear magnetic resonance spectroscopy for determining the three-dimensional structure of biological macromolecules in solution.*

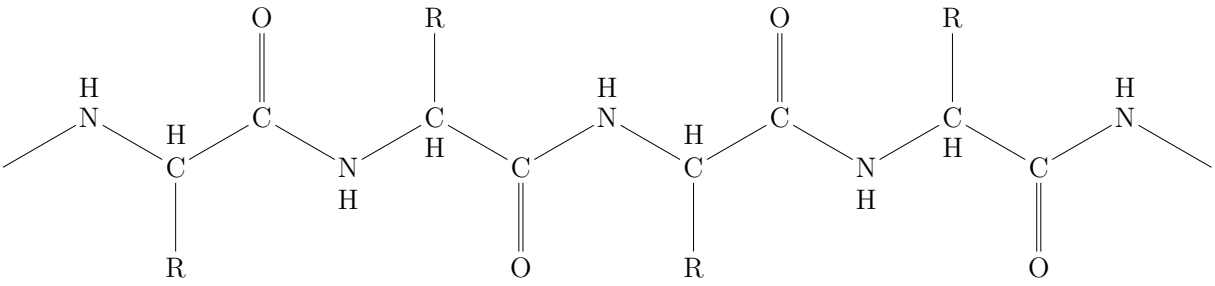


FIG. 2.1 – Part of the backbone of a polypeptide chain of a protein. Full side chains are omitted and represented by R.

The *backbone* of each chain of a protein, starting from the N terminus, and ending at the O terminus, is composed of the repeated template NH-C_αH-CO (See Fig. 2.1). An amino acid in a protein is called a *residue*.

The atoms of the backbone, from the C_α atom of a residue to the C_α atom of the next residue, define a *peptide group*. An interesting geometric property of a peptide group is that its atoms are coplanar. Moreover, within a peptide group, the bond lengths and bond angles are nearly the same in all proteins.

Considering the C_α atom common to two peptide groups, the rotation angles around the N-C_α bond and around the C_α-C bond are called the *torsion angles* ϕ and ψ respectively[28]. These two angles are the two degrees of freedom of each residue that define the 3D structure of each polypeptide chain (Notice that the flexibility of side chains is modeled using the so called χ torsion angles).

2.1.2 The Secondary Structure: Two Essential Patterns of Stability

In proteins, residues of the same chain often adopt a periodic configuration of successive ϕ and ψ angles, leading to the so called secondary structure elements. The first and more common secondary structure elements defined by Linus Pauling et al. in 1951 [155, 156] are the *β-sheet* and the *α-helix*.

A α-helix in a protein consists of a connected set of residues, such that the backbone atoms of these residues are organized as a regular helix with 3.6 residue per turn. This structure is stabilized by hydrogen bonds between oxygen atom of residue n and nitrogen atom of residue $n+4$ (the bond length is about 2.8Å). Side chains are obviously positioned outside the helix (See Fig. 2.3 for an illustration). Note that a α-helix may be *right-handed* or *left-handed*, depending on the screw direction of the chain (the most common in protein is the *right-handed* one).

A β-sheet consists of several sets of connected residues (one such set is called a *β-strand*). β-strands are linked by hydrogen bonds between nitrogen atoms and oxygen atoms of two adjacent strands, as illustrated on Fig. 2.4 and 2.5. Two β-strands can be either parallel (see Fig. 2.4) or anti parallel (see Fig. 2.5) depending on their relative orientation: The β-sheet is said to be anti parallel, when the sequences of the two adjacent strands are read (from N to C terminus) in opposite way, and parallel otherwise.

The organization of each polypeptide chain mainly relies on α-helices and β-sheets, but more complicated structures are also observed (β-bends, Ω-loop,...). These structures impose constrains on the aforementioned torsion angles (ϕ, ψ). For each C_α atom, plotting its ϕ value against its ψ value results in regions dedicated to β-sheet and α-helix, as observed on Fig. 2.6. This is called a Ramachandran plot [162].

2.1.3 The Tertiary Structure: a Global Spatial Level of Organization

Once a polypeptide chain has adopted a local organization into secondary structure elements, then the whole chain folds to form the tertiary structure. A classical experiment has revealed that the folding of a protein can be reversible and relies on chemical interactions [175].

[28] Note for the case of a C_α atom of a proline residue, ϕ is constant

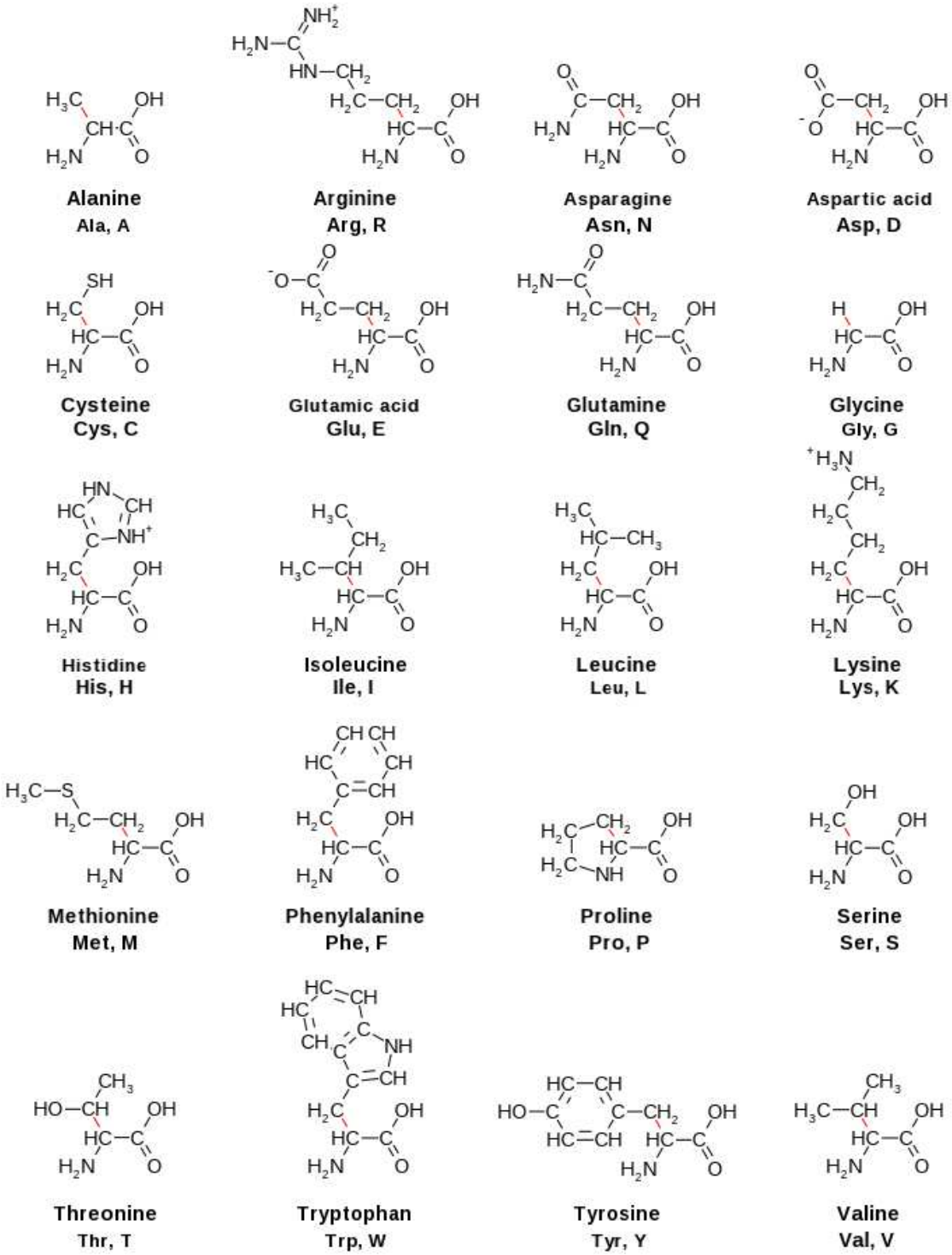


FIG. 2.2 – List of the main amino acids used to build proteins. Each amino acid is associated a one letter code and a three letter code.

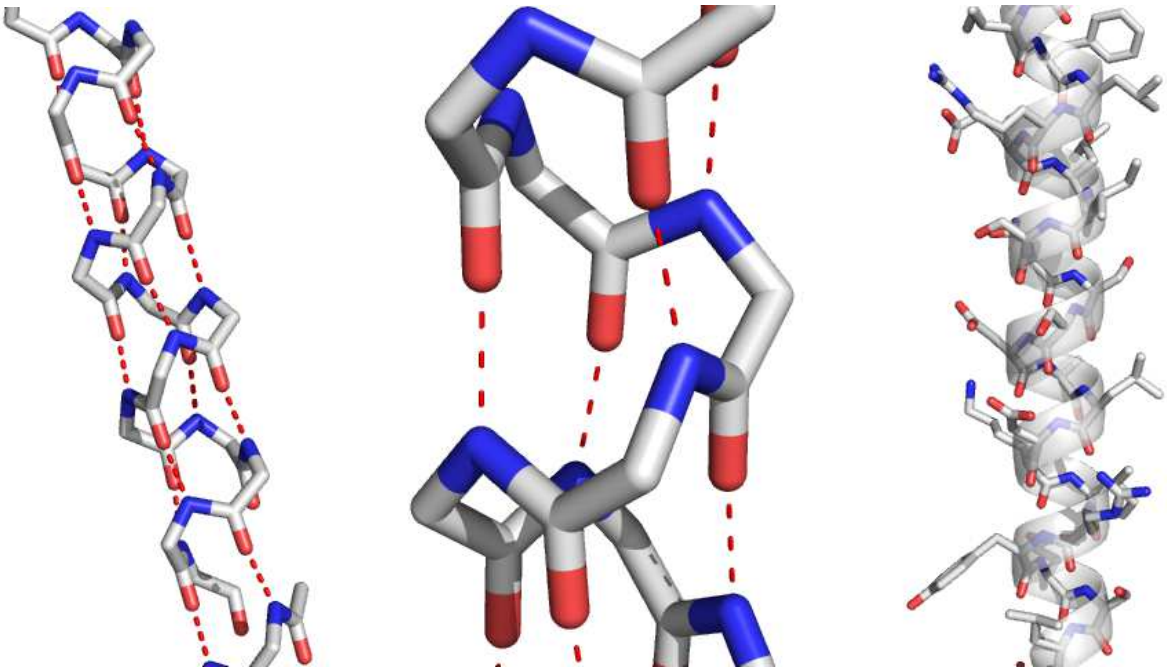


FIG. 2.3 – α-helices in proteins. From left to right: an ideal α-helix of proteins, hydrogen bonds are represented by dashed lines and side chains are omitted; a zoom of the first picture; A α-helix of proteins with its side-chains. Carbon, nitrogen and oxygen atoms are colored white, blue and red respectively. Hydrogen atoms are omitted

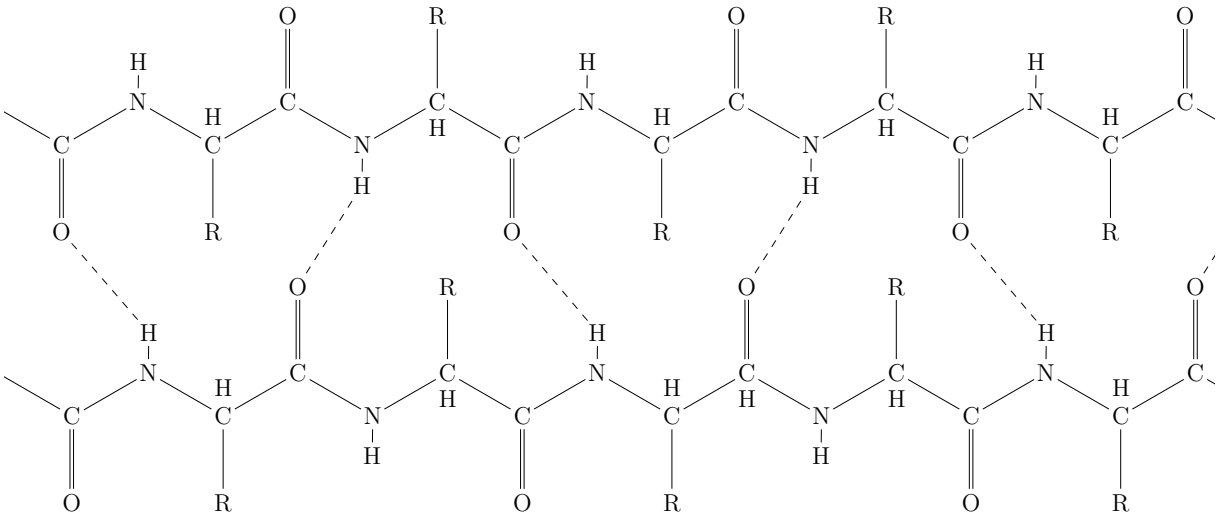


FIG. 2.4 – Parallel β-sheet. Full side chains are omitted and represented by R. Hydrogen bonds are represented by dashed lines.

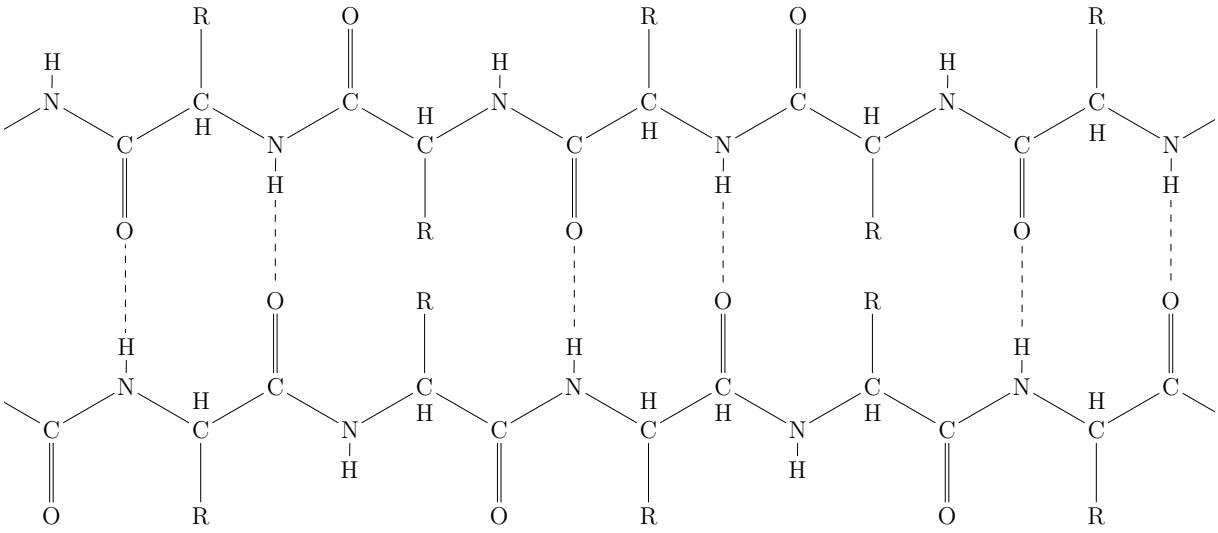


FIG. 2.5 – Anti parallel β-sheet. Full side chains are omitted and represented by R. Hydrogen bonds are represented by dashed lines.

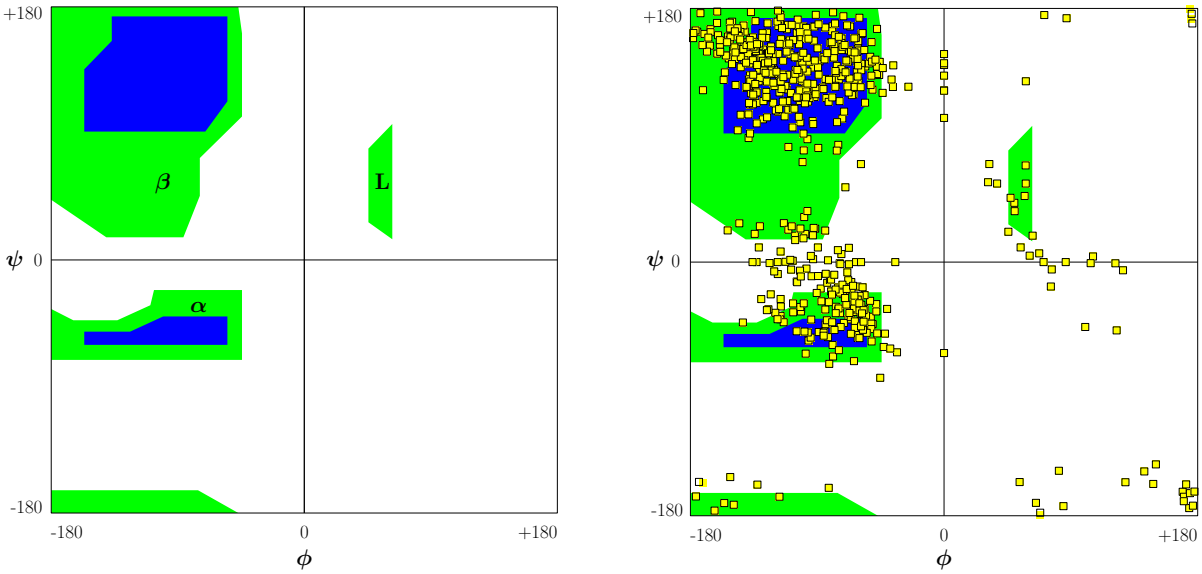


FIG. 2.6 – Ramachandran plots [162]. Left: Green and blue regions represent probable and most probable values of backbone angles (ϕ, ψ) respectively. Angles (ϕ, ψ) for right-handed α-helix, β-sheet and left-handed β-sheet are approximately in the regions denoted $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\mathbf{L}$ respectively. Right: Values of (ϕ, ψ) for the yeast allantoicase (PDB code `1SG3`).

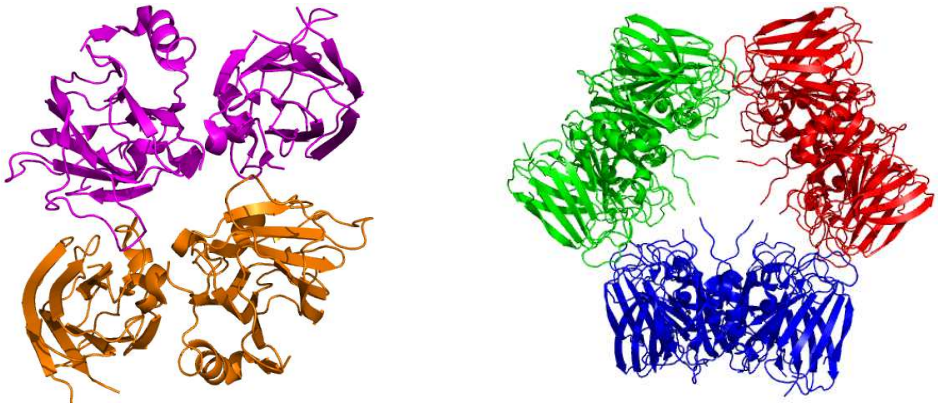


FIG. 2.7 – Symmetries in proteins. Left: The asymmetric unit of the yeast allantoicase crystal (PDB code `1SG3`) is a dimer (each subunit has its own color). Right: The biological unit is a trimer of dimers (each dimer has its own color). The D_3 transformation is used to reconstruct the biological unit from the protomer.

The tertiary structure is essential for the function of proteins. As observed in [175], a bovine pancreatic ribonuclease protein (a digestive enzyme) has no activity when unfolded (this was obtained by increasing temperature). When the protein folds again, its activity can restart. *The lack of regularity* of the tertiary structure of a protein is actually really important for the diversity of the interactions, and thus functions of proteins. Upon formation of the tertiary structure, some regular *domains* (patterns of secondary structure) are often observed in proteins [130] (α-domains, β-domain, $(\alpha+\beta)$-domain and (α/β)-domain). More details can be found in [33]. Kendrew et al., after having successfully determined the high resolution structure of myoglobin [117] in 1960, observed that hydrophobic side chains tend to be more in the core of the protein, whereas hydrophilic ones tend to be on its boundary [118]. This property, together with the accompanying entropic loss upon folding, is known as the *hydrophobic effect* (this is the *hydrophobic collapse* model), and is supposed to be a key parameter in the folding process [74].

2.1.4 The Quaternary Structure: a Functional Assembly

The last level of protein structure consists in assembling several folded polypeptide chains, using mainly non-covalent bonds. Each such chain is called a *subunit*. A protein composed of more than one subunit is called an *oligomer*.

Any two subunits of a protein may be identical or different. The *protomer* of a given protein is the smallest subset of different subunits used to build the protein (if all the subunits are different, the protomer is the protein itself). As an example, illustrated on Fig. 2.7, the yeast allantoicase is an hexamer.

The subunits are usually symmetrically arranged, using exclusively axial rotations. A C_n transformation involves n copies of the protomer, by successive rotations of $\frac{2\pi}{n}$ around a given axis. A D_n transformation involves $2n$ copies of the protomer, by axial rotation of π, followed by successive rotations of $\frac{2\pi}{n}$ around an axis perpendicular to the first one. Higher order transformations involve rotations around symmetry axis of a convex regular polyhedra.

Several reasons have been proposed to explain the existence of this quaternary structure, all related to the size of proteins [190]. First, the folding process should be made easier (following the classical divide-and-conquer paradigm). Second, for maintenance reasons, it is more convenient for an organism to replace only deficient subunits of a large protein rather than the whole assembly. Finally, the fact that symmetry is observed allows a protein to have a symmetric behavior (duplication of active sites,...).

2.1.5 Experimental Structure Determination Techniques

X-ray crystallography The principle of this method is to use the diffraction of a X-ray beam by a protein crystal to obtain an electron density map of the structure.

The first limitation of this approach comes from the crystal growing. Its must be sufficiently large, without defect and *pure*. Due to the globularity of proteins and their physico-chemical properties, water molecules account for about 50 percent of the crystal volume. The crystallization depends on several parameters (ionic concentration, temperature, pressure, ...), and getting a large enough crystal may take weeks. Once the crystal is obtained, it is often rapidly frozen so as to limit displacement in the crystal and to improve the resistance of the crystal to the X-rays. Suitable X-rays are usually produced by synchrotron radiations. The quality of the crystal (and thus of the electron density map) is determined by the quality of the diffraction. The position of each atom is estimated from the electron density map to match the corresponding densities. A 3D structure is considered to be at high resolution when its coordinates are correct up to a less than 2 Å displacement. The structure resulting from those experiments is not necessarily the protein itself, but the *asymmetric unit* of the crystal (the smallest entity which is repeated to build the crystal).

Such an experimental method is suitable to determine the structure of a protein of up to 500 amino acids.

Nuclear Magnetic Resonance spectroscopy (NMR) The main idea of this method is to use the magnetic moment of some atomic nuclei (called the *spin*). One puts a protein inside a strong static magnetic field, and exposes it to another oscillating magnetic field. This induces a change in the nucleic spin, and when the second field is deactivated, each atom nuclei by returning to its previous state emits radiations that can be measured. The frequency of the radiation depends on both the atom type and the molecular environment of the nucleus. The coordinates of a protein structure is mainly obtained by studying the radiation of its hydrogen atoms induced by the spin change. Varying the parameters of the oscillating magnetic field, one can estimate (i) the distance between hydrogen atoms that are covalently linked to either the same atom or two covalently connected atoms, and (ii) the distance between two hydrogen atoms that are close, but not of the previous type.

The output of this method is a list of distance restraints. Therefore, several structures may satisfy the distance restraints and several models are usually proposed.

An advantage of this method, is that the proteins are in solution. One can adjust the composition of the solution, as well as temperature, pressure and pH to observe the behavior of the protein in different environment (physiologically equivalent or not). Nevertheless, depending on how these parameters are adjusted, and on the protein structure, the signal may be hard or even impossible to interpret. Moreover, for some experiments, carbon, nitrogen or hydrogen atoms have to be mutated to a radioactive isotope.

This method is suitable to obtain the structure of a protein of up to 250 amino acids.

Cryo Electron Microscopy (Cryo-EM) This method relies on the direct observation of proteins using a powerful microscope. Given a molecule in its natural environment, the sample is often rapidly frozen using liquid nitrogen (or ethane) and then observed using an electron microscope. The main difference between a classical microscope is that the molecule is not *illuminated* by a visible light but bombarded by electrons (the resolution is better because of the smaller wavelength of electrons), the beam being directed by an electrostatic or electromagnetic *lens* (an annulus). The output is a density electron map, inferred by elastic and inelastic (producing noise) bounces with atoms of the sample. When possible (the electron beam tends to damage the sample), images of a sample tilt series are used to ease the reconstruction step. Three main approaches are used to reconstruct the structure of a molecule, from higher to smaller resolution: (i)In *electron crystallography*, the sample is organized as a 2D crystal (of much smaller size than those needed in X-ray crystallography). The reconstruction uses the diffraction principle as in X-ray crystallography, together with the symmetries of the crystal network. Best resolution is about 5Å. (ii)In *single particle reconstruction*, the sample contains several copies of the same molecule. The reconstruction is achieved by using the signal from several copies, averaging the molecule structure. Best resolution is about 10Å. (iii) In *electron tomography*, the sample used represents a single entity of a molecule. This precludes using other copies to improve the signal-to-noise ratio. The principle is to increase the number of views by incrementally rotating the sample around an axis perpendicular to the electron beam. Best resolution is about 20Å.

This method is suitable to determine larger protein assembly structures of at least 300 amino acids.

2.1.6 Data Bases of Protein Structures

The first data base available was founded in 1971 at the *Brookhaven National Laboratory* (USA), and named Protein Data Bank. In 1999, the data base was transferred to members of the *Research Collaboratory for Structural Bioinformatics*[29] (RCSB). In 2003, RCSB together with the *Macromolecular Structure Database at the EMBL's European Bioinformatics Institute*[30] (PDBe) and *Protein Data Bank Japan*[31] (PDBj) worked in collaboration to create the *Worldwide Protein Data Bank*[32] (wwPDB) [27, 28], an international Protein Data Bank Archive of macromolecular structural data that is freely and publicly available to the global community. In 2006, the *Biological Magnetic Resonance Data Bank*[33] (BRMB) group joined the project. This collaborative archive is providing the results of each group and has a standardization role. Each file in the data base is provided in at least a common file type (PDB format). They guarantee the overall quality and consistency of the files in the data bases, and also avoid duplicated entries. The consistency of the file type is especially important for software developers, but also for its own evolution (decisions will be followed by the whole community). As of September 2008, 42026 (respectively 6506 and 134) protein structures determined by X-ray (respectively NMR and Electron Microscopy) were available.

The Structural Classification of Proteins[34] (SCOP) data base [148, 13] provides a classification of proteins, according to their structure and their evolutionary relationships. All protein structures available in wwPDB are taken into account. A similar data base provides a hierarchical protein structure classification, using the four levels Class, Architecture, Topology and Homologous superfamily[35] (CATH) [154, 104].

One problem related to X-ray crystallography, is that the output reported is the asymmetric unit of the crystal. The biological unit may not be directly represented. The asymmetric unit may contain several copies of the biological unit, or several crystallographic symmetry operations may be needed to reconstruct the biological unit. Even if the transformation matrix is given in the input file (for PDB files fields labeled REMARK 350), the *Protein Quaternary Structure server*[36] (PQS) [109] provides for each entry of the wwPDB obtained by X-ray crystallography, the coordinates of a computed probable quaternary structure.

2.2 Some Challenges in Computational Structural Biology of Proteins

In this section, we present questions raised in *computational structural biology* (CSB) which combines mathematics and computer science with structural biology. Selected topics presented here, are related to the motivations of the on-going and achieved work. We start by introducing notions related to the study of proteins. Then we introduce the folding problem, and we finish with the docking problem.

2.2.1 Prerequisite: Molecular Models Used in Computational Structural Biology

An atom is composed of a nucleus surrounded by an electron cloud. Ideally, the physics of a protein should be best described by quantum mechanics. However, due to the computational overhead [37], the description of proteins usually resorts to Newtonian physics. In the following, we present selected geometric quantities to manipulate proteins in this model.

[29] http://www.rcsb.org/pdb
[30] http://www.ebi.ac.uk/msd/
[31] http://www.pdbj.org/
[32] http://www.wwpdb.org/
[33] http://www.bmrb.wisc.edu/
[34] http://scop.mrc-lmb.cam.ac.uk/scop/
[35] http://www.cathdb.info
[36] http://pqs.ebi.ac.uk
[37] Currently only small systems (up to 50 atoms) can be handled.

Atom type	radius in Å
Oxygen	1.40
Trigonal nitrogen	1.65
Tetrahedral nitrogen	1.50
Tetrahedral carbon	1.87
Trigonal carbon	1.76
Sulphur	1.85
Water	1.40

TAB. 2.1 – Extended radii for VdW model of proteins, extracted from [57].

Root mean square deviation A protein structure given in silico is mainly described by one point per atom (or one set of positions per atom in case of an NMR experiment). One measure often used to assess the deviation of pairs of structures of the same protein sequence is the *root mean square deviation* (RMSD). If a protein with n atoms is considered as a point in $\mathbb{R}^{3n}$, the RMSD between two proteins is the Euclidean distance between the two proteins divided by $\sqrt{n}$. Usually, the RMSD is given after a minimization step between two structures using rigid transformations to bring one close to the other. The value reported is denoted the *least-RMSD*. If the computation of the RMSD is restricted to the carbon alpha atoms, it is denoted C_α-RMSD.

Union of balls To ease both the simulation and the visualization of proteins, an atom is usually represented as a solid ball, whose radius depends on the nature of the atom but also on its chemical environment: This is the *Van der Waals* (VdW) model. Since position of an hydrogen atom is hard to evaluate, depending on the application, they are often discarded. Sets of extended radii for heavy atoms (i.e. non-hydrogen atoms) are available to take into account this second approximation. In the following, an *atomic ball/sphere* should be understood as the ball/sphere corresponding to the atom in the VdW model. For more information on radii used, an overall view is provided in [100]. We give in Table 2.1, a set of radii [57] extracted from packing in amino acid crystal structures.

In coarse grained representations of proteins, each residue is approximated by a reduced number of balls—with larger radii than atoms [128]. This model presents an overall view of protein structure (fuzzy enough to ignore some levels of side chain flexibility), and is also useful to reduce the number of primitives to handle.

Surfaces and volumes Using the VdW model, several surfaces (and thus volumes bounded by these surfaces) have been defined. The *VdW surface* of a molecule is defined as the boundary of the union of its atomic balls. A water molecule is represented by a solid ball (called a *solvent ball*), representing its oxygen atom in the VdW model—its radius is usually 1.4Å. The *Solvent Accessible Surface* (SAS) of a molecule, is defined as the boundary of the union of its atomic balls whose radii have been increased of the radius of a solvent ball. This is the surface that bounds the volume in which the center of a solvent ball can be placed, such that the solvent ball intersects at least one atomic ball of the molecule. The *Connolly surface* (also called *molecular surface*) of a molecule, is defined as the boundary of the volume which is not accessible to a water molecule oxygen atom (i.e. the volume that a solvent ball can not cover, without covering a part of an atomic ball of the molecule). One way to construct this surface, is to consider the surface traced by the inward-facing surface of the solvent ball when rolling on the VdW surface of the molecule.

These definitions are illustrated on Fig. 2.8.

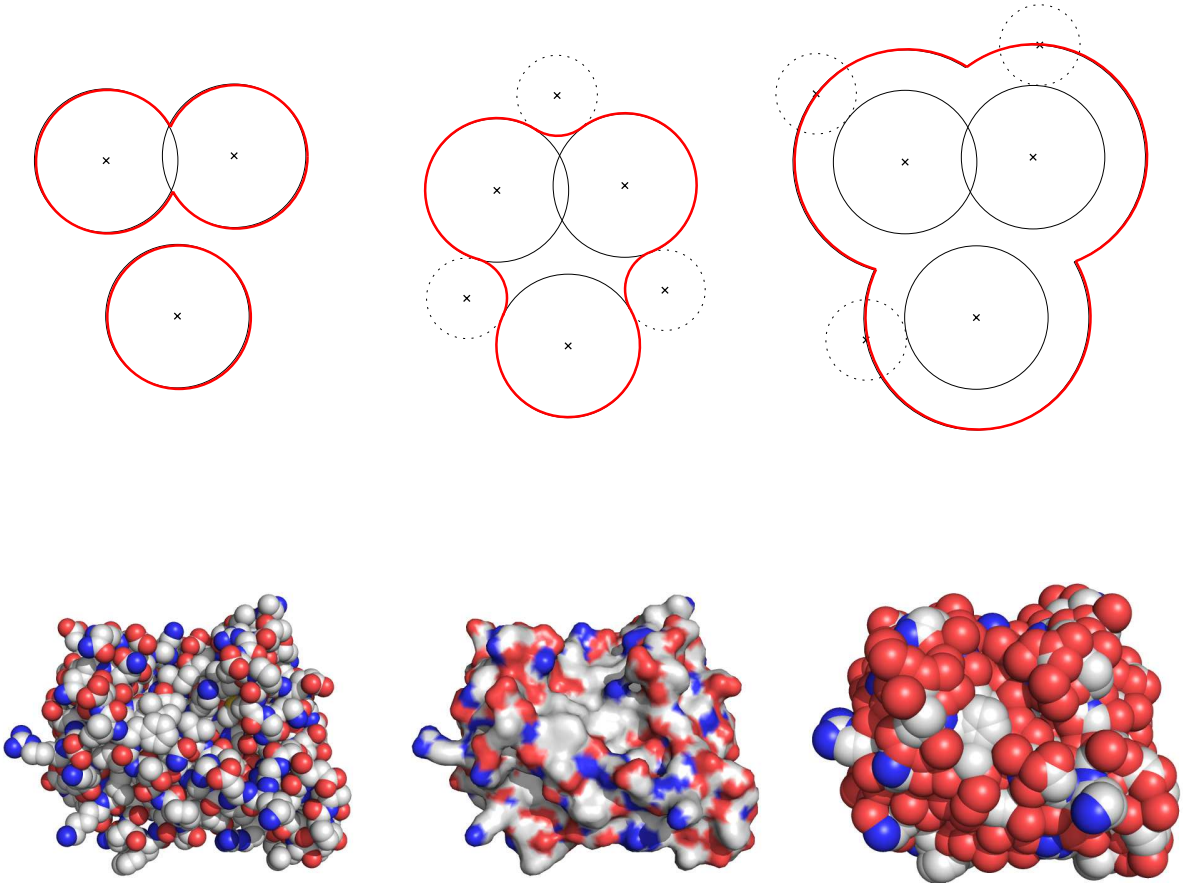


FIG. 2.8 – First line: 2D illustrations of molecule surfaces. Atoms are depicted as solid black circles, water molecules as dotted circles, while the different surfaces are represented in red. Second line: 3D illustrations of molecule surfaces. Carbon, nitrogen and oxygen atoms are colored white, blue and red respectively. From left to right: The Van der Waals surface, the Connolly surface and the Solvent Accessible Surface.

2.2.2 Prerequisite: Classical Geometric Quantities on Proteins

Biophysics

Exposed surfaces Using the Van der Waals model, Lee and Richards [126] introduced the solvent accessible surface (SAS). The definition can be restricted to an atom or a group of atoms (side chains, residues,...). The goal was to study the problem of folding. Change in accessibility of a residue from an unfolded to a folded state was measured. The value for the unfolded state was computed from a configuration where the residue is squeezed between two Alanine. The hypothesis that hydrophobic collapse was dominant in driving the folding was investigated by Chothia [56]. He observed a linear relationship between the area of the solvent accessible surface ($SASA$) of amino acids and their (measured) free energy of transfer from water to organic solvent. This relation has been later observed in [188] and [87] for example. In 1986, Eisenberg and McLachlan [83] gave an estimate of the contribution of each protein atom to the solvation free energy using the accessibility of the atom to the solvent. They used it to estimate protein stability in water. Vallone et al. [188] experimentally observed a good agreement of their measure with this estimated energy. For a complete review on molecular surface (definition, applications and algorithms) up to 1996 refer to [63].

Contact surfaces The modeling of geometric contacts within molecular models has been studied at different levels (between proteins, residues or atoms). The buried surface area (BSA) introduced by Chothia and Janin [59] for a protein-protein complex is defined as the loss of accessible surface area upon formation of the complex: $BSA(A \cup B) = SASA(A) + SASA(B) - SASA(A \cup B)$. Notice that this formula assumes the molecules undergo a rigid motion—no deformation. They used it to study the stability of three protein-protein complexes using the hydrophobic contributions derived from the loss of accessible surface in water. The BSA has shown to be a parameter of major interest in structural studies of interfaces in macro-molecular complexes [45, 160]. Abagyan and Totrov [1] introduced several measures at different scales to evaluate the differences between two structures of the same proteins. They were based on a matrix of contact areas between each pair of residues. Note that to compute these coefficients, they first computed the $SASA$ of all residues taken separately. Then given a residue i, for each residue j, they measured the loss of $SASA$ of i induced by j. The resulting matrix was not symmetric as the area lost by i was not the same than the one lost by j. A way proposed to make this matrix symmetric was to take the mean of $SASA$ lost. They showed improved discrimination results w.r.t. classical RMSDs, and also provided a nice graphical display of contact preservation. Another definition of contact surfaces was used for modeling a binding site of the Quinone-B [180]. The contact surface of atom a with atom b was defined as the part of the surface of a included into b and *closest* to b (which is equivalent to consider the intersection of the surface of a with cell of b in the power diagram associated to all atoms). Mc Conkey et al. [141] gave a new method for computing these contact surface coefficients, based on the tessellation induced by the power diagram. They used these coefficients to determine a knowledge-based discrimination function using non-covalent interactions for native protein structure detection [142]. They also used it together with a surface complementarity criterion for defining a scoring function to predict side-chain conformations [87]. Atom contact area (ACA) was defined as follows for a pair of atoms [70]: For two atoms a and b, it is defined as the $SASA$ lost when adding b to a and its covalent neighbors plus the same quantity exchanging the role played by a and b. It was shown that ACA is related to Van der Waals and electrostatic interactions.

Volume Several studies involving the molecular volumes have been carried out. As testified by the survey of Poupon [159], atomic volumes have been extensively used, among others applications, to report on the packing of molecules. For example, Gerstein and Chothia observe that the core of proteins is more compact that the part at the interface with water [99]. Many of these studies are on the volume occupied by a special polyhedron associated to each atom. More precisely, two choices were usually made: (i)the volume associated to one atom is defined as the volume of its cell in the Euclidean Voronoï diagram [17] of the centers of the atoms; or (ii)the volume associated to one atom is defined as the volume of its cell in the power diagram [16] of the centers of the atoms with weight corresponding to the squared Van der Waals radii. An example of a cell in the Voronoï diagram of the centroids of the side chains is presented

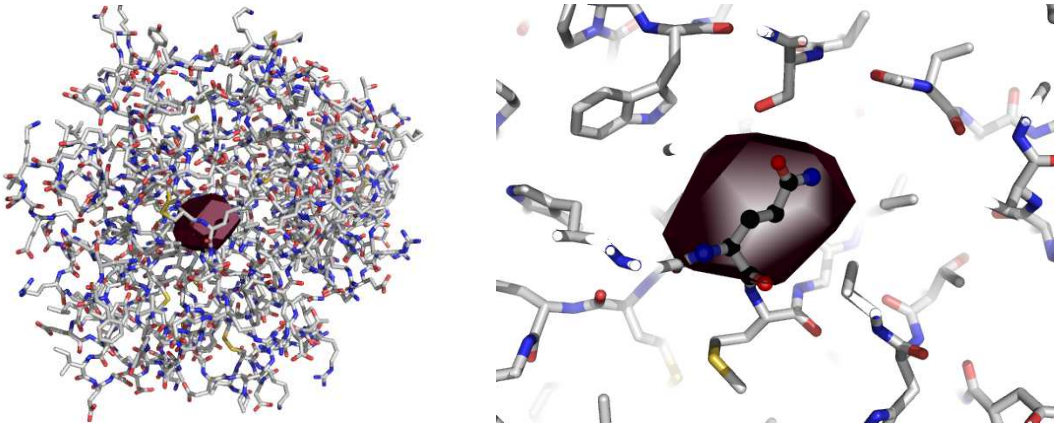


FIG. 2.9 – Voronoï cell of a glutamine (residue number 209) in a one point per residue representation of the Thrombin `E192Q-BPTI` (PDB code `1BTH`). The protein is represented with sticks. Left: The Voronoï cell in the protein; Right: The glutamine within its Voronoï cell. (Courtesy of Julie Bernauer [29])

on Fig. 2.9.

These approaches raise several issues. In particular, an atom on the *boundary* of a model is associated an infinite cell. To avoid such a problem, fictitious water molecules have been added around the model (either on a regular grid or placing a water molecule tangent to three boundary atoms). Another solution, when using the power diagram, is to define the volume associated to each atom as the volume of its cell in the diagram but restricted to its Van der Waals ball. To finish, we mention that the solvation free energy of a solute in water has been estimated using the volume of the union of atomic balls and many-body interactions terms [111].

Algorithms

Area We give a selection of algorithms and put the emphasis on the underlying geometry. Programs accompanying a method (if any) are cited in parenthesis. Methods used may be split into those resorting to analytical formulae, and those resorting to approximations. The latter category uses a discretization of the ambient space (a grid) or of the atomic spheres (sampled with points) to provide estimates (the quality of the approximation depends on the density of the sampling). Considering methods of the former type, whatever the surface considered (Van der Waals or solvent accessible surface), the area is computed using the area of spherical caps on each atomic sphere that contribute to the boundary of the union of atomic balls.

We consider two classes of algorithms for computing the area of a union of balls.

The first class of algorithms is based on properties of power diagrams [16]. In this paragraph, a cell should be understood as a cell in the power diagram of the weighted points corresponding to atoms. The contribution of any ball to the boundary of the union lies in its cell—parts outside this region are covered by other atomic balls. A natural strategy consists in computing the intersection of each atomic sphere with its cell in the power diagram. This step requires a list of neighbors. These neighbors can be retrieved for example from a grid [189, 141]. Following the non-naive way to construct a cell using polarity, selected redundant neighbors can be filtered out [95] (GETAREA). Another strategy relies on properties of the dual complex [78]. Atomic spheres contributing to the boundary directly reads from the dual complex [7] (See Fig. 2.10). Alternatively, using space partitioning data structures, the contribution of the 0-complex to the boundary of the union can be constructed by a greedy exploration of contacts between pairs and triples of balls [170] (MSMS).

The second class of algorithms focuses on the intersection circles found on a given atom sphere, so as to construct the loops made of circular arcs bounding the convex spherical patches contributed by the atomic sphere to the surface. Example algorithms incrementally maintaining the relevant loops are [185], [186] (SurfRace, FastSurf). Finally, in [106], the boundary of the union is retrieved through exposed faces

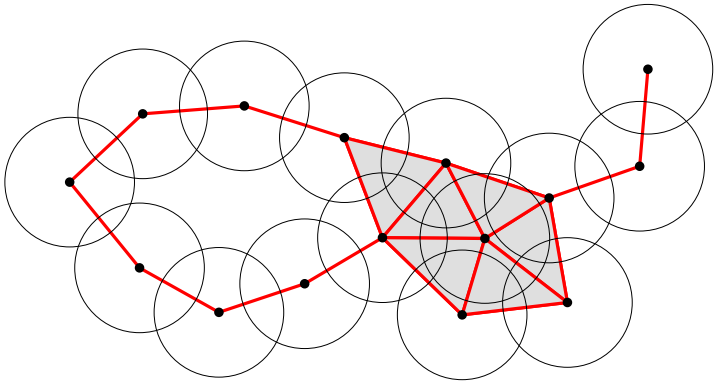


FIG. 2.10 – The dual complex of a union of disks, known as the α-complex. The dual complex is included in the union of disks, and a cavity of the union of disks is included in a cavity of the dual complex. Disks contributing to the boundary of the union of disks, are read in the dual complex considering non-*interior* simplices [78].

in the arrangement of intersection circles on each atomic sphere.

Volume Different methods have been developed to compute the volume of a whole molecule. Connolly [62] gave a method to compute the volume bounded by the so-called Connolly surface using a decomposition of the volume into disjoint pieces and an analytical formula for each of them. Gibson and Scheraga relied on the inclusion-exclusion formula to compute volume of a whole molecule as a union of balls [101]. Edelsbrunner [79] further improved the reduction of the inclusion-exclusion formula, stating that the volume of the union of all the balls of a set B is given by

$$\mathrm{vol}(\cup B) = \sum_{X \subseteq B} -1^{\mathrm{card}(X)-1} \mathrm{vol}(\cap X)$$

where X is defined for each simplex in the dual complex of B, as the balls corresponding to the vertices of the simplex. Edelsbrunner and Attali [15] later proved that the formula remains correct for any abstract simplicial complex whose simplices are an independent set of balls and whose canonical geometric realization has the same boundary and same space as the dual complex. They also proved that all minimal inclusion-exclusion formula are encoded in such an abstract simplicial complex. In particular, this implies that formulae to compute the volume of the intersection of at most four spheres is sufficient to obtain the volume of the union. An alternative using the tessellation induced by the power diagram associated to a molecule, developed by Mc Conkey et al. [141], was based on volume formulae of cones and polyhedra to compute the volume of the restriction of each atomic ball to its cell.

2.2.3 The Folding Problem

Understanding and being able to predict how each chain of a protein folds to acquire its tertiary structure, is a main issue referred as folding. George D. Rose described the problem [165] as follows: *For a protein molecule, function follows form. Unreeled, it is just a long string of amino acids. But wad it up just the right way, into a bundle known as its native conformation, and it becomes the stuff of life.* The structure of some proteins being yet impossible to obtain experimentally, coming up with reliable and efficient folding algorithm is a challenge (to find the structure of unknown proteins and to manufacture new proteins).

The *Critical Assessment of Techniques for Protein Structure Prediction* (CASP) is a world-wide experiment held biannually since 1994, that gives the opportunity for researchers to assess the quality of their structure prediction methods in a blind setting. Unpublished tertiary structures are submitted to participants, who upload on a server a limited number of their best predictions. Results are then published and give the state of the art of software and methods available (the last meeting held was CASP7 [145]).

The methods used can be decomposed into two classes, those relying only on the protein sequence to determine its structure (called *de novo* protein modeling), and those that use data bases of folded proteins to identify similarities in addition to the protein sequence (called *Homology modeling*). In the latter class two main sub-classes of methods are observed: (i) *Threading* consists in using information encoded in homologous protein sequences whose structures have already been solved. Using sequence alignment methods, the best matching is used to copy the folding. The predicted structure may contains *gaps* (i.e. missing residues) when parts of the sequence are too different. This method is based on the hypothesis that there is restricted number of fold scheme in Nature [58]. (ii) *Segment matching* consists in splitting the protein sequence into small segments and matching each segment in a library of fragments of folded protein. This is again based on the hypothesis that the same sequence leads to the same structure.

In the following, we focus on the first class of method, de novo protein modeling.

De novo protein modeling and sampling The principle of *de novo* protein modeling is to resort to in silico simulation (usually using an energy minimization) only to predict the protein structures. This method has been proved to be successful on small proteins.

Several approaches have been developed to predict protein structures. Monte Carlo methods use a random sampling of the movement directions. An interaction score is given using a potential energy of atomic interactions. A movement is accepted when the interaction score is below a given threshold. Molecular dynamics uses a potential function to integrate Newton's law of motion. These methods subsumes the definition of potential functions that are good approximations of the reality. Such a potential energy (ENCAD [131]) is illustrated on Fig. 2.11.

Started in October 2001, the Folding@home[38] project [178], proposed a framework to build the most powerful shared computer for molecular dynamics simulations. Based on the principle of distributed computations, they provided a computer screen saver (a client) that people can install on their personal computer to execute simulations. Each client is computing small jobs, and communicates the results to main central servers. The algorithm supposed that the folding problem follow a Markov state model: when the system dynamics crosses an energy barrier, it leads to the next state. All the clients perform an extensive exploration of the possible configurations, by running molecular dynamics from an identical state, but with different parameters. When one client informs the server that it crossed an energy barrier (the energy of the system is below a given threshold), then a message from the server is sent to all clients so as to stop their computation and to restart from the new state.

With this huge computation power[39], complete folding simulation for small proteins is possible in few days. However, some restriction due to the architecture (no fast shared memory between clients) may be a limitation for more general simulations.

Structure scoring and refinement We now address two particular problems related to protein folding: (i)Consider a set of conformations corresponding to different 3D structures of the same protein (called *decoys*). The experimentally determined fold (called the *native* fold), is supposed to be unknown. The scoring problem consists of extracting conformations close from the native fold, and of rating their similarity with respect to it. (ii)Given a conformation structure *close* from the native fold, the refinement problem consists of refining it so as to make it *even more similar* to the native fold (similarity refers to the RMSD w.r.t. the native structure).

Most of the methods solving these problems rely on energy functions (that can often be used in molecular dynamics), that are of two different types: those that are derived from physical approximations (called *physics-based*), and those that are derived from statistics and properties of high-resolution protein structures found in data bases (called *statistics-based* or *knowledge-based*). A broad literature dealing with these problems is available and the reader is referred to [179, 150, 136] for more details. Introductions of papers on the subject also provide useful insight [191, 177]. However, we describe some papers of interest, that sketch a path for possible improvements that geometry can lead to.

[38] `folding.stanford.edu`

[39] In September 2008, more that 320,000 active CPUs where available, which includes computer CPUs (any OS), graphic card CPUs, and even Playstation3 CPUs.

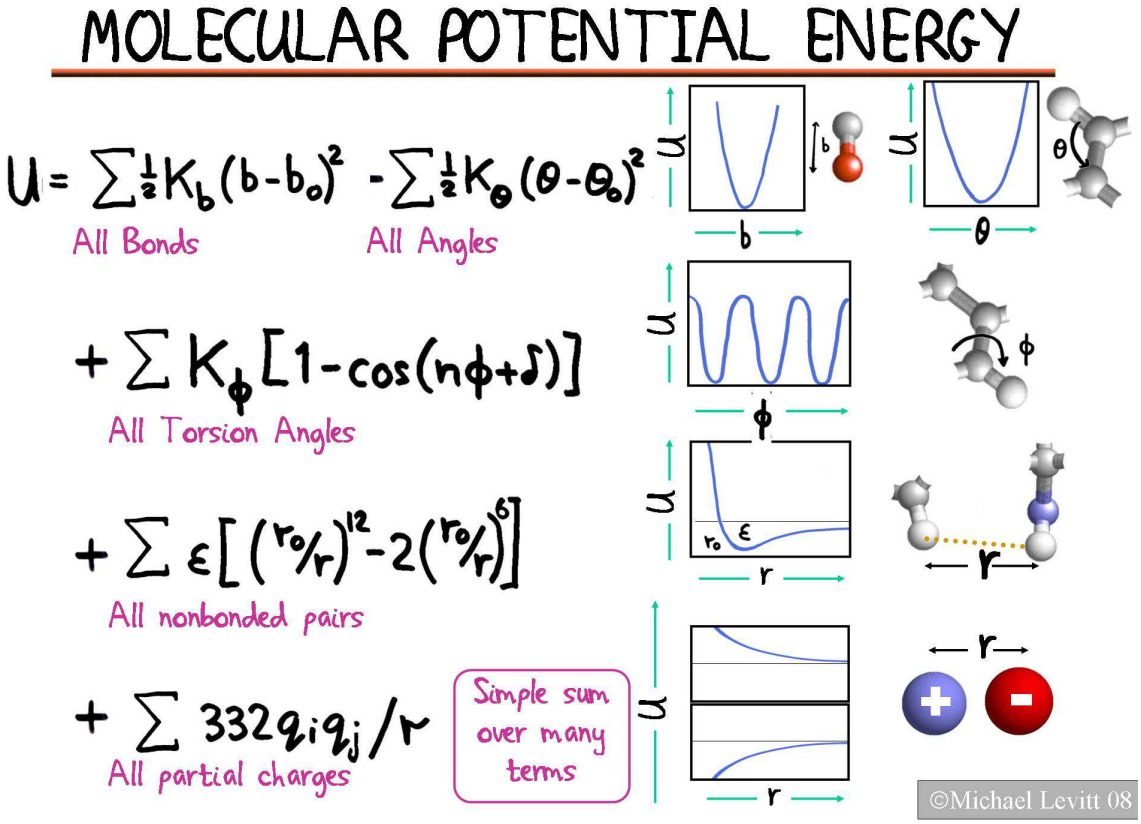


FIG. 2.11 – The total potential energy of any molecule is the sum of terms allowing the bond stretching, bond angle bending, bond twisting, Van der Waals interactions and electrostatics. Many properties of biomolecules can be simulated with such empirical energy function. (Courtesy of Michael Levitt [129])

Extracting a knowledge-based (KB) potential from a high resolution protein data base, a protein being represented as a coarse grain model, has been investigated by Fogolari et al., using an adaptation of the notion of bonded interactions (adapted to the coarse grain model) [94]. Corroborating the efficiency of KB potential, their KB potential was able to find the native and near-native structures in almost all the cases presented. The quality of the data base used to derive KB potential is obviously important, as observed by Liu et al. [138].

Using an energy minimization in vacuo, the efficiency of different physics-based potentials for structure refinement has been assessed by Summa and Levitt [182]. In addition, Summa and Levitt proposed a potential mixing the ENCAD potential [131], replacing the non-bonded terms by a KB potential based on residue-specific pair-wise distances [168]. This approach lead to good results when compared to physics-based potential energy expressions freely available in GROMACS [137].

While many articles rely on geometric properties (mostly pair-wise distances) of protein to derive a KB potential, few addressed the problem of using the notion of neighborhood.

For each triple of residues, the radius of the circumscribed circle to the residues, using a one point per residue representation, was considered by Ngan et al. to derive a KB residue-specific potential [151]. In another approach, considering dual complex[40] of a all heavy atom representation, Li and Liang used the observed proportions of atom-specific triples of atoms corresponding to a 2-simplex in the dual complex [135] to define their KB potential. A KB function was developed by Krishnamoorthy et al., using distributions of Delaunay tetrahedra of a one point per residue representation of protein [123]. Tetrahedra were classified using the four residue types on the one hand, and the adjacency of residues along the backbone within a tetrahedron on the other hand. To take into account stability problems using Delaunay-based KB potential, this approach was later extended by Bandyopadhyay et al. [21], using almost-Delaunay simplices [20]. For each atom of a protein, Summa et al. defined a microenvionment as the atoms that are at distance less than r (best observed values 4.5 and 4.8), and that are at least at s residues along the backbone (best value between one and three) from that atom [183]. While a central atom can be of 20 different types, a microenvironment cumulates atom counts into four classes.

The combination of physics-based potential and KB potential seems to be an efficient complementary approach. The neighborhood of each atom should encode fine properties of folded proteins. The microenvironment is of particular interest, but has major issues. As it relies on the count of atoms (instead of a kind of continuous quantity), it does not make the difference between two atoms of the same type at distance $\frac{r}{2}$ and r from the central atom. Moreover, the cooperation of atoms are not really taken into account: the fact that a given set is well distributed around an atom, or only on one side is not encoded in an microenvironment.

2.2.4 The Docking Problem

The function of a protein is determined by its potential interactions with other partners (called *cofactors*). Starting from the structures of the partners, a *docking* procedure is a method that predicts a assembly of the partners. One such assembly is called a *complex*. Complexes fit in three main categories: protein-protein complexes, protein-drug complexes and protein-nucleic acid complexes. Note that a given protein may have a slightly different tertiary structure depending if it is found experimentally in a complex (called its *bound* form) or free (called its *unbound* form), as it may deform (of even locally refold) upon binding. For illustration purpose, we now focus on protein-drug and protein-protein complexes (protein-nucleic acid complexes are also of fundamental interest, but will not be addressed here).

Protein-drug complexes Protein-drug complexes are key targets for pharmaceutical companies. Indeed, proteins are drug receptors, and a good understanding of all the processes related to this assembly type is key to more efficient treatments. Given an interesting protein (a therapeutic target), finding all regions of that protein that are able to accommodate a drug is a major issue. Such a set of atoms is called a *pocket* (See Fig. 2.12). This problem is indeed of particular interest for drug makers, because

[40] The dual complex is a subset of the regular triangulation, which is equivalent the 0-alpha complex defined on a set of weighted points [79]. A sphere/ball can be considered as a weighted point, using its center as point, and its squared radius as weight.

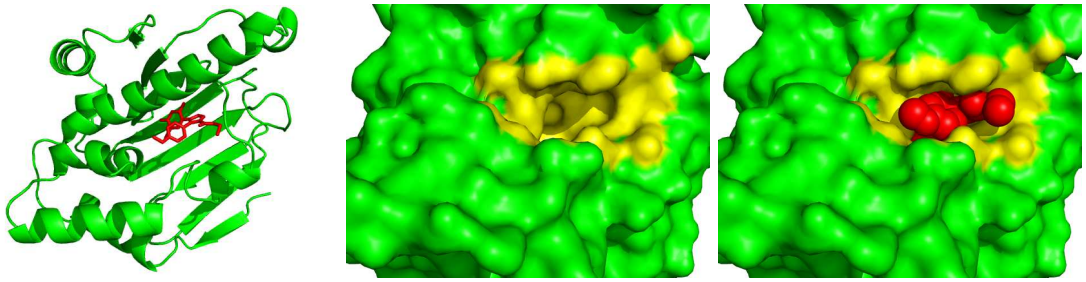


FIG. 2.12 – The molecular chaperone HSP90, and a small molecule inhibitor (PDB code 2BSM). One subunit of the dimer of the N-terminal domain only is depicted. From left to right: The protein is represented in cartoon mode, the ligand with sticks; the yellow Connolly surface patch corresponds to the approximate locus of the pocket accommodating the inhibitor; Using the Van der Waals representation, the ligand fills the pocket.

given a pocket, research on drugs that can fit and bind inside this pocket delivers information about its interactions within an organism (namely its primary and potentially its side effects). A process called *screening* consists in taking a protein and trying to bind a library of drugs in vitro. This process is time consuming and expensive. If the structure of the previously selected protein is available, and its pockets are known, a virtual screening process can be used to discard drugs with low binding affinity, so as to focus on promising drugs (based upon geometric and chemical properties). In addition, sequence homology is also often used to detect potential binding on known proteins that can lead to side effects. Several algorithms have been developed to compute these pockets. When the pocket is totally buried in a protein (i.e. a cavity), using dual complex is obviously a good solution [79]. Indeed the union of balls and its corresponding dual complex are homotopy equivalent. Moreover, the dual complex is included inside the associated union of balls, and a cavity of a union of ball is contained inside a cavity of the dual complex (see Fig. 2.10). When the pocket is not a cavity, several approaches have been developed (See [124] for the details). Protein-drug docking algorithms used to rely on the lock-and-key paradigm, and only rigid motions were considered. As testified by [181], efforts are now directed towards flexibility handling.

Docking assessment Protein-protein complexes are central to all activities of a living organism (immune response, signal transmission, DNA repair, ...). The study and understanding of the association process is thus really important for biological and medical applications. Indeed, many diseases are related to an assembly defect due to protein mutation. For example the mutation of a glutamic acid to a valine in the sequence of one chain of the hemoglobin (a tetramer), results in loss of elasticity of this protein (responsible of oxygen transport) and a change of its shape when releasing oxygen (leading to vessel occlusion and ischemia). This disease is known as the sickle-cell anaemia.

Most of the docking experiments suppose that two partners are known to interact. The problem of predicting whether two given proteins interact is hard. As of today, finding a criterion that can discriminate a native from a non-native interaction is difficult[41], but tractable if the interaction regions are known [167]. In the following the docking procedures considered are provided two partners in an already experimentally observed complex.

A docking algorithm usually decomposes into two parts: an exploration step which consists in exploring the relative position of the partners, and a scoring step which evaluates the quality of the proposed complexes. To assess the quality of protein-protein docking algorithms (and increase their prediction quality), an objective test procedure is needed. The *Critical Assessment of PRediction of Interactions*[42] (CAPRI) experiment is the equivalent of CASP but for prediction of complexes. When a structural biology group is about to publish a new complex structure, the complex can be submitted to the CAPRI committee. Several complexes (two or three)[43] are gathered to set up a *round*. After a couple of rounds

[41] especially if a so called non-native interaction is actually a true native interaction not already experimentally observed.
[42] http://www.ebi.ac.uk/msd-srv/capri/
[43] The overall number of complexes whose structure is known is relatively low. As of September 2008, RSCB web site

a meeting is organized to discuss the performance of prediction algorithms. The third CAPRI meeting was held in April 2007 [115]. The main conclusions of the last meeting [127] are that current algorithms perform better on targets that provided at least one of the two partners in its bound form (the case when the two partners are in their bound forms is usually not considered as it is not biologically relevant). Most of the targets presented were composed of the unbound forms of the partners, and the modeling of conformational changes remains a difficult task for the prediction algorithms. On some targets no group were able to find better than *acceptable* predictions. We refer to the prediction report [127] for a detailed discussion of specific problems encountered and quality assessment.

Scoring predictions One of the research directions in CAPRI is the scoring of predicted complexes. For this purpose, CAPRI comes with a scoring round experiment that follows each prediction experiment. In addition to the ten predictions per target, each *predicting* group is allowed to provide their best predictions. Predicting groups are invited to upload a total of 100 predictions per target. After shuffling, *scoring* groups have to report the 10 best scored predictions among those uploaded by all initial predicting groups. A situation justifying the need to enhance the development of scoring functions is that on one target, a scoring group reported within the best prediction another group's prediction, whereas one of their own predictions was of higher quality. A striking observation is that current algorithms may be able to predict high quality complexes, but scoring functions may discard them.

The bibliography on docking algorithms and scoring functions is wide. For more details on docking algorithm we refer to the CAPRI prediction report [127], but also to an early review [107]. This review also presents the developments of scoring functions. The main criterion used is geometric complementarity. Geometric complementarity is necessary for the *stability*, but it is not sufficient. The chemical nature of interacting atoms is essential. Geometric complementarity is now used to filter out complexes with fitting defects. Research directions for scoring functions are now based on using a weighted combination of several terms (including geometric parameters), and especially energetic terms [157].

Nevertheless, geometric properties may lead to further improvements. Identifying interface atoms as those losing accessibility to the solvent, several studies focused on the number of residues involved at interfaces, the number of hydrogen bonds, packing of interface and SAS lost upon formation of a complex [114, 64]. Considering the Voronoï diagram of a coarse grain representation of residues in complexes, the volume of interface cells and area of interface facets have been used as parameters of an empirical scoring function [29]. A knowledge-based potential scoring function relying on the dual complex of a coarse grain approximation of a complex has been defined [134]. For each pair of pseudo-atoms connected by a 1-simplex in the dual complex, the size of the 1-star of each pseudo-atom in the dual complex is used to define their function. The definition of complex interfaces by filtering power diagram between two partners [43], (in addition to report figures on size, curvature, chemical contact distribution and connectivity of the interface) shed a light on the fact that even atoms/residues that do not lose solvent accessibility can contribute to a definition of an interface.

Many studies on protein interface limit the geometric investigation to the surface of proteins losing accessibility. This remark raises questions on the role of a second layer of atoms/residues close to the surface of the interface. Moreover, several studies report specific properties of atoms getting buried upon formation of a complex. Another issue that should be addressed is that of multi-body interactions, and in particular keeping tracks of which type of atom is covering an atom of a given type.

2.3 Contributions and Thesis Overview

To the best of our knowledge, considering geometric contact areas (between atoms, residues, ...), only pairwise contacts have been studied. One possible reason may be due to the lack of easily implementable methods to go beyond pairwise contacts. The needs are to encode for each atom a decomposition of the surface of its corresponding sphere into faces. A face is defined as the maximum connected component of the surface of an atomic sphere that is covered by exactly the same atomic balls—possibly none. The computational geometry construction that perfectly matches this definition is the arrangement on the

reports that only 2141 out of 52,959 structures available are complexes.

surface of each atomic sphere of the circles induced by the intersection with neighboring atomic spheres. Providing each face with the list of atoms whose ball covers it, produces a precise encoding of atomic neighborhoods. Notice in particular that a region covered by k balls encodes a contact of order $k+1$. That particularly faces the problem of encoding multi-body contacts. It also generalizes the algorithms to compute the classical SAS or VdW surface (areas), or even those computing contact areas aforementioned. Such a method can also be used to investigate questions raised about knowledge-based potentials and scoring functions for folding and docking.

In the following chapters, we present an exact solution to the problem of computing an exact arrangement of circles on a sphere, with list of balls (called *covering lists*) for each face. We describe our solution as follows. Chapter 3 describes an adaptation of the Bentley-Ottmann algorithm to report all the intersection points of a set of circles on a sphere. Chapter 4 describes an extension of this algorithm, to explicitly construct the arrangement and the covering lists. The algebraic and geometric primitives needed to perform these actions are part of the 3D Spherical Kernel concept, which is described in chapter 5. Chapter 6 shows an application to flexible-loop docking where the arrangement and the covering lists enhance predictions. Finally, chapter 7 details the implementations of algorithms and concepts introduced in the chapters 3, 4 and 5.

2.3.1 Chapter 3: Reporting the Intersection Points of a Set of Circles on a Sphere

Motivations and previous work

History In 1976, Shamos and Hoey [176] described a $O(n \log n)$ time and $O(n)$ memory algorithm to determine whether any two line segments in the plane among n intersect, together with an application to detect whether two simple planar polygons intersect. Answering an open problem posed in this latter article, Bentley and Ottmann [23] in 1979 reported an output sensitive algorithm to compute the k intersection points of a set of n line segments in the plane. The algorithm has $O((n+k)\log n)$ time complexity and requires $O(n+k)$ memory—further improved by Brown [34] in 1981 to $O(n)$ memory with the same time complexity. The first algorithm computing the k intersection points between n line segments in the plane in linear time in k, was given by Chazelle [49] in 1986. The complexity of this algorithm was $O(n(\log^2 n/\log\log n)+k)$ time and $O(n+k)$ space, later improved to the optimal $O(n\log n+k)$ time with same space requirements by Edelsbrunner and Chazelle [50] in 1992. In 1995, Balaban [19] published an algorithm with the same optimal time complexity but improved the space requirements to $O(n)$—later made in-place by Vahrenhold [187]. An incremental version is given by Chazelle et al. [53] with $O(n^2)$ time complexity and storage. We may cite papers of Clarkson and Shor [60], and Mulmuley [147], describing randomized algorithms for this problem, in $O(n\log n+k)$ time and $O(n)$ space. Considering the problem of reporting the arrangement of a set of n lines in the plane, Edelsbrunner and Guibas introduced the idea of a topological sweep (replacing the sweep line by a sweep curve) [80, 81] running in $O(n^2)$ time with $O(n)$ space requirements. The algorithm is based on the idea that each pair of lines is intersecting once. An extension to directly handle degenerate cases has been proposed by Rafalin et al. [161]. Many properties of the arrangements of different geometric objects have been studied (complexity of a single face, complexity of a zone,...) and the reader is referred to [2, 4, 14] for more details. For a survey on geometric intersection algorithms refer to [98]. For parallel algorithms, refer to chapter 4.1 of [8] and to [10] for example. For more information on arrangements, the reader can read the most recent survey available [92] while writing this chapter.

Robustness issues for sweeping circles on a sphere Recent implementations of geometric algorithms usually make a clear distinction between, on the one hand, the predicates and constructions, and on the other hand, the combinatorial part of the algorithm (that calls predicates and constructions).

A *predicate* is a test function returning an element of a discrete set of values (for instance: *do the two curves c and c' intersect?*). A *construction* is a function that constructs new objects (for instance: *compute the intersection points between curves c and c'*). The robustness of a geometric algorithm is thus related to the expected behavior of the predicates and constructions with respect to the combinatorial part, and also to the design of the combinatorial part (i.e. degeneracy handling). The problems related to

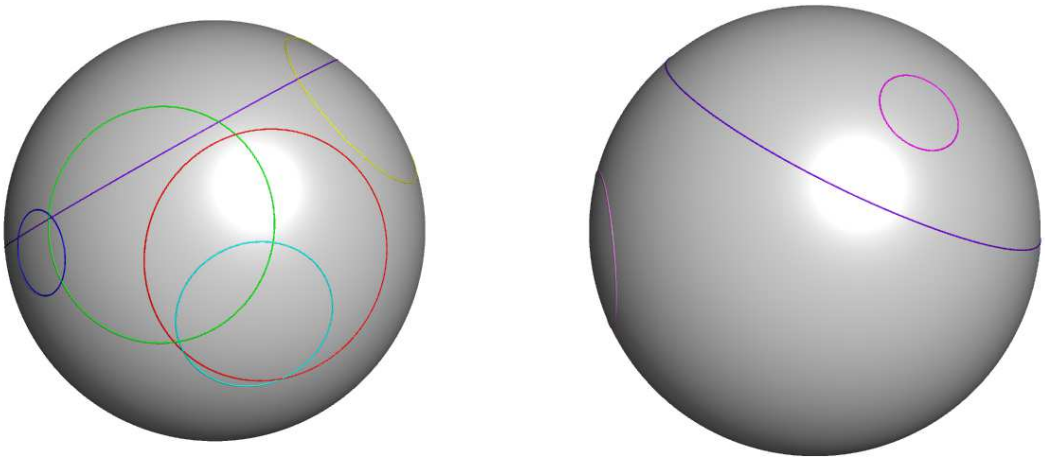


FIG. 2.13 – Eight circles are drawn on a sphere. Chapter 3 presents an algorithm to compute the intersection points of these circles (front and back views).

the evaluation of predicates and constructions are introduced in section 2.3.3, and addressed in chapter 5. One issue in the design of the combinatorial part is the handling of degenerate cases (e.g. when two circles are tangent). In order to cope with such cases, one can rely on two strategies. The first one is to address all of these cases explicitly. The second one is to use perturbations, which allow to write the combinatorial part under the assumption that the input data is not degenerate. There are two efficient types of perturbations: (i) controlled explicit perturbations which consist in perturbing the input data so as to guarantee that the input is degeneracy free (as used in [106] for example); (ii) implicit perturbations [82] which consist in applying a symbolic perturbations to the input data, using a polynomial in ϵ that vanishes with ϵ. A degenerate case is then detected within a predicate using the non-perturbed input. In such a case the evaluation of the predicate resorts to the perturbed data using a small enough ϵ value. For the case of reporting an exact arrangement, none of the perturbation schemes is relevant. Explicit perturbations change the input data. Implicit perturbations induce a coherent change on decisions taken during the algorithm, the embedding is not changed. In the case of two tangent circles, an intersection predicate would report either no intersection or an intersection in two points, while the construction would report one point.

Sphere versus plane The original Bentley-Ottmann algorithm [23] was already designed to handle circles in the plane. Theoretically, the transition from the sphere to the plane requires to a mere a stereographic projection. Implementation-wise, it is not so clear. Indeed, operations involved to sweep circles in the plane or on a sphere are the same, up to intrinsic degenerate cases of the spherical case. As we will see in chapter 3, these degenerate cases are easy to handle and do not induce any time complexity overhead. Moreover, the stereographic projection also has degenerate cases when the projection center is on a circle (see section 3.1).

Contributions

Given a collection of circles on a sphere, the work presented in chapter 3 is dedicated to reporting the points common to at least two circles. The algorithm presented uses an original data structure to handle degeneracies due to more than two circles going through a common point: the *event site*. The representation of intersection events allows a combinatorial reversal of arcs in the vertical ordering, together with the maintenance of the linear size of the event queue. Both running time and memory requirements are actually concerned with this latter aspect, as testified by experiments in section 5.5. Finally, this algorithm allows to implement the construction of the arrangement of a set of circles on a sphere, together with the covering lists presented in chapter 4.

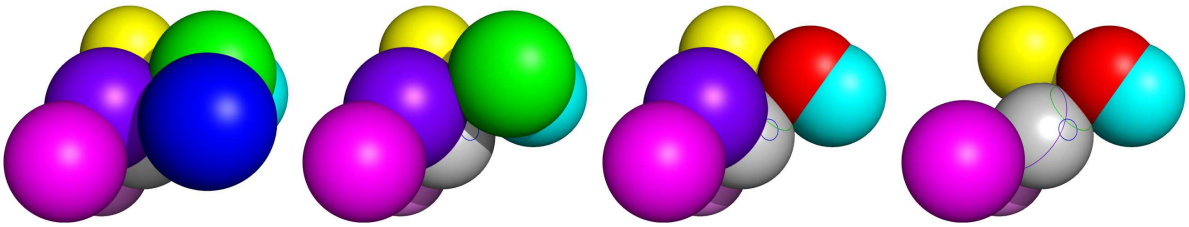


FIG. 2.14 – Eight spheres intersect a grey one. Chapter 4 presents an algorithm computing the arrangement on the grey sphere of the intersection circles. In addition, for each face, the list of spheres whose balls cover that face is also reported.

2.3.2 Chapter 4: Constructing the Arrangement Induced by a Set of Circles on a Sphere, together with Covering Lists

Motivations and previous work

Applications Consider a sphere S_0 and a list of circles on it. The *arrangement* of circles on S_0 is the partition of S_0, induced by the circles, into regions whose interior is connected (See Fig. 2.13). In general, most of the applications involving spheres and balls (see introduction of chapter 5 in section 2.3.3) are based upon simple geometric tests between balls, in particular the intersection test. However, a number of them would benefit from finer information—like the aforementioned arrangement. Taking advantages of this construction, some applications have already been developed to model directions in which an object can move. Considering one point per object, each region of the arrangement describes a set of directions either allowed or forbidden. For example, an arrangement of great circle arcs on a unit sphere was used in casting problems, to define regions capturing directions along which a polyhedral object can be extracted from its mold [5]. Such an arrangement was helpful in planning assembly of mechanical parts to define directions free from collision [164](subsection 3). Arrangements of great circular arcs on a sphere were also used in radiosurgery to enhance and speed up the treatment of brain tumors by defining forbidden regions where the beam could damage critical parts of the patient's brain [174]. In computational structural biology applications, perturbed arrangements of circles on atomic spheres were used to compute the boundary of the union of atomic balls of a molecular model [106].

Applications in computational structural biology In section 2.2.2, the importance of pairwise contacts has been stressed. Suppose now that each circle on S_0 stems from the intersection between S_0 and a sphere S_i. Assuming that each sphere S_i is associated a ball B_i, the *covering list* of a face of the arrangement is the list of balls that contain it, and the *multiplicity* of the face is the size of the covering list (See Fig. 2.14). In the following we mention briefly a computational structural biology problem directly concerned with this extension. We consider the problem of encoding multi-body contacts between an atom and its atomic neighbors. In representing a molecule as a collection of balls, we recalled above the importance of molecular surfaces. For a given atomic sphere within a molecule, consider the arrangement induced by the intersection circles with neighboring atomic spheres. The contribution of this atom to the molecular surface, say the SAS or the VdW surface, consists of the cells of null multiplicity—the surface features the boundary of the union of balls i.e. the exposed spherical caps. Fig. 2.15 features the cumulative area (over all atoms of a complex) as a function of the multiplicity. Notice in particular that the surface area of the SAS corresponds to a mere 4.5% of the total computed surface area. Given that all previous studies overlooked faces of multiplicity > 1, arrangements of circles hold great promises to refine our understanding of inter-atomic multi-body contacts.

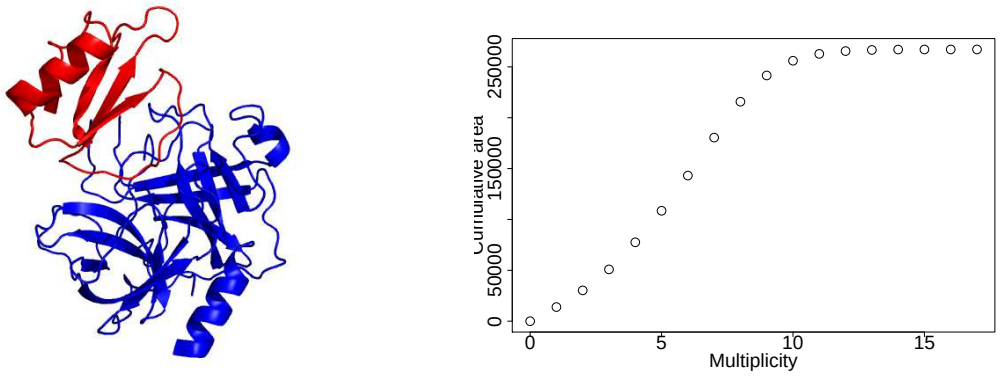


FIG. 2.15 – Modeling atomic multi-body contacts. Left: Backbones of a protein-protein complex (the bovine alpha-chymotrypsin-eglin c complex, PDB code `1ACB`) with two partners; Right: Cumulative surface area, over all atoms of the complex, as a function of the multiplicity of the cells. Multiplicity 0 corresponds to the SAS surface, and accounts for a mere 4.5% of the cumulative area. See text for details.

Another problem related to flexible-loop docking is introduced in section 2.3.4 and described in chapter 6.

Algorithm to compute arrangements of circles on a sphere An algorithm to compute the trapezoidal decomposition of a sphere induced by a collection of circles was developed [106, 86]. Degeneracies were removed using explicit perturbations. The first complete and exact implementation computing a planar map induced by the intersection curves of a set of quadrics running on the surface of one of them, say Q, is described in [26]. The algorithm reduces the problem to a planar arrangement of algebraic curves using a projection. Moreover, it is shown how this planar arrangement can be used to compute the arrangement on Q—a lifting step which can be avoided when directly computing the arrangement on Q in 3D [44]. Following previous work, a generic sweep line algorithm to compute the exact arrangement of curves on a surface was proposed [25]. Considering a surface and a set of curves on it, the arrangement of these curves is computed thanks to this algorithm, provided that a set of functionality gathered into a so called *traits class* fulfill a list a requirements of the algorithm. For the moment, in the case of circles on a sphere, such a traits class is available for great circles only[45]. Finally, and for the sake of completeness, we mention that an algorithm to compute the exact volumetric decomposition induced by an arrangement of quadrics [146]. Their strategy does not rely on surface arrangements, and no implementation is provided.

Contributions

The chapter 4 presents an integrated framework allowing one to construct the arrangement of circles on a sphere and to compute a compact representation of the covering lists called the *covering tree*. Thus, we make three contributions.

(i) We present the first effective algorithm to compute the exact arrangement of circles on a sphere. It is based on the spherical Bentley-Ottmann algorithm presented in chapter 3 that handles all degenerate cases.

(ii) The first description of an extension of the plane sweep line algorithm to compute an arrangement of line segments in the plane was given in 1982 [152]. It reports oriented cycles of edges describing simply connected regions of the plane. We present the construction of an extended half-edge data structure (faces may not be simply connected) storing the arrangement, a construction performed on the fly during the sweep process, and using Union-Find—an original alternative.

[44] However, this strategy is sufficient for reporting intersection points (i.e. problem in chapter 3), but more work is needed to construct the arrangement.

[45] Using the CGAL 3D Spherical Kernel presented in chapter 5, we started to write a traits class to enable the handling of all types of circles on a sphere by this algorithm.

(iii) Taking advantages of the construction while sweeping, a calculation of the covering tree of the arrangement using a compact representation is presented.

The only effective alternative, to the best of our knowledge, is the computation of a trapezoidal map, based upon explicit controlled perturbations of the spheres so as to get rid of degeneracies [106, 86]. Our strategy cumulates several advantages. First, the computation of the arrangement is exact, exactness being the route we chose to warrant robustness. While exactness is not a prerequisite for the applications we addressed in computational structural biology, it might be necessary in other applications. Moreover, we guarantee the exactness of the arrangement up to a small running time overhead (without changing the input data), as observed in section 5.5. Second and most importantly, running Bentley-Ottmann on the sphere provides a direct route for the construction of the arrangement (instead of the classical trapezoidal decomposition [31]), and of the covering lists.

2.3.3 Chapter 5: Algebraic and Geometric Operations Necessary to Handle Circles and Spheres in 3D

Motivations and previous work

Linear versus curved primitives in computational geometry. Until recently geometric algorithms used to limit their framework to linear objects in affine Euclidean space (points, segments, triangles, ...), curved objects being discretized by linear elements. However, handling directly curved objects allows one to skip this discretization process, thus reducing the number of primitives and yielding algorithms with better combinatorial complexity. For these reasons, the direct manipulation of curved objects is an important challenge in computational geometry. Of particular interest in the realm of curved objects are the simplest primitives, namely circles, circular arcs and spheres, which are ubiquitous to either represent or approximate more elaborate objects.

Regarding one dimensional primitives, circular arcs in the plane are involved in the design and the manufacturing of printed and integrated circuits [37], and they also generate an increasing interest for approximating curves [77, 6].

Balls and spheres are probably even more central, especially to approximate 3D objects. Given a known 3D object, balls can be used to provide hierarchical approximations, for example to efficiently perform collision detection [3] or multi-scale visualization [166]. Given a 3D point cloud sampling an unknown object, balls are also key for the inference of geometric and topological properties of the unknown object from the samples. Topologically speaking, balls can indeed be used to reconstruct the object by mimicking the medial axis transform, which is the bottom-line of say the power crust algorithm [11]. For a large class of (non-smooth) compact sets, the union of balls centered on sample points allows one to retrieve the topology of the compact under suitable sampling conditions [47]. From a more geometric standpoint, a number of quantities can be inferred from balls centered at the sample points. One may mention the so-called boundary measure of a point cloud, from which singular points and sharp features of the sampled model can be estimated [48]. This inference requires computing the volume of a ball contained within its Voronoï region, a calculation involving geometric constructions on a sphere [141, 18]. Given a 3D polyhedron, additional geometric modeling constructions involving spheres are the computation of offset surfaces. (Notice that the offset surface calculation covers the 3D Minkowski sum between a polyhedron and a sphere.) Volumetric objects in 3D can be approximated by a union of balls [163]. Interested readers may also consult chapter 11.12 of the Visionbib bibliography project at `http://www.visionbib.com/bibliography/describe482.html`, which contains an exhaustive list of representations involving spheres.

Finally, spheres are also central in molecular modeling since a Van der Waals model features a collection of balls whose radii depend upon the atom type and its covalent environment. In particular, the combinatorial and geometric properties of the three main definitions of surface of a molecule used (the Van der Waals surface, the solvent-accessible surface and the molecular surface) derive from the boundary of a union of balls [63].

Predicates versus constructions in the exact geometric computation paradigm. Recent implementations of geometric algorithms usually distinguish between predicates and constructions. Since rounding errors due to floating point calculations often produce inconsistencies [120], the robustness of

geometric algorithms relies on the exact evaluation of predicates [196]. This problem can be solved using two different alternatives: (i) evaluating the sign of polynomial expressions on the input data, which can be done efficiently with arithmetic filtering techniques [132]; (ii) storing in the course of the predicate evaluation, intermediate geometric objects to be reused later on. For curved objects, the former strategy typically consists of using an implicit encoding of an algebraic number by some polynomial vanishing on it, which calls for resultant-like calculations on the input polynomials. The latter relies on exact representation of algebraic numbers [65, 125], and requires in the worst cases running multi-precision calculations down to separation bounds. [35, 133]. But in doing so, one reuses intermediate calculations, and the design of predicates, a difficult task for algebraic objects, is made easier. In fact, work on filtered constructions [96, 88] may reconcile the evaluation of predicates and constructions. The implementation of such a framework still needs to be worked out to possibly surpass the approach relying on predicates without intermediate constructions.

The CGAL library, kernels and traits classes. The goal of the CGAL open source project is to promote research in computational geometry, so as to make reference algorithms available as robust programs for academic research and industrial applications. We refer the reader to the CGAL web site and to the literature for more background on the history and running of the CGAL project [44, 93]. Geometric algorithms are generic, and the geometric classes are instantiated with a *Traits* class: a class offering the minimum set of functionality required the algorithm. Such a template parameter is supposed to provide a set of functionality, with prescribed signatures, described and documented as a *concept*.
In the CGAL library, constant size geometric primitive objects, together with general purpose predicates and constructions on them, are made available in *kernels*. As kernels can be directly used as traits classes by several CGAL classes, CGAL kernels are documented as C++ concepts—we say that one implementation of a given concept is a *model* of the concept.

Circles and spheres: from primitives to algorithms As a follow-up to algorithms based on linear primitives, a recent trend in computational geometry has been the design of algorithms and programs based on curves and surfaces. The work mostly focused on computing arrangements in 2D and 3D, which involves primitive computations such as intersection and relative position of objects.

As far as we know, in the planar case MAPC was the first geometric library handling general algebraic curves [121], but not all degenerate cases were not handled. The EXACUS prototype library allows to exactly compute arrangements of conics and cubics in the plane [85]. The first version of a kernel for non-linear objects (circles and circular arcs in 2D) was released in CGAL 3.2 [158]. The CGAL arrangement package can use this kernel to compute an arrangement of circular arcs in the plane; it can also deal with conics or Bézier curves [193].

In the three dimensional setting, Andrade and Stolfi proposed selected exact manipulations of 3D circular objects lying on a given sphere, with an implementation in Modula-3 [12]. Their representation of objects uses Plücker coordinates and is restricted to the case of circular arcs on a given sphere, whereas our framework handles general circles and circular arcs in 3D. Some algorithm for arrangement of curves on surfaces were also introduced previously.

Contributions

The work presented in chapter 5 provides the first complete and robust implementation of essential basic functionality to manipulate in an exact way spheres, circles and circular arcs in 3D, so as to match most of the needs of applications described above. The perspective is that of a hand-crafted package geared towards the aforementioned objects. Following best practices, a clear distinction is made between the algebraic and the geometric aspects on the one hand, and on the concepts of a kernel and its implementation on the other hand. The 3D Spherical Kernel concept is accompanied by an implementation, which is used to compute the exact arrangements of circles on a sphere. As a quality label, the basic part of the package, related to general spheres, circles and circular arcs in 3D, has been reviewed and accepted by the CGAL Editorial Board [38]. The second part of the package, offering functionality on a reference sphere, will follow the process in a second step. In particular, the diffusion of our code will enable a number of endeavors. As an example, it will foster the development of a traits

class providing the predicates required to handle all types of circles in the general sweep-line algorithm for curves on a surface [25].

2.3.4 Chapter 6: Using the Arrangement of Circles to Select Diverse Conformational Ensembles, with Application to Flexible Docking

Motivations and previous work

Ensembles in molecular modeling Protein-protein interactions are paramount to all biological processes, but their prediction from unbound geometries faces major difficulties, as evidenced in the CAPRI experiment, by the low number of *medium* and *high* predictions carried out on flexible systems—as opposed to *incorrect* and *acceptable* one [115]. Since proteins are intrinsically flexible, they continuously undergo conformational changes over time, or equivalently, they exist at a given time as an ensemble of conformations in equilibrium. During the exploration of conformational space, conformations preferentially occupy energy regions which are characterized by low free energies. For proteins of moderate size undergoing small amplitude movements occurring in tens of nanoseconds timescale, conformational changes can be investigated using molecular dynamics. For more complex cases, where flexibility applies to large parts of the protein backbone or where the amplitude of the movement is important, discrete ensembles of conformations known as *conformers* can be pre-generated and considered simultaneously. This representation is particularly appropriate when dealing with macromolecular docking. In the case of protein association, one indeed wishes to predict the best possible bound geometry of two flexible objects, which implies exploring the relative position and orientation of the partners, but also their conformational space so as to pack the interface. In the Monod-Wyman-Changeux interpretation [144], the unbound proteins are considered as two collections of conformers in thermodynamic equilibrium. When the partners bind, the equilibrium is shifted toward the structure observed in the complex. Implementing this strategy may be done at the global (i.e. protein) scale [105], local (i.e. side chain) scale [39], or intermediate (i.e. loops or domains) scale [22].

Generating and selecting conformers: energy versus geometry Representing flexibility through an ensemble of conformers is computationally feasible only if this number is not too large. It is therefore essential for this reduced number of conformers to be as representative as possible of the conformational space available to the flexible molecule or molecular fragment. Since conformers are of interest for a number of applications, which criteria (geometric or energetic) should one use to generate and/or select them? From a statistical viewpoint, minimum energy should be the criterion of choice for generating ensembles representative of the thermodynamic equilibrium between conformations. However, this criteria is generally not tractable for several reasons.

First, the exhaustive exploration of the conformational space of large systems or of systems with large amplitude deformations is not possible. To keep calculations tractable, methods undertaking this task favor geometric calculations, and defer energy calculations to later stages [67, 73]. Second, when conformers are used to model a region of a protein, the energy associated to each conformer varies with its environment. In docking, for example, the energy of each copy depends upon its interactions with the partner of association (direct electrostatic or Van der Waals interactions, modification of the dielectric environment, desolvation energy). Therefore, weighting a conformer as if it were alone does not, in general, account for its probability of occurrence. Third, it may happen that the energy landscape associated to a flexible protein is rather flat, with very small energy barriers between the conformers. In contrast to flipping between well separated conformers, the protein flexible fragment can largely explore the available space. In this case, it is important to be able to sample exhaustively the space available to the flexible element.

We may also notice that generation of diverse ensembles is a strategy of choice to simulate complex processes. For example, diverse ensembles generated using a repulsive umbrella potential have recently been used to investigate domain swapping [140].

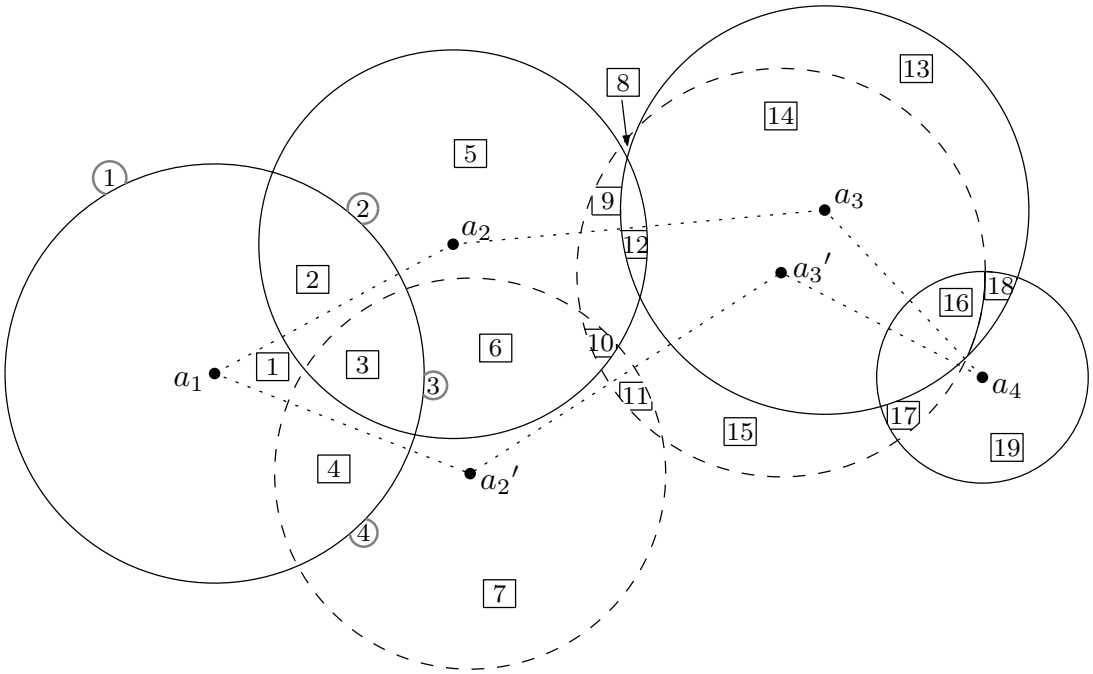


FIG. 2.16 – Example of $2D$ conformers, each consisting of four balls—first and fourth balls are common. The (two dimensional) volume occupied by the two conformers is decomposed into 19 cells (boxed numerals). The circled numerals feature the surface arrangement of the ball centered at a_1, based on intersections with neighboring balls.

Contributions

Let us consider a collection $\mathcal{C} = \{C_1, \ldots, C_n\}$ of n conformers (rotamers, protein loops, whole protein), each represented by a collection of balls, each ball being bounded by a sphere. This model is rather general, as the balls in Van der Waals (VdW) model may represent atoms, or may model residues: The model considered can be either atomic or coarse grained protein. As just argued, sampling the conformational space available is an important requirement. In chapter 6, we actually wish to solve the following problem:

Given a pre-computed collection of n conformers and an integer $s < n$, report a *selection* of s conformers maximizing some geometric diversity criterion.

To specify the type of geometric criterion we have in mind, observe that the union of the balls of the conformers in the selection defines a volume, whose partition by the spheres bounding the balls is called a *volumetric arrangement* (also called *volumetric decomposition*). Similarly, the decomposition of each sphere by the intersection circles with other spheres defines a *surface arrangement* (also called *surface decomposition*). See Figs. 2.16 and 2.17 for a $2D$ illustration. Using these arrangements, we investigate several geometric optimization problems whose output is the selection. These problems aim at maximizing the *spatial occupancy* of the selection, in several ways. For example, we may wish to report the s conformers maximizing (i) the volume occupied by these conformers (ii) the molecular surface area (MSA) of the union of the conformers (see Fig. 2.19).

As an illustration, consider Fig. 2.18(i), which features 40 conformers of the flexible loop of a complex. A number of these loops are obviously redundant, and one would like to trim this set to select a diverse subset. Such a selection, generated by our algorithm, is presented on Fig. 2.18 (ii) and Fig. 2.19.

Application-side, we replace the conformer selection problem in the context of multiple-copy flexible-loop docking. On the systems tested, we observed that using the loops selected by our strategy can significantly improve the result of a such docking procedure.

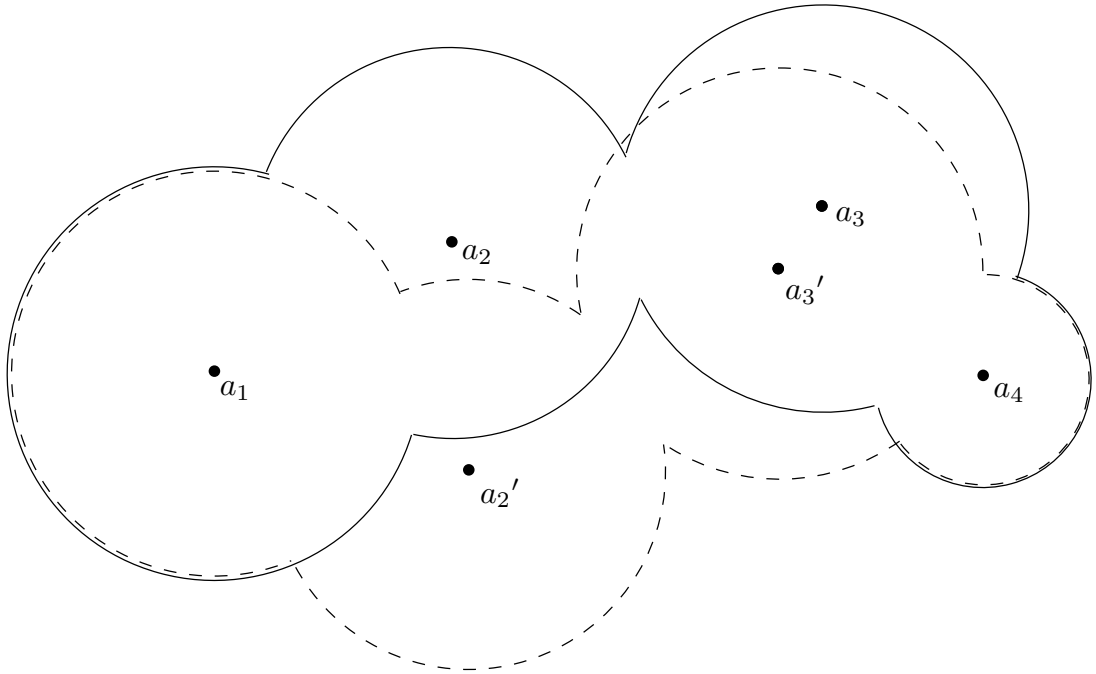


FIG. 2.17 – The boundaries of the unions of balls of the two conformers of Fig. 2.16, in solid and dashed lines.

FIG. 2.18 – Selecting diverse conformational ensembles from a pool of conformers of a flexible loop of a thrombin—complexed with bovine pancreatic trypsin inhibitor in file PDB code `1BTH`. From left to right: (i) backbone of the receptor in cartoon mode, with 40 conformers from the pool; (ii) backbone together with 10 conformers selected by our algorithm—called `Greedy`; (iii) backbone with 10 conformers selected by a standard hierarchical clustering algorithm—called `HClust`.

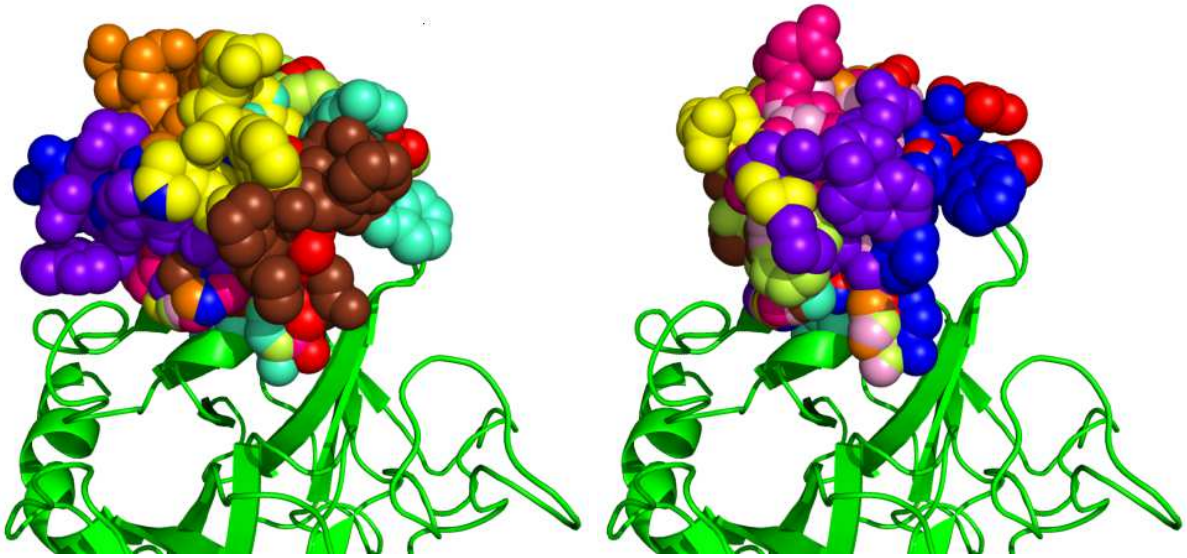


FIG. 2.19 – Follow-up to Fig. 2.18: Van der Waals representations of the 10 conformers presented on Fig. 2.18 (ii) and Fig. 2.18 (iii). Using one color per conformer, we observe that on the left, conformers are well separated (covering the available space), while on the right they are covering each other. The maximization of the surface exposed by the conformers is a diversity criterion.

2.3.5 Chapter 7: Implementation of Algorithms

Motivations and previous work

The CGAL project formally started in 1996 under the initiative of a consortium of a several teams in Europe and Israel (including our group at INRIA) and was supported by the European Community for three years. It evolved to an *Open Source* project, allowing other researchers to participate. GEOMETRY-FACTORY [46], a start-up launched in January 2003, is selling commercial licenses and supports specific developments based on CGAL.

At the same period, the *Computational Geometry Impact Task Force Report*, coordinated by Bernard Chazelle, gave a few recommendations [51, 52], that included the production of useful and usable geometric software, and the creation of a rewarding structure for implementations in the academic world. CGAL is striving to meet these two recommendations.

CGAL has high quality standards, which are ensured by a number of different means. First, an editorial board, created in 2001 and currently consisting of 12 members, is in charge of making technical decisions and coordinating CGAL promotion. The committee evaluates proposals of new packages, and manages a review process similar to that of papers submitted to journals. This process guarantees the coherence, homogeneity and quality of the library, and is also rewarding for authors whose packages are getting integrated within CGAL. Second, a detailed documentation featuring a user and a reference manual for each package is made available—over 3000 pages for 57 packages for the 3.3.1 release as of September 2007. Third, tests are run every night on internal versions so as to avoid code regressions.

As the arrangement of circles on a sphere package has applications in computational structural biology, we mention the BALL project[47] [122] which goal is to provide a unified and generic framework to develop software in this field.

Contributions

Chapter 7, outlines the implementation of the CGAL 3D Spherical Kernel and the different levels of the implementation of the arrangement of circles on a sphere package. An example showing how to

[46] http://www.geometryfactory.com/
[47] http://www.ball-project.org/

interface the arrangement package and the 3D Spherical Kernel to compute the surface arrangement induced by a set of spheres is also provided. The part of the 3D Spherical Kernel not dealing with object on a reference sphere and the Algebraic Kernel will be available in the next release (scheduled for late 2008). The remaining part, together with the *all-types-of-circles* traits class for the new CGAL generic algorithm [25], need indeed to be polished before being presented to the editorial board for review.

Chapitre 3

The Bentley-Ottmann Algorithm for Circles on a Sphere

Chapter Overview

This chapter is related to the problem of reporting the intersection points of a set of circles on a sphere using an adaptation of the Bentley-Ottmann [23] sweep paradigm. This adaptation is introduced in section 3.1. The handling of degeneracies is done through the structure of event site that groups events according to their embedding on the sphere. These event sites defined in section 3.2 are manipulated in the event queue. Section 3.3 describes the operations performed over the course of the algorithm from the viewpoint of manipulation of blocks of arcs involved in event sites. Maintenance operations of the vertical ordering is discussed in section 3.4, while section 3.5 deals with the correctness and the complexity analysis of the algorithm. To end up this chapter, the section 3.6 presents the pseudo-code of the algorithm.

This chapter is a longer version of the short paper published with Frédéric Cazals in the *Proceedings of the twenty-third Annual Symposium on Computational Geometry* [42] which accompanied the video accepted to the video and multimedia track[48]. An earlier version of this work has been described in a technical report [41].

3.1 Bentley-Ottmann Algorithms

3.1.1 The Planar and Spherical Settings

Line-segments in the plane The algorithm of Bentley and Ottmann (BO algorithm) reports intersections of line-segments in the plane [23, 68], by sweeping them using a vertical line. The position of each line-segment along this line is recorded, so as to detect intersection upon adjacency of two segments. The relative position of two intersecting segments along the vertical line is exchanged while the vertical line passes their intersection point. One needs an event queue $\mathcal{E}$ to schedule events modifying the vertical ordering (maintained in a sorted data structure $\mathcal{V}$), which are points where a segment starts/ends to be swept or where two segments intersect. The algorithm requires predicates to test whether two segments intersect, maintain $\mathcal{E}$, and maintain $\mathcal{V}$.

Circles in the plane In their article [23], Bentley and Ottmann also provided three properties that, if satisfied by a set of geometric objects in the plane, ensure that their algorithm can be used with the same guarantees for these objects:

- **Property P1**: *Any vertical line through the object intersects the object exactly once.*
- **Property P2**: *For any pair of objects intersecting the same vertical line it is possible to determine algorithmically (at constant cost) which is above the other at that line.*

[48] video available at `http://www.computational-geometry.org/SoCG-videos/socg07video/Arrangement.avi`

– **Property P3**: *Given two objects it is possible to determine algorithmically if they intersect, and if so to compute their leftmost intersection point after some fixed vertical line*

For circles in the plane, Property P1 implies that each circle must be decomposed into two circular arcs using the two tangency points of the circle with the sweep-line (called *critical points*). Property P2 asks for a predicate to compare the position of two circular arcs along a vertical line (to maintain $\mathcal{V}$). Property P3 asks for a predicate to determine if two circular arcs intersect and for a construction to build such an intersection point (maintain $\mathcal{E}$).

Circles on a sphere Considering the problem of circles on a sphere, the solution we chose is to directly handle circles on the sphere—a discussion on an alternative solution using a stereographic projection is given in the next paragraph. Our extension of the BO algorithm sweeps the sphere with a meridian M_θ anchored at the poles. The tangency point(s) between M_θ and a circle, if any, induce a decomposition of the latter into one or two θ-monotone circular arcs. This decomposition allows one to maintain the vertically sorted data structure $\mathcal{V}$ listing the θ-monotone circular arcs intersected by M_θ during the sweep—such circular arcs are called *active*. Since each θ-monotone circular arc is intersected in at most one point by M_θ (Property P1), notice that $\mathcal{V}$ implicitly orders their intersection points. Therefore, at a given θ, *below* and *above* should be understood w.r.t. this ordering along $\mathcal{V}$. In addition and to ease the search operations, $\mathcal{V}$ is also equipped with two sentinels set to the poles. The data structure used for $\mathcal{V}$ (and for $\mathcal{E}$) is a dictionary supporting the usual `contain, insert, delete` operations in logarithmic time[49]. The sweep process consists of having θ span the interval $(0, 2\pi]$. Similarly to the classical case, we consider $\mathcal{E}$ as an event queue. However, a major difference between line-segments and circles is that while endpoints are explicitly given in the former case, they are not in the latter. Notice that predicates and construction needed to satisfy properties P2 and P3 (on a sphere) are developed in chapter 5.

The stereographic projection vs. direct handling on the sphere From common geometric knowledge, the stereographic projection of a circle on a sphere, is either a circle or a line in a plane. The analytical geometry of this projection reads as follows:

Lemma 1 *Consider a sphere S_0 of radius R centered at the origin, and a plane $P = \{(x, y, z) \in \mathbb{R}^3; aX + bY + cZ + d = 0\}$ intersecting S_0 along a circle C. Using the stereographic projection from north pole N, to the plane $\{(x, y, z) \in \mathbb{R}^3; z = 0\}$, the image of circle C is:*

– *if P does not contain N, a circle of equation:*

$$\left(x + \frac{aR^2}{cR + d}\right)^2 + \left(y + \frac{bR^2}{cR + d}\right)^2 - \frac{a^2R^4 + b^2R^4 + c^2R^4 - R^2d^2}{(cR + d)^2} = 0$$

– *otherwise, a line of equation:*

$$2(ax + by) - cR + d = 0$$

Proof. The image of a point $p = (X, Y, Z) \in \mathbb{R}^3$, lying on S_0 and P, by the stereographic projection is a point $q = (x, y, 0) \in \mathbb{R}^2 \times \{0\}$ such that $p = N + kNp$, $k \in \mathbb{R}$.
To obtain the equation of a circle in the projection plane, an expression independent of k and involving $x^2 + y^2$ is expected.

We have: $x^2 + y^2 = \frac{X^2+Y^2}{k^2} \iff x^2 + y^2 = \frac{R^2-Z^2}{k^2} \iff x^2 + y^2 = \frac{R^2-Z^2}{(\frac{R-Z}{R})^2} \iff x^2 + y^2 = R^2\frac{R+Z}{R-Z}$. From this and using expressions of $\frac{x^2+y^2}{R^2} \pm 1$ we obtain that $Z = R\frac{x^2+y^2-R^2}{x^2+y^2+R^2}$ and that $x^2 + y^2 + R^2 = \frac{2R^2}{k}$.

The fact that p lies in the plane P implies that $aX + bY + cZ + d = 0$. Using previous equalities, we can rewrite this $akx + bkx + cR\frac{x^2+y^2-R^2}{x^2+y^2+R^2} + d = 0$. This expression is equivalent to $(x^2 + y^2 + R^2)k(ax + by) + (cR + d)(x^2 + y^2) + R^2(d - cR) = 0$ and to $2R^2(ax + by) + (cR + d)(x^2 + y^2) + R^2(d - cR) = 0$. Finally, if $cR + d \neq 0$, rewriting the last equality gives the expression of the equation of a planar circle. Note that P is the north pole of S_0 iff $cR + d = 0$. □

[49] Practically, we use a red-black tree.

Based upon the previous lemma, an alternative to the direct computation of the arrangement of circles on a sphere, is to resort to an arrangement of circles and lines in the plane. The direct construction was preferred for several reasons.

First, the spherical setting does not require dealing with infinite objects. Second, upon computing the arrangement in the plane, one does not need to pull back the coordinates of the singular point on the sphere—to get the spherical embedding of the arrangement. Third, consider a sphere whose squared radius is a rational number (its radius is not so). Then, the radius of a circle in the plane, which is used to infer the x-coordinate of its critical points, is the quotient of two algebraic numbers of degree two, which are not in the same extension. This situation does not arise on the sphere, where one always deals with algebraic numbers in the same extension (refer to section 7).

3.1.2 Circle Classification

The sweep meridian being anchored at the poles, the definition of θ-monotone circular arcs needs the following classification of circles w.r.t. the poles, illustrated on Fig. 3.1:

Definition 1 *Consider a sphere S_0, endowed with a north and a south poles. A circle C_i is called* polar *if it goes through a single pole of S_0;* bipolar *if it goes through the two poles of S_0;* threaded *if the intersection point between the supporting plane of C_i and the z-axis belongs to the open disk bounded by C_i;* normal *otherwise.*

With this definition, a normal circle defines two θ-monotone circular arcs homeomorphic to the closed line-segment $[0, 1]$; a polar circle defines such an arc, the second one reducing to a point—the pole itself. During the sweep process, $\mathcal{V}$ contains active arcs. For homogeneity, we consider that a threaded circle defines an arc which is always active. Note that defining θ-monotone circular arcs for a bipolar circle is useless since such a circle is not transversely intersected by M_θ, so that no insertion into $\mathcal{V}$ is planed—the circle contains the meridian at special θ values. From now on in this chapter, these θ-monotone circular arcs are just called arcs. For clarity in the sequel, an arc shall never be reduced to a pole.
As a *great circle* on S_0 is a circle whose center is the center of S_0, observe that a non-great circle is normal, polar or threaded, while a great circle is either threaded or bipolar.

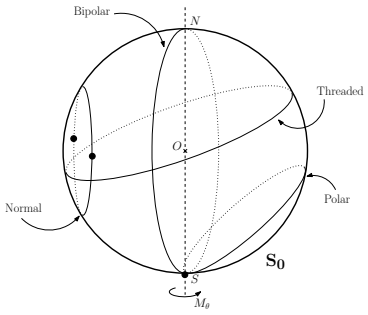


FIG. 3.1 – The four types of intersection circles.

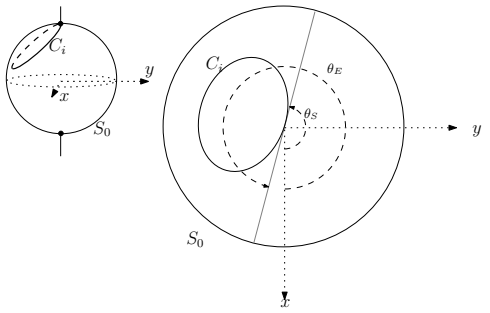


FIG. 3.2 – Start (θ_S) and end (θ_E) values of start and end points of a polar circle C_i.

3.1.3 Notation

Notation In this chapter we suppose that the sphere on which the arrangement is computed is denoted S_0. Each circle C_i drawn on S_0 is defined as the intersection of S_0 with another sphere S_i. A sphere S_i is represented by its center $c_i = (x_i, y_i, z_i)$ and radius r_i. Sphere S_0 is endowed with cylindrical coordinates (θ, z), and by poles we refer to its north and south poles. We consider the meridian M_θ of S_0 parametrized by $\theta \in (0, 2\pi]$.

In the following, we develop the combinatorial operations required by the spherical BO algorithm. All the predicates involved in the manipulation of intersection circles are presented in chapter 5 where the following is proved:

Theorem 1 *Predicates involved in the computation of the exact arrangement of intersection circles on a sphere rely on the comparison of algebraic numbers of degree at most two.*

3.1.4 Circles on a Sphere: Degenerate Cases

Generically, two spheres intersect in a circle and three spheres intersect in two points if they intersect at all. To enumerate all degenerate cases, we proceed as follows: starting from a generic configuration, we add the least number of spheres to end up with a non-generic one, and such that any subset of smaller size is a generic configuration. In the following, the spheres we start with and those which are added are separated by a semi-colon.
▷ **1.** PC_1, $\{S_0; S_i\}$: spheres S_0 and S_i are tangent i.e. intersect in one point.
▷ **2.** PC_2, $\{S_0, S_i; S_j\}$: S_0 and S_i intersect along a circle, and S_j is another sphere in the pencil of these two spheres, i.e. intersects along the same circle.
▷ **3.** PC_3, $\{S_0, S_i; S_j\}$: S_i intersects S_0 along a circle. S_j intersects S_0 so that the common intersection of S_i, S_j and S_0 is one point. Equivalently, C_i and C_j are tangent circles on S_0.
▷ **4.** PC_4, $\{S_0, S_i, S_j; S_k\}$: the two circles C_i and C_j intersect at two points, and C_k goes through one of these points. Equivalently, C_i, C_j, C_k intersect in a single point on S_0.
▷ **5.** PC_5, $\{S_0, S_i, S_j; S_k\}$: Circle C_k goes through the 2 intersection points of S_0, S_i, S_j.

Notice that case PC_3 is not inferred from PC_1 although S_i and S_j are tangent, since PC_1 is concerned by the tangency between S_0 and another sphere. Notice also relevant cases to consider in our BO adaptations are PC_3, PC_4 and PC_5. Cases PC_1 and PC_2 are concerned with the covering of faces of the arrangement by balls, and are not considered here as addressed in chapter 4.

Let us now just mention how degenerate cases are managed. Case PC_1 is detected from the radii of the intersection circles, which is null in this case [50]. Case PC_2 is detected using an order introduced in section 4.3.1 to sort intersection circles. Case PC_3, which features a single event, is handled in Algorithm 2. So are cases PC_4, PC_5, for which the ordering introduced in Def. 6 is essential in handling the several (at least two) colliding events.

3.2 Points, Events, and Events Sites

This section presents the formal definitions of the geometric and algorithmic representations involved in our algorithm. We start by a discussion of the difficulties faced.

3.2.1 Perspectives and Difficulties

Representation From a geometric standpoint, over the course of the algorithm, we shall focus on *event points*, i.e. points of S_0 where something relevant for the sweep process occurs: two arcs intersect in their interior at that point, or the meridian M_θ is tangent to a circle at that point. But since the algorithm manipulates θ-monotone arcs, we record the arc(s) involved with a particular event point into an *event*. Additionally, all events having the same event point are gathered into a so-called *event site*.

Ordering As the BO algorithm requires sorting event sites, one naturally resorts to the lexicographic order over cylindrical coordinates away from the poles. But this order is not sufficient to handle event sites involving polar and bipolar circles. The difficulties stemming from such circles motivate definitions 4 to 8.

[50] In constructing the arrangement in chapter 4, we shall actually skip this point, a design choice. We could equally report it as a point, located in a 2-face, 1-face or 0-face of the arrangement. As this point may be seen as the critical point of a circle of null radius, it can be located during the sweep process.

3.2.2 Points vs. Events

As emphased above, a point refers to a point of S_0, expressed in cylindrical coordinates.

Normal critical points Consider a normal circle C_i. The *normal critical points* associated to C_i are the two points where the meridian intersects C_i in a single point. The start point (θ_S, z_S) and the end point (θ_E, z_E) are the two points where the intersection between the circle and the meridian starts and ends respectively [51].

Polar and bipolar critical points When a normal circle is shifted towards a pole, its critical points coalesce at the pole. Denoting z_p the z coordinate of the corresponding pole, the *polar critical points* are defined as the pairs (θ_S, z_p) and (θ_E, z_p), with θ_S and θ_E the values such that the meridian is tangent to the circle. The start point is distinguished from the end point as for normal circles (see Fig. 3.2). We extend this notion to bipolar circles, yielding *bipolar critical points*, the θ values being those where the circle is included in the plane containing the meridian—the z coordinate being irrelevant. For such a circle, the start critical point has a smaller θ-coordinate than the end critical point.

Remark 1 *Critical points of a polar or a normal circle decompose its circle, say C_i into its lower and its upper arcs denoted $\underline{A_i}$ and $\overline{A_i}$ respectively, w.r.t. the z axis. When the upper or lower status does not matter, an arc is just denoted A_i.*

Crossing, tangency, intersection and singular points Whenever two arcs of non-bipolar circles intersect in their interior, the point is termed *crossing point*. If such arcs are tangent in their interior, the point is termed *tangency point*. If the tangency point coincides with the critical points of the supporting circles, we speak of degenerate tangency. Similarly, if the crossing point coincides with either critical point, we speak of degenerate crossing. In the following, an intersection points shall only refer to a crossing or a tangency point. We introduce the notion of *singular* point to refer to a point on S_0 which is common to at least two different circles. Refer to Fig. 3.3 for a summary of the notation.

The previous classification qualifies the local geometry at special points of S_0. But as *critical point* subsumes one circle while *intersection point* subsumes two circles, in order to distinguish the geometry (the *point*) from the arcs involved, we define the notion of event:

Definition 2 *The* normal (respectively polar) critical event *associated to a normal (respectively polar) critical point refers to the pair of arcs (respectively the arc) defining the circle, together with a tag* {Start, End} *stating whether the point is a start or an end point. A* bipolar critical event *refers to the relevant part of its circle bounded by the poles with a* {Start, End} *tag. An* intersection event *associated to an intersection point refers to a pair of intersecting arcs, together with a tag* {Intersection$_1$, Intersection$_2$, Tangency}*, stating whether the intersection corresponds to the smallest or largest point for a crossing* [52]*, or is a tangency.*

An event which is neither polar nor bipolar is termed normal. *The point of S_0 associated to an event is called the event point.*

Notice that no intersection event is associated to a degenerate crossing or tangency point: a degenerate crossing does not induce a permutation of arcs in $\mathcal{V}$; a degenerate tangency is not specified by a pair of active arcs.

3.2.3 Filling the Event Queue $\mathcal{E}$ with Event Sites

To run the BO algorithm, we schedule *event sites* into $\mathcal{E}$. In this section, we introduce event sites and a total order on them. To present this order, which is guided by the local arrangement of arcs in a neighborhood of the point of interest, we first define:

Definition 3 *Two normal events are in conflict if their event points coincide. A (bi)polar critical event and an event are in conflict if their event points have the same θ coordinate.*

[51] Note that this definition is independent from where the sweep starts.
[52] Recall we use the lexicographic order away from poles.

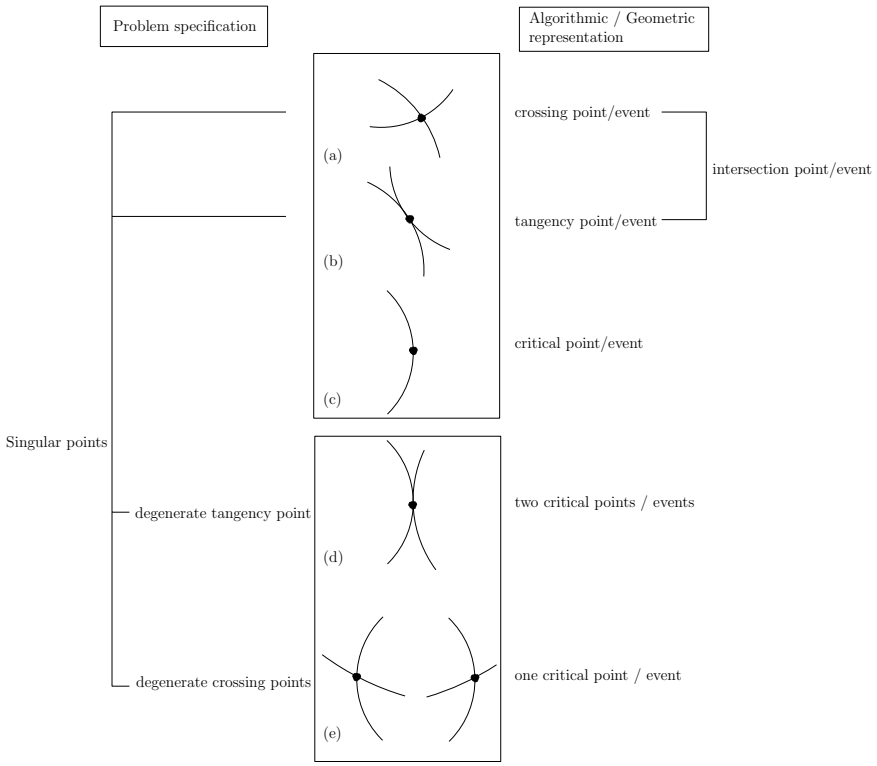


FIG. 3.3 – Terminology away from poles. Singular points are used to address the complexity of the algorithm, while the algorithm uses intersection (crossing, tangency) and critical (start,end) points and events. The terminology is driven by the decomposition of circles into θ-monotone arcs. Crossing points are the only singular points triggering a permutation of arcs in the vertical ordering $\mathcal{V}$.

Normal and (bi)polar event sites. To handle conflicts between normal event points, we define the (normal) event site data structure, which uniquely associates a collection of normal events to a point of S_0. More precisely:

Definition 4 *A* normal event site *is a collection of normal events with the same event point, partitioned into the following three data structures:*
– One list $\mathcal{F}$ for end (finish!) events, sorted by increasing circle radius.
– One list $\mathcal{S}$ for start events, sorted by decreasing circle radius.
– One list $\mathcal{CT}$ which contains the crossing events and tangency events. The restriction of $\mathcal{CT}$ to crossing (respectively tangency) events is denoted $\mathcal{C}$ (respectively $\mathcal{T}$).

Notice that elements of $\mathcal{F}$ ($\mathcal{S}$) correspond to the arcs of a circle to be removed from (inserted into) $\mathcal{V}$, while elements of $\mathcal{CT}$ refer to pairs of arcs intersecting. We will get back on the representation of intersection points in section 3.3.2. To handle circles through poles, we define similarly:

Definition 5 *A* (bi)polar event site *associated to one (bi)-polar event is the event itself.*

Since all events stored in a normal event site have the same event point, we can say that a normal event site also defines an event point. Using this point, the notion of conflict can be extended among (bi)polar event sites, and between (bi)polar and normal event sites.

Ordering normal event sites This case is easily handled using the lexicographic order:

Definition 6 *Consider two normal event sites whose event points are $p = (\theta_p, z_p)$ and $q = (\theta_q, z_q)$. The event site associated to p is said to* occur *before the one associated to q iff*

$$\theta_p < \theta_q, \quad or \quad \theta_p = \theta_q \quad and \quad z_p > z_q \tag{3.1}$$

Ordering polar event sites For conflicts between polar events, we need to distinguish between conflicts at the same pole (circles having the same tangent at the pole) and conflicts between circles through the two poles. Since (i)arcs associated to end events are removed from $\mathcal{V}$ before the insertion of arcs of start events; (ii)$\mathcal{V}$ is maintained top-down; (iii)the relative position of circles at a pole is a function of their radii, we get:

Definition 7 *For two polar critical event sites in conflict, one has:*
– A polar end event site occurs before a polar start event site.
– A polar event site at the north pole occurs before one at the south pole.
– Among polar start (respectively end) event sites at the same pole, the one whose associated circle is of largest (respectively smallest) radius occurs first.

Ordering event sites: the general case Finally we must resolve conflicts between two different types of event site (normal, bipolar, polar). First, as a natural extension of Def. 7, and since a bipolar circle features the largest radius possible, a bipolar event site in conflict with polar event sites must be enclosed between those representing the start and the end events. Second, as a normal event point whose θ coordinate matches that of a bipolar event site is located on the associated bipolar circle, the bipolar event site must be handled first. These two constraints do not impose an order between normal and polar start events. However, as normal event points are on the bipolar circle, we process them sequentially. Summarizing, we get the total order used to schedule event sites into $\mathcal{E}$:

Definition 8 *Event sites of different types, if in conflict, are ordered as follows:*

$$e_{P_{ep}} < e_B < e_N < e_{P_{sp}}$$

where $a < b$ means a occurs before *b; e_{P_*}, e_B, e_N stand for* polar, bipolar, normal event sites*; $e_{P_{sp}}$, $e_{P_{ep}}$ stand for polar* start, end.

Note that a conflict between two bipolar event sites cannot arise, as we excluded the case of identical circles in section 3.1.4.

Predicates needed to compare two event sites are developed in section 5.3.

3.3 Bentley-Ottmann on a Sphere

In this section, we recall basics about the classical BO (section 3.3.1), proceed with the maintenance of an essential invariant (section 3.3.2 and 3.3.3), and handle event sites (section 3.3.4).

3.3.1 Algorithm and Perspective

The algorithm, whose pseudo-code is presented on Algorithm 1 in Appendix 3.6, is similar to the standard BO algorithm in the plane [68]: we iterate over event sites and maintain the queue together with the ordering along the meridian. The ordering in $\mathcal{V}$ is initialized with arcs intersected by M_θ at $\theta = 0^+$. Queue $\mathcal{E}$ is initialized using critical events of all but threaded circles together with the intersection events between arcs adjacent along $\mathcal{V}$ at $\theta = 0^+$.

Of particular interest will be the problem of maintaining a linear size event queue. For BO dealing with line-segments, this strategy may be termed of classical in terms of algorithmic design [34]. But the picture is different implementation-wise as testified by the following footnote, from [143]:

> *Our X-structure may contain intersection points between segments that are no longer adjacent in the Y-structure. These events could be removed from the X-structure. Removing these events would guarantee an X-structure of linear size, however, at the cost of complicating the code. Since the size of the X-structure is always bounded by the size of the output graph we do not remove these events.*

In our work, the linear size of the event queue is not a goal per se, but it is a requirement of the definition of an intersection event as a pair of adjacent arcs along $\mathcal{V}$. To present our strategy, we first introduce the notion of block, which is used to encode the loss (and the gain) of adjacency upon updates of the event queue.

3.3.2 Blocks within a Normal Event Site and Reversal of Arcs

As reported in [34], maintaining a linear size queue requires keeping in $\mathcal{E}$ events (intersection events in our case) corresponding to arcs adjacent along $\mathcal{V}$ only. To do so, we need to detect pairs of arcs which are not adjacent anymore along $\mathcal{V}$ when processing an event site from $\mathcal{E}$, so as to update $\mathcal{E}$ accordingly. That is we wish to maintain the following:

Invariant 1 *Upon processing of an event site, the pair of arcs associated to each intersection event in the list $\mathcal{CT}$ of an event site of $\mathcal{E}$ are adjacent along $\mathcal{V}$.*

This invariant will be maintained by algorithm `break_adjacencies` (see section 3.3.3), which relies upon the notion of *blocks* partitioning the lists of a normal event site.

Definition 9 *A* block *is a sorted collection of events such that the arcs of these events match a connected component of arcs adjacent along $\mathcal{V}$. The* top *and* bottom *arcs of a block are defined with respect to the position of the arcs in $\mathcal{V}$. The block is termed a* crossing (respectively tangency) *block if all its events are crossing (respectively tangency) events.*

Moreover, the upper and lower bounding *arcs of a block are the arcs of $\mathcal{V}$ located above and below its top and bottom arcs.*

Notice the bounding arcs of a block always exist since the poles are used as sentinels in $\mathcal{V}$. This definition is illustrated on Fig. 3.4. Given the list $\mathcal{CT}$ of a normal event site, an important operation (required by algorithm `break_adjacencies` to maintain Invariant 1, but also algorithm `handle_event_site` to reverse arcs along $\mathcal{V}$, and to detect new adjacency between arcs), consists of finding its tangency and crossing blocks. The corresponding algorithm, `find_CT_block_bounds`, consists of two steps. First, thanks to Invariant 1, the block $\mathcal{CT}$ is retrieved by chaining the events, as each arc but the bounding ones appears in exactly two events. Second, from the $\mathcal{CT}$ block, the crossing and tangency blocks are defined by the maximal sequences of events of a given type (a crossing or tangency tag is stored in each intersection event).

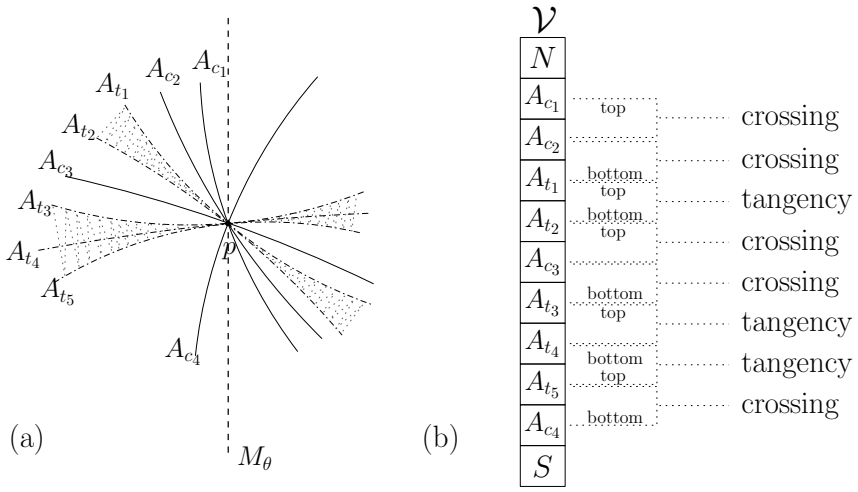


FIG. 3.4 – (a) A set of events can define many blocks by considering its partition into subsets of events of same type. In the event site corresponding to point p, there are: five crossing events: $(A_{c_1} \cap A_{c_2})$, $(A_{c_2} \cap A_{t_1})$, $(A_{t_2} \cap A_{c_3})$, $(A_{c_3} \cap A_{t_3})$ and $(A_{t_5} \cap A_{c_4})$; three tangency events: $(A_{t_1} \cap A_{t_2})$, $(A_{t_3} \cap A_{t_4})$ and $(A_{t_4} \cap A_{t_5})$. There are two tangency blocks defined by the tangency events: the connected components whose arcs are $[A_{t_1} A_{t_2}]$ and $[A_{t_3} A_{t_4} A_{t_5}]$, and three blocks defined by crossing events whose connected components of arcs are $[A_{c_1} A_{c_2} A_{t_1}]$, $[A_{t_2} A_{c_3} A_{t_3}]$ and $[A_{t_5} A_{c_4}]$. $\mathcal{CT}$ defines the block $[A_{c_1} A_{c_2} A_{t_1} A_{t_2} A_{c_3} A_{t_3} A_{t_4} A_{t_5} A_{c_4}]$ (b) Retrieving the blocks using the arcs stored in events, and the crossing/tangency tag.

Remark 2 *We assumed each intersection event effectively stores the two arcs involved. In degenerate cases i.e. when several arcs go through the same event point, this naive implementation duplicates arcs common to two events. But since adjacent arcs are sorted along $\mathcal{V}$, one can reduce an intersection event to the highest arc of the pair, the second one being retrieved, if necessary, from $\mathcal{V}$—both arcs being consecutive in $\mathcal{V}$.*

3.3.3 Maintaining a Linear Size Event Queue

Equipped with the notion of block, we present algorithm `break_adjacencies` to maintain Invariant 1. An intersection event is termed *valid* if its associated arcs are in $\mathcal{V}$ and are adjacent along $\mathcal{V}$. We wish to keep in each normal event site of $\mathcal{E}$ valid intersection events only. Assuming by induction that all intersection events stored in normal event site of $\mathcal{E}$ are valid before processing the top event site e, we need to remove from event sites of $\mathcal{E}$ events which get non-valid upon processing e.

Summary:

The algorithm `break_adjacencies` removes all non-valid crossing and tangency events, and thus maintains Invariant 1. The main idea is to take into account the structure of an event site, and to consider all possible cases (illustrated on Fig. 3.5) to remove all intersection events already detected associated to a pair of arc no longer adjacent in $\mathcal{V}$.

Remark 3 *During the course of the algorithm, two arcs from two intersecting circles yield one or two events: two if they intersect transversely twice and no event has been processed, and one otherwise. Upon adjacency loss, to avoid calling the numerical predicates required to locate the event site of the intersection event to be removed, we use a map (its size is always smaller than or equal to that of $\mathcal{E}$) associating to each pair of arcs adjacent along $\mathcal{V}$ the event site containing the corresponding event(s).*

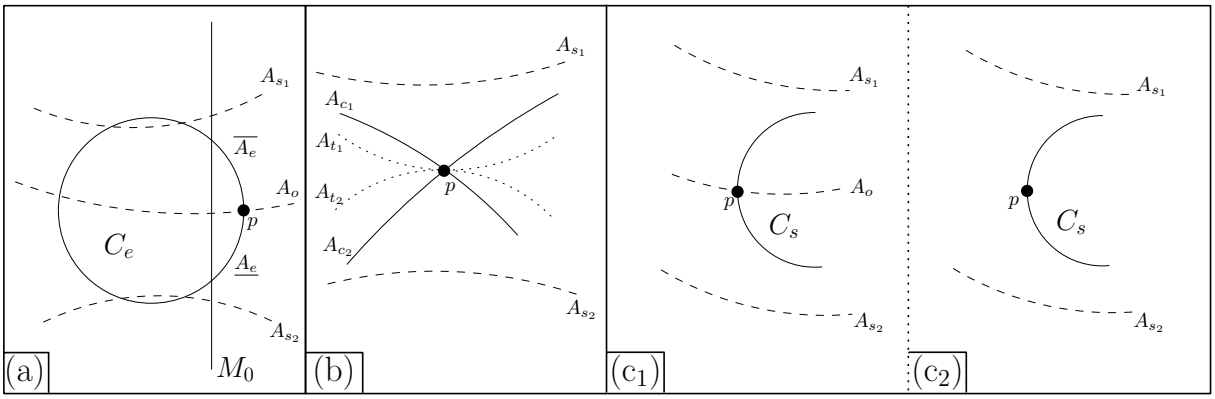


FIG. 3.5 – Breaking adjacencies between arcs. Points associated to events are represented by black bullet. (a) When circle C_e intersected by M_0 ends, we remove intersection events of $(A_{s_1}, \overline{A_e})$, $(\underline{A_e}, A_{s_2})$ and $(\overline{A_e}, A_0)$; (b) Processing the event site corresponding to p breaks adjacency between pairs (A_{s_1}, A_{c_1}), (A_{c_1}, A_{t_1}), (A_{t_2}, A_{c_2}) and (A_{c_2}, A_{s_2})
(c_1) Inserting circle C_s breaks adjacency between pairs (A_{s_1}, A_o) and (A_o, A_{s_2}); (c_2) Inserting circle C_s breaks adjacency between of (A_{s_1}, A_{s_2}).

All the details:

We proceed here with a enumeration of all possible cases to maintain invariant 1.

Normal event sites To describe algorithm `break_adjacencies`, we consider and enumerate all possible cases where adjacency is lost, which includes the following two steps.
▷ **1.** The first step, illustrated on Fig. 3.5, deals with the preservation of adjacencies between the top and bottom arcs of the $\mathcal{CT} \cup \mathcal{F}$ block, and its bounding arcs. This step features three exclusive cases:
(a). $\mathcal{F} \neq \emptyset$. Function `handle_event_site` removes from $\mathcal{V}$ arcs associated to events of $\mathcal{F}$. Note that if a circle C is intersected by M_0, then an intersection event involving arcs of this circle may have been detected but not processed. Let C^+ and C^- the circle of largest and smallest radius associated to end events of $\mathcal{F}$, and $\overline{A^+}/\underline{A^+}$ and $\overline{A^-}/\underline{A^-}$ associated upper/lower arcs.
– if $C^+ \cap M_{0^+} \neq \emptyset$ remove from $\mathcal{E}$, if exists, any intersection event between $\overline{A^+}$ ($\underline{A^+}$) and its upper neighbor (lower neighbor) in $\mathcal{V}$
– if $C^- \cap M_{0^+} \neq \emptyset$ remove from $\mathcal{E}$, if exists, any intersection event between $\overline{A^-}$ ($\underline{A^-}$) and its lower neighbor (upper neighbor) in $\mathcal{V}$
(b). $\mathcal{F} = \emptyset$ **and** $\big[(\mathcal{S} \neq \emptyset$ **and** $\mathcal{CT} \neq \emptyset)$ **or** $(\mathcal{S} = \emptyset$ **and** $\mathcal{CT} \neq \emptyset)\big]$. As the top (bottom) arc of the $\mathcal{CT}$ block loses adjacency with the upper (lower) bounding arc, such intersection(s), if any, are removed from $\mathcal{E}$.
(c). $\mathcal{F} = \emptyset$ **and** $\mathcal{S} \neq \emptyset$ **and** $\mathcal{CT} = \emptyset$. Arcs associated to the inserted circle of largest radius C_s may break adjacencies as follows:

1. if there exists an arc passing through the start point of C_s, we remove from $\mathcal{E}$, if exists, any intersection event between this arc and the upper (the lower) neighbor in $\mathcal{V}$ of $\overline{A_s}$ ($\underline{A_s}$).
2. if arcs of C_s are inserted between two arcs, we remove from $\mathcal{E}$, if exists, any intersection event between these two arcs.

▷ **2.** The second step takes care of adjacency loss between arcs reversed in the $\mathcal{CT}$ block, when $\mathcal{T} \neq \emptyset$ and $\mathcal{C} \neq \emptyset$. Indeed, as the bounding arcs of each tangency block of $\mathcal{CT}$ change upon processing the event, any intersection between such an arc and the top or bottom arc of the block must be removed from $\mathcal{E}$. One can refer to Fig. 3.4 for an illustration.

The completeness of algorithm `break_adjacencies` comes from the fact that all cases $\{\mathcal{F} \neq \emptyset, \mathcal{F} = \emptyset\} \times \{\mathcal{S} = \emptyset, \mathcal{S} \neq \emptyset\} \times \{\mathcal{CT} = \emptyset, \mathcal{CT} \neq \emptyset\}$ are covered (case $\mathcal{CT} \cup \mathcal{F} \cup \mathcal{S} = \emptyset$ not relevant).

(Bi-)polar event sites The only relevant case is that of an end event site of a polar circle intersected by M_0: we remove from $\mathcal{E}$, if any, the intersection event between the arc corresponding to the polar circle and its neighbor in $\mathcal{V}$ which is not a pole.

3.3.4 Handling Event Sites

We explain how to handle normal, as well as polar and bipolar, event sites.

Summary:

The algorithm `handle_event_site` detects all intersection points corresponding to arcs getting adjacent along $\mathcal{V}$ after handling an event. Algorithm `handle_polar_bipolar_event_site` takes care of intersections detected when a polar circle starts, and more generally of intersections involving a bipolar circle. Using the structure of an event site to perform the update of $\mathcal{V}$, this consists in inserting to (respectively removing from) $\mathcal{V}$ arcs starting (respectively ending), and reverting order in $\mathcal{V}$ of arcs intersecting transversally. Intersections revealed by new adjacencies in $\mathcal{V}$ are inserted into $\mathcal{E}$.

The pseudo-code of `handle_event_site` is given in Algorithm 2 in Appendix 3.6. The topological parts of `handle_polar_bipolar_event_site` are given in Appendix 4.4.3.

All the details:

Algorithm `handle_event_site` This algorithm processes the three lists of events associated to a normal event site. To exploit the relative position of circles at the associated event point, two variables *arc_sup* and *arc_inf* are used to record the nodes of $\mathcal{V}$ holding the arcs bounding the several blocks encountered. More precisely:
▷ **1.** First, arcs ending (list $\mathcal{F}$) are removed, a process from which the variables *arc_sup* and *arc_inf* are initialized so as to bound all the arcs from events of the block;
▷ **2.** Second, arcs starting (list $\mathcal{S}$) are iteratively inserted in-between *arc_sup* and *arc_inf*, which are (initialized if necessary, and) maintained along the process;
▷ **3.** Third, arcs corresponding to crossings and tangencies are reversed.

The operations just listed are accompanied by the appropriate intersection tests, so as to update $\mathcal{E}$. We proceed by analyzing cases where two arcs get adjacent along $\mathcal{V}$. We describe intersection tests to perform, each time a configuration matches one of the following cases:
– **If $\mathcal{F} \neq \emptyset$ and $\mathcal{S} \cup \mathcal{CT} = \emptyset$**: If there exists only one arc passing through the end points, this arc becomes adjacent to the two arcs bounding the circles ending, after removing its arcs from $\mathcal{V}$. Else the two arcs bounding the circles ending become adjacent, after removing its arcs from $\mathcal{V}$.
– **If $\mathcal{S} \neq \emptyset$**: After insertion, the upper (respectively lower) arc of the circle of largest radius starting becomes adjacent with the arc above (respectively below) it in $\mathcal{V}$. Having inserted the upper (respectively lower) arc of the circle of smallest radius, it becomes adjacent to the arc below (respectively above) it in $\mathcal{V}$.
– **If $\mathcal{C} \neq \emptyset$ and $\mathcal{S} = \emptyset$**: The top and bottom arcs of the block of arcs defined by $\mathcal{CT}$ after reversal get adjacent to the bounding arcs of the block defined by $\mathcal{CT}$.
– **If $\mathcal{C} \neq \emptyset$**: The top and bottom arcs of each tangency block of $\mathcal{CT}$ after reversal get adjacent to the bounding arcs of that block. Note that if the top (respectively bottom) arc of $\mathcal{CT}$ block is also a top (respectively bottom) arc of one $\mathcal{T}$ block, the intersection test involving this arc is not performed again.

The cases presented cover all possible situations, and no intersection test is performed twice.

Algorithm `handle_polar_bipolar_event_site` Consider a polar event site. To handle such a start (respectively end) event, we have to insert in (respectively remove from) $\mathcal{V}$ the associated arc close to the correct pole. Consequently, the intersection test is performed after insertion of arc close to the corresponding pole, between this arc and its only relevant neighbor in $\mathcal{V}$.

Bipolar event sites do not induce any insertion of arc in $\mathcal{V}$, and thus do not trigger any intersection test. As describe in section 4.4.3, singular points on such circles are handled on the fly, intersecting active arcs.

3.4 Handling $\mathcal{V}$

This section describes the three operations underwent by the vertical ordering $\mathcal{V}$: its initialization; the insertion/deletion of arcs when start/end events are encountered; the reversal operations induced by intersection events.

3.4.1 Initializing $\mathcal{V}$

To initialize $\mathcal{V}$, we first collect among relevant circles (normal, threaded, polar) those whose intersection with M_0^+ does not reduce to a pole. The corresponding arcs are sorted that is we find the vertical ordering of the arcs at $\theta = 0^+$. For two arcs going through distinct intersection points with M_0, the ordering is that of the intersection points along M_0; for two arcs going through the same point of M_0, we resort to a first order calculation if the tangents to the circles at that point are distinct, or a second order calculation in case of tangent arcs (See section 5.1.3 for the details).

All pairs of adjacent arcs in $\mathcal{V}$ are tested for intersection, and the corresponding events inserted into $\mathcal{E}$.

3.4.2 Inserting Starting Arcs into $\mathcal{V}$

First observe that only normal circles trigger non-trivial insertions of arcs into $\mathcal{V}$: the arc of a north (respectively south) polar circle comes right after (respectively before) its pole; bipolar circles are processed on the fly at the event sites without any update of $\mathcal{V}$.

The key step in inserting arcs of normal circles associated to start events of the list $\mathcal{S}$ consists of locating the start point p of the circle of largest radius C_s among arcs in $\mathcal{V}$. (If $\mathcal{F} \neq \emptyset$, this operation is superfluous.) Then, using the sortedness of $\mathcal{S}$, arcs of circles with smaller radii are consecutively inserted closed to arcs of the last circle processed.
To locate p and since $\mathcal{V}$ is vertically sorted, we wish to run a binary search, which requires stating whether p is located below/above/on a given arc of $\mathcal{V}$. This latter predicate is developed in section 5.3.

If p is on no arc, the upper and lower arcs are inserted into $\mathcal{V}$ where indicated by the binary search. Otherwise, if p lies on a least one arc, it is easily seen that its circle and C_s are transverse. (By assumption $\mathcal{F} = \emptyset$ so that no arc is ending at p; moreover C_s is assumed to be the normal circle of largest radius starting at p.) Therefore, the upper and lower arcs are respectively inserted above and below the collection of arcs point p lies on.

3.4.3 Intersection Events and Arcs Reversing

In all generality, an event site features crossing and tangency events. The maintenance of $\mathcal{V}$ once an event site has been passed requires exchanging the order of the arcs in blocks defined by crossing events of the event site, w.r.t. blocks defined by tangency events. We do so in two steps, by first reversing the arcs involved in the block defined by $\mathcal{CT}$, and second by reversing again the the order of arcs involved in each tangency block (this strategy is similar to that describe in [24]). Since we use a tree for $\mathcal{V}$, reversing a set of consecutive arcs can be done by re-writing the values of the nodes holding the arcs involved—which avoids any insertion or deletion in the tree.

3.5 Correctness and Complexity Analysis

To analyze the algorithm complexity, we shall need considerations on the structure of the arrangement. A vertex of this arrangement is either a singular point, or a non-degenerate critical point. Denoting n the number of circles, and k (respectively v) the number of singular points (respectively vertices), we have $v = O(n + k)$. An edge is a circular arc in-between two vertices. A face is a region of S_0 whose interior is connected, and which is bounded by circular arcs and singular points. Some difficulties arise if the 1-skeleton of the arrangement is not connected. We formally define holes with respect to faces as follows:

Definition 10 *Consider a simply connected region R consisting of the union of one or several faces of the arrangement. If the 1-skeleton of R is not connected, the* principal 1-skeleton *is the connected component containing the boundary of R.*

Consider a non-simply connected face f bounded by several cycles, and let c_0 be one of these cycles. We define the support *R of face f with respect to c_0 as the simply connected region of S_0 bounded by cycle c_0. Moreover, let $dom(f)$ stand for the geometric domain defined by face f. The connected components of $R \backslash dom(f)$ are* holes *in the support of face f.*

These notions are illustrated on Fig. 3.6. Notice a hole is a simply connected region consisting of the union of a collection of faces.

Lemma 2 *Denote v and e the number of vertices and edges of an arrangement of n circles on the sphere, and let l stand for the number of faces bounded by exactly two edges. One has $e \leq 3(v-1)+l$.*

Proof. We first introduce a hierarchical representation of the arrangement, based on the concepts of Def. 10. Given an arbitrary edge of the arrangement, let K_0 be the connected component of the 1-skeleton of the arrangement containing this edge. The graph K_0 defines a decomposition D_0 of the sphere into topological disks—simply connected faces of the arrangement or their supports induced by K_0. For each such disk, we can construct a tree as follows: one leaf corresponds to a simply connected region consisting of one (or several) simply connected face(s); one internal node corresponds to the support of a non-simply connected face of the arrangement. See Fig. 3.6 for an illustration.

The proof consists of applying Euler's relationship to simply connected regions (denoted SCR for the sake of conciseness in this proof), namely disks of D_0, and holes recursively found in traveling down the tree decomposing each such disk.

Let V_s, E_s, F_s the numbers of vertices, edges and SCR of the decomposition D_0. One has $F_s = F_s^{(3)} + L_s$, with $F_s^{(3)}$ (respectively L_s) the number of SCR bounded by at least (respectively exactly two) three edges. As an edge bounds two SCR, we have $2E_s \geq 3F_s^{(3)} + 2L_s$. But from the Euler characteristic of the sphere, we get $V_s - E_s + F_s = V_s - E_s + F_s^{(3)} + L_s = 2$. Whence

$$E_s \leq 3(V_s - 2) + L_s. \tag{3.2}$$

Consider now a SCR of D_0 featuring holes. We apply Euler's relationship to the decomposition of each such hole by its principal 1-skeleton. To do so, some care is in order as the SCR corresponding to the support of the face containing the hole has been accounted for. For a given hole whose number of vertices, edges and SCR are also denoted V_i, E_i, F_i, let T_i be the number of edges bounding the hole. Counting edges over faces of the decomposition of the hole yields $2E_i - T_i \geq 3F_i^{(3)} + 2L_i$. As, $V_i - E_i + F_i^{(3)} + L_i = 1$, we get

$$E_i \leq 3(V_i - 1) - T_i + L_i < 3(V_i - 1) + L_i. \tag{3.3}$$

The analysis just carried out for a hole is valid at every level of the tree decomposing one SCR of D_0. Moreover, the 1-skeletons involved in the proof of Equations (3.2) and (3.3) are independent and form a partition of all vertices and edges of the arrangement, that is, summing over all connected components, we get $v = V_s + \sum_i V_i$, $l = L_s + \sum_i L_i$. Therefore, summing the above two inequalities completes the proof. □

We can finally state the main theorem. Notice its complexity involves the so-called *lenses* (convex faces) and *lunes* (concave faces) determined by a family of circles. For n circles of arbitrary radii in the plane, this number is known to be $O(n^{3/2+t})$, for any $t > 0$, where the constant of proportionality depends on t [9]. Notice that in non-degenerate cases, since a pair of circles intersecting defines one lens and two lunes, l is $O(k)$. The time complexity of our implementation, given in the following theorem, thus matches that given in the original implementation [23].

Theorem 2 *Algorithm 1 correctly reports all the singular points of a family of n circles on a sphere. Denoting k the number of such points, the algorithm uses $O(n)$ storage and has $O((n+k+l)\log n)$ complexity.*

Proof. **Correctness.** Following the terminology of Fig. 3.3, we establish that Algorithm 1 reports intersection points, all degenerate tangency points, and all degenerate crossing points.

For the first class, the correctness is similar to that of the classical BO algorithm. More precisely, any intersection points is reported, since we detect an intersection events between two arcs when they get adjacent in $\mathcal{V}$. Second, a degenerate tangency is detected upon insertion of a start event or an end event into an event site of $\mathcal{E}$, as the corresponding event points are identical. Third, for degenerate crossings, we distinguish degeneracies associated to a start and an end critical point. For the former, the crossing is detected when the two circulars arcs are inserted into $\mathcal{V}$, as the start point is found to be on a circular arc. For the latter, for an event site with $\mathcal{F} \neq \emptyset$ and $\mathcal{CT} = \emptyset$, when removing the two arcs associated to the circle of smallest radius, the fact these arcs are not adjacent in $\mathcal{V}$ witnesses the degenerate crossing.

Memory requirements. The vertical ordering requires linear storage. The event queue also has linear size since Algorithm `break_adjacencies` makes sure only events corresponding to arcs incident along $\mathcal{V}$ are stored.

Time complexity. To analyze the complexity, let us consider the initialization, and the overall cost of the **while** loop over the queue $\mathcal{E}$. Before doing so, a comment is in order with respect to the insertion of an event into $\mathcal{E}$. Such an insertion indeed requires looking for an event site in $\mathcal{E}$—and creating one if necessary. But for a normal critical event, the insertion into the sorted list $\mathcal{S}$ or $\mathcal{F}$, whose size is $O(n)$, has complexity $O(\log n)$.

To begin with, the algorithm classifies the circles, which incurs a linear cost. As there are $O(n)$ normal circles, inserting these critical events into $\mathcal{E}$ requires $O(n \log n)$. Similarly, inserting at most $2n$ arcs into $\mathcal{V}$ costs $O(n \log n)$.

Consider now the **while** loop over queue $\mathcal{E}$. For each event processed, denote m_p the number of arcs passing through the corresponding point. The following steps need to be accounted for:
– removing from $\mathcal{E}$ the intersection points corresponding to arcs no longer adjacent in $\mathcal{V}$: $O(m_p \log n)$;
– removing/inserting ending/starting arcs from/into $\mathcal{V}$: $O(m_p \log n)$;
– finding bounding arcs of the block $\mathcal{CT}$: $O(m_p \log m_p)$;
– detecting and inserting into $\mathcal{E}$ new intersection events corresponding to consecutive arcs along $\mathcal{V}$: $O(m_p \log n)$;
– exchanging the position in $\mathcal{V}$ of arcs intersecting at the event point: $O(m_p)$.

The cost of one iteration is therefore $O(m_p \log n)$, so that the overall costs of iterations is $O(M \log n)$, with $M = \sum_{p \text{ an event site}} m_p$. But $M = 2e$ with e the number of edges. From lemma 2, we have $M = O(v + l)$, and since $v = O(n + k)$, we get $M = (n + k + l)$. Adding up the cost of the initialization and of the iterations yields the overall complexity. □

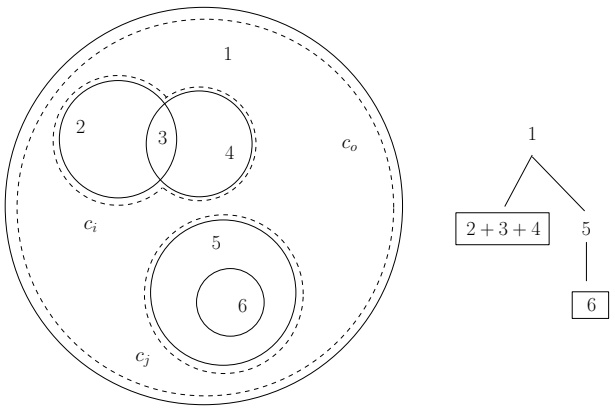


FIG. 3.6 – Six faces decomposing the disk bounded by c_0. The cycles bounding face 1 are c_o, c_i, c_j. The support of face 1 with respect to cycle c_0 is the disk itself, and this support contains two holes bounded by cycles c_i and c_j. Cycle c_i bounds a hole with 3 faces, while c_j bounds face 5 which contains a hole.

3.6 Appendix: Pseudo-code

We shall use the following conventions: *variables* are written in italic, while `functions` are written in typewriter style. Functions prefixed `topo_` are dedicated to the construction of the arrangement and are detailed in chapter 4. In the pseudo-code of Algorithm 2, the node of $\mathcal{V}$ holding arc A_i is denoted $v[A_i]$. Reciprocally, the arc associated to a node say x of $\mathcal{V}$ is denoted $*x$. The function `Above` (respectively `Below`) returns the node holding arc above (respectively below) a in $\mathcal{V}$ (i.e. previous and next node).

This section provides the pseudo-code of the main functions:

- Function `classify_circles` classifies circles according to Def. 1.
- Function `topo_init_arrangement` initializes the HDS used to store the arrangement.
- Function `break_adjacencies` removes from $\mathcal{E}$ intersection events which do not correspond to two adjacent arcs in $\mathcal{V}$ upon processing of an event site.
- Function `handle_event_site` maintains $\mathcal{E}$ and $\mathcal{V}$, but also calls topological routines to manage half-edges at the event point: `topo_handle_left`, `topo_handle_right` and `topo_handle_above_below`.
- Function `handle_polar_bipolar_event_site` updates $\mathcal{E}$ and $\mathcal{V}$ for circles passing by a pole and has a topological part (described in Appendix 4.4.3) consisting in managing half-edges anchored at a pole.
- Function `topo_merge_virtual_faces` completes the construction of the arrangement once the event queue has been exhausted.

Algorithm 1 Bentley-Ottmann: pseudo-code

```
1: classify_circles {find circle types }
2: Initialize vertical order V
3: Initialize event queue E
4: topo_init_arrangement
5: while (E ≠ ∅) do
6:    e=E.pop
7:    break_adjacencies(e)
8:    if (e is a normal event) then
9:       handle_event_site(e)
10:   else
11:      handle_polar_bipolar_event_site(e)
12:   end if
13: end while
14: topo_merge_virtual_faces
```

Algorithm 2 handle_event_site

1: arc_sup = **NULL**
2: arc_inf = **NULL**
3: `topo_handle_left`
4: {Remove arcs ending, if any, by increasing radius}
5: **if** ($\mathcal{F} \neq \emptyset$) **then**
6: **for** each circle C_{e_i} of the end point associated to event in $\mathcal{F}$ **do**
7: Remove from $\mathcal{V}$ the arcs $\underline{A_{e_i}}$ and $\overline{A_{e_i}}$
8: **end for**
9: Record into arc_sup and arc_inf, the nodes of the two arcs below and above the circle of largest radius ending, if exists
10: **if** ($\mathcal{S} = \emptyset$ and $\mathcal{CT} = \emptyset$) **then**
11: **if** ($arc\_sup \neq$ `Above`(arc_inf)) **then**
12: Test intersections of $*arc\_sup$ and $*$`Below`(arc_sup), and possibly update $\mathcal{E}$
13: **end if**
14: Test intersections of $*arc\_inf$ and $*$`Above`(arc_inf), and possibly update $\mathcal{E}$
15: **end if**
16: **end if**
17: {Insert arcs starting (s_0 is the circle of greatest radius starting), if any.}
18: **if** ($\mathcal{S} \neq \emptyset$) **then**
19: **if** (arc_sup = **NULL**) **then**
20: Insert arcs $\underline{A_{s_0}}$ and $\overline{A_{s_0}}$ into $\mathcal{V}$
21: **else**
22: Insert arc $\overline{A_{s_0}}$ below arc_sup
23: Insert arc $\underline{A_{s_0}}$ above arc_inf
24: **end if**
25: $arc\_sup \leftarrow v[\overline{A_{s_0}}]$
26: $arc\_inf \leftarrow v[\underline{A_{s_0}}]$
27: Test intersections of $*arc\_sup$ and $*$`Above`(arc_sup), and possibly update $\mathcal{E}$
28: Test intersections of $*arc\_inf$ and $*$`Below`(arc_inf), and possibly update $\mathcal{E}$
29: **for** each remaining start event in $\mathcal{S}$ whose circle is C_{s_i} **do**
30: Insert arc $\overline{A_{s_i}}$ below arc_sup; $arc\_sup \leftarrow v[\overline{A_{s_i}}]$
31: Insert arc $\underline{A_{s_i}}$ above arc_inf; $arc\_inf \leftarrow v[\underline{A_{s_i}}]$
32: **end for**
33: **if** ($\mathcal{CT} = \emptyset$) **then**
34: `topo_handle_right`
35: **if** ($arc\_sup \neq$ `Above`(arc_inf)) **then**
36: Test intersections of $*arc\_sup$ and $*$`Below`(arc_sup), and possibly update $\mathcal{E}$
37: Test intersections of $*arc\_inf$ and $*$`Above`(arc_inf), and possibly update $\mathcal{E}$
38: **end if**
39: **end if**
40: **end if**
41: {Process tangent arcs or arcs intersecting}
42: **if** ($\mathcal{CT} \neq \emptyset$) **then**
43: `find_CT_block_bounds`
44: { if arc_sup and arc_inf are **NULL**, they are respectively updated to the upper and lower bounding arcs of the block defined by $\mathcal{CT}$ }
45: Reverse block of all arcs defined by $\mathcal{CT}$ in-between arc_sup and arc_inf
46: **for** (each block **B** defined by tangency events of $\mathcal{CT}$) **do**
47: Reverse block **B**
48: **end for**
49: `topo_handle_right`
50: Test intersections between top,bottom and lower,upper of bounding arcs of the blocks
51: Update $\mathcal{E}$ accordingly
52: **end if**
53: `topo_handle_above_below`

Chapitre 4

One Pass Construction of the Arrangement of Circles on a Sphere, with Application to Covering Lists

Chapter Overview

This chapter presents an additional layer on the top of the BO algorithm presented in chapter 3 allowing one to construct the arrangement of circles on a sphere and to compute a compact representation of the covering lists called the covering tree.

Section 4.2, is dedicated to the construction of the half-edge data structure (HDS) storing the arrangement, a construction performed on the fly during the sweep process, and using Union-Find. In section 4.3, we present a calculation of the covering tree of the arrangement that encodes the covering lists of each faces. The Appendix 4.4 provides proofs of the time and space complexities related to the two constructions aforementioned, together with more details regarding topological operations at the poles.

This chapter corresponds to a paper accepted for publication in the journal *Computational Geometry: Theory and Applications*. This is a joint work with Frédéric Cazals.

4.1 Definitions

Having introduced the spherical Bentley-Ottmann algorithm in chapter 3, we present a classification of θ-monotone arcs (called arcs in this chapter) and half-edges, to be used in sections 4.2 and 4.3.

Qualifying an Arc w.r.t. Circles: Upper or Lower. A threaded circle is called *north threaded* if its center has a z coordinate larger than or equal to that of the center of S_0, and *south* threaded otherwise. In the same way, a polar circle containing the north pole is called a north polar circle, and south polar circle otherwise. Considering the two arcs of a normal circle, we call them the *upper* and the *lower* arcs w.r.t. the z axis. A north (respectively south) polar circle has a single non-trivial arc, which is *lower* (respectively *upper*). Similarly, we consider that a north (respectively south) threaded circle has a single arc which is *lower* (respectively *upper*).

Qualifying a Half-edge w.r.t. an Arc: Upper or Lower. For all but bipolar circles, to each arc A_i, we associate two active half-edges qualified w.r.t. the unique intersection between the meridian and the arc: The *upper* (respectively *lower*) half-edge associated to A_i is the half-edge leaving to its left the portion of S_0 lying above (respectively below) A_i. We now address particular cases: (i)for a threaded circle which is not intersected by any other circle, at the end of the sweep process, the source and the target of the half-edges associated to its arcs are fixed to a common null vertex (ii)for a bipolar circle, no arc is defined. Half-edges are created on the fly at a bipolar event, so as to handle, if any, intersections with active arcs.

Qualifying a Half-edge w.r.t. a Circle: Inner or Outer. For a circle which is not a great circle, consider the spherical cap of S_0 of smallest area induced by this circle. If a half-edge on this circle induces this cap, it is called *inner*, and *outer* otherwise. For a great circle, we break the tie and call the cap of smallest area that induced by inner half-edges, that is: (i) if the great circle is bipolar, an half-edge is *inner* if it induces the spherical cap swept by M_θ between its θ_S and θ_E associated values, and *outer* otherwise; (ii) if the great circle is threaded, an half-edge is *inner* if it induces the spherical cap containing the north pole, and *outer* otherwise.

The relationship between the qualifiers upper/lower and inner/outer of half-edges is illustrated on Fig. 4.1, and we have:

Observation 1 *The spherical cap of smallest (respectively largest) area bounded by a circle is described by inner (respectively outer) half-edges of the circle. The lower/upper half-edge of an upper (respectively lower) arc of a all but bipolar circle is always inner/outer (respectively outer/inner).*

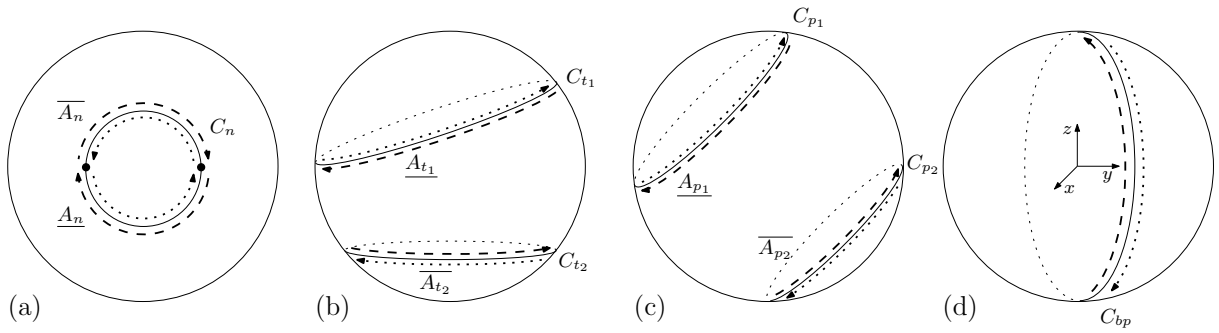


FIG. 4.1 – Qualifying arcs and half-edges: upper and lower arcs are respectively denoted as $\overline{A}$ and $\underline{A}$; inner and outer half-edges are respectively represented using dotted and dashed lines; (a) normal circle C_n and its upper and lower arcs; (b) north threaded circle C_{t_1} defining a lower arc, and south threaded circle C_{t_2} defining an upper arc; (c) north polar circle C_{p_1} defining a lower arc, and south polar circle C_{p_2} defining an upper arc; (d) bipolar circle C_{bp} inducing half-edges (no arc is defined). The θ-coordinate of the start point of C_{bp} lies in $(0; \pi]$.

4.2 Constructing the Arrangement During the Sweep Process

In this section, we develop the topological operations required to construct the arrangement induced by the circles and stored it into an extended half-edge data structure [119] (HDS) possibly featuring holes in faces.

4.2.1 Describing the Arrangement: definitions

Arcs and Half-edges. The vertices of the HDS correspond to event points, its edges are arcs delimited by such vertices, while the faces are the regions of S_0 bounded by vertices and edges. Recall that a face is a two-dimensional region whose interior is connected, and that half-edges are oriented so as to leave the interior of the face to its left. Over the course of the sweep algorithm, an half-edge is created when its *first* vertex is fixed, and remains *active* until its *second* vertex is also fixed. Following classical terminology, notice the first vertex may be the source or the target vertex of the half-edge. In the following, stitching two half-edges should be understood as fixing the next pointer of one half-edge to the second.

Faces and Holes. To describe the arrangement, we use sequences of connected half-edges (SCH), such a sequence being called *closed* if its topology is that of a circle. Two active half-edges associated to arcs in $\mathcal{V}$, with respect to the ordering along $\mathcal{V}$, are termed *adjacent* if their arcs are adjacent in $\mathcal{V}$, and if they bound the same segment along the meridian M_θ. (Out of the four pairs of half-edges associated to two consecutive arcs along $\mathcal{V}$, a single pair corresponds to adjacent half-edges.) A *face* of the arrangement is

represented by a collection of closed SCH. Each such sequence is called a *Connected Component of the Boundary* or CCB. A CCB is oriented and always induces a contractible region on the sphere S_0. The set of all CCB describing a face can be split into two categories: one CCB called *principal* defines the spherical cap containing the face; the remaining ones define *holes* in the face. See Fig. 4.2.

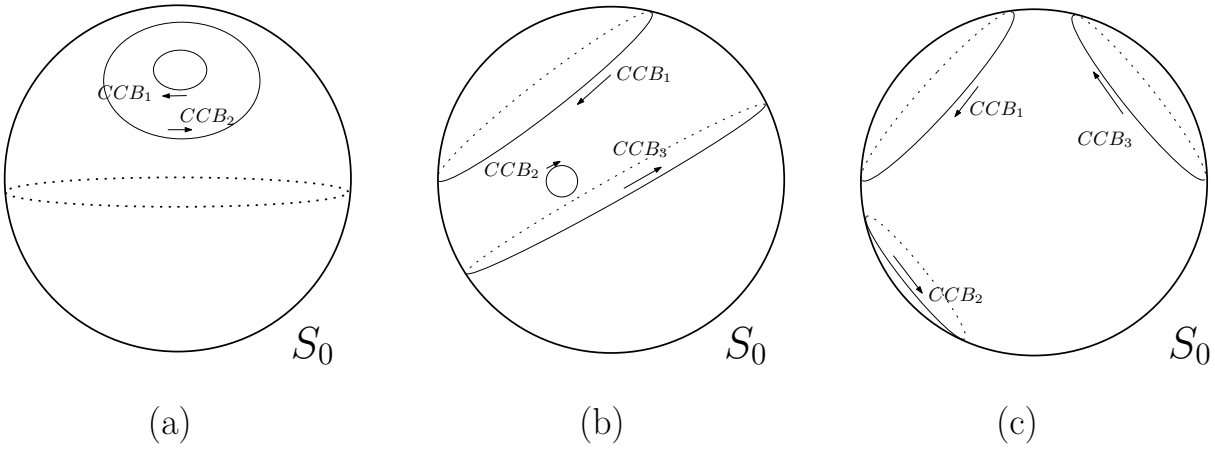


FIG. 4.2 – Defining a face with a set of CCB. Each arrow represents a CCB. In the four cases, only one face is defined by the CCB represented.

4.2.2 Handling Half-edges

The initialization of $\mathcal{V}$ orders the arcs along M_θ for $\theta = 0^+$. To be able to describe the faces cut by M_{0^+}, to each arc in $\mathcal{V}$ after initialization, we attribute a pair of opposite half-edges with one *null* vertex implicitly representing the intersection between M_{0^+} and the corresponding arc. This collection of half-edges is stored in a list H_0. At the end of the sweep process, we have a one-to-one correspondence between half-edges of arcs in $\mathcal{V}$ and the sequence of half-edges in H_0, so that a merge can be performed.

To describe the operations underwent by half-edges at a normal event site, consider the arcs involved in the three lists of a normal event site, in the neighborhood of the corresponding event point, say p. Two types of operations need to be performed. The first type, to the left (respectively to the right) of p, stitches each pair of adjacent half-edges of the arcs involved, before (respectively after) updating $\mathcal{V}$. The second type stitches the half-edges of the highest and lowest arcs along $\mathcal{V}$. These operations are straightforward, and the details are left to the reader. These operations are mentioned using functions prefixed by `topo_` in Algorithm 2 of Appendix 3.6.

Operations involving polar and bipolar circles are also straightforward: Following Def. 8, half-edges incident to a pole are stitched while rotating around that pole; When handling a bipolar event, from the north to the south pole, all active arcs in $\mathcal{V}$ are intersected and their half-edges are updated accordingly. If the bipolar circle goes through a point corresponding to an event site, then the event site is handled at the same time taking into account the half-edges of the bipolar circle. These operations are illustrated in Algorithm 4 of Appendix 4.4.3.

4.2.3 Building the Faces

Building the faces of the arrangement requires two steps, namely creating CCB and creating faces by joining the CCB. To do so, we resort to two independent union-find algorithms.

Creating a CCB. A SCH becomes a CCB whenever stitching two half-edges creates a topological circle. As the intersection between a CCB and the meridian may feature several connected components (see Fig. 4.3), we merge SCH using union-find, which requires endowing each half-edge with a pointer to a master half-edge—called the *CCB master*. Stitching two half-edges is accompanied by a union of their CCB masters (if different), and a CCB appears when two half-edges being stitched already have the same CCB master.

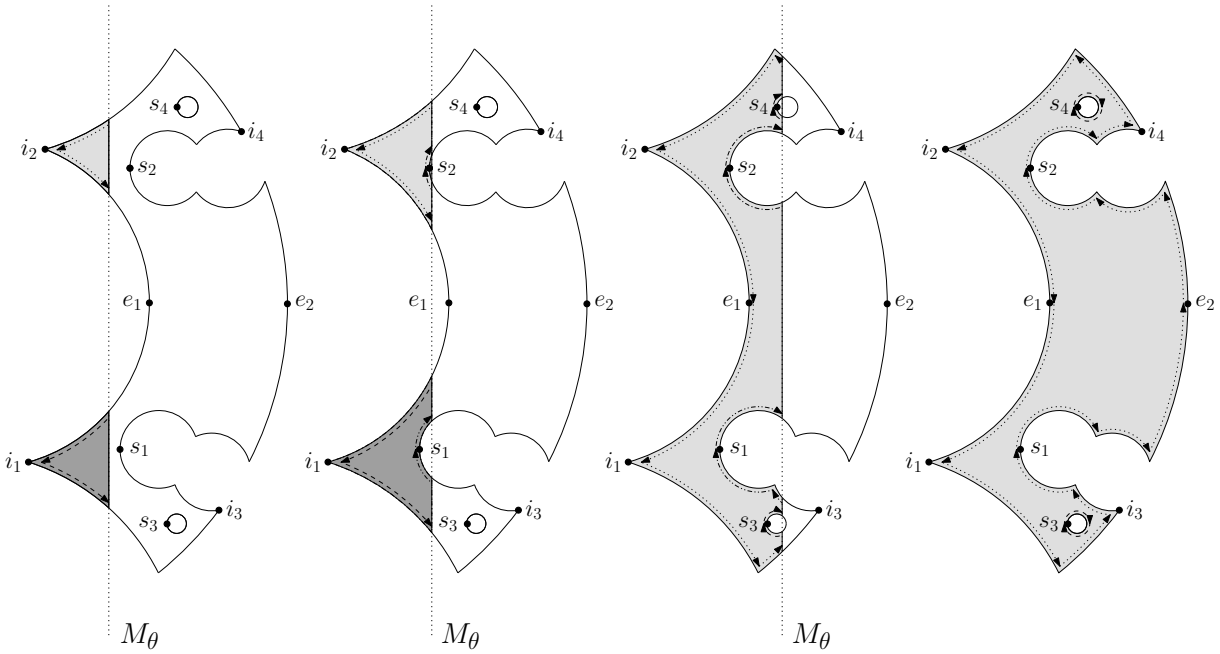


FIG. 4.3 – Illustration of Union-Find: one color per face, one line-style per CCB. Two SCH get created at events corresponding to i_1 and i_2, together with two faces. When the meridian reaches start point s_1, (and s_2) a new SCH is created and it contributes to the definition of an existing face. The handling of the event corresponding to the end point e_1 results on the one hand, and on the other hand to the union of two CCB masters and to the union of two face masters: The three remaining CCB describe a unique face. Thanks to this operation, the CCB started with points s_3 and s_4 also describe that face. Finally, the unions of CCB masters at events associated to i_3 and i_4 allows to detect the closure of the CCB at point e_2.

Creating a Face. A face consists of a principal CCB and of CCB defining holes. We construct faces using union-find on SCH, which requires a union-find pointer called the *face master*. While closing a SCH, the resulting CCB is principal if it is its own face master; if not, the CCB becomes a hole of the face referred to by the face master. Moreover while stitching two half-edges, if the face masters are different, then a union operation is done (see Fig. 4.3). We complete the description by the initialization of the face master.

Whenever two arcs are adjacent in $\mathcal{V}$, a pair of active adjacent half-edges bounds the same face. When creating a new SCH, the SCH either contributes to a face in progress (i.e. an adjacent half-edge already handled can be found), or starts a new face. Difficulties arise if one of the half-edge creating the new SCH is adjacent to the north pole. This particular case is carried out by recording the *face containing the north pole*, denoted $F_N(M_\theta)$—the subscript θ indicates that this face may evolve during the sweep. If no circle is bipolar or north polar, $F_N(M_\theta)$ is set once and does not change over the course of the algorithm. Otherwise, the face $F_N(M_\theta)$ is defined as follows. If the meridian M_θ is not tangent to any circle at the pole, $F_N(M_\theta)$ is defined as the face having the north pole on its boundary and containing an infinitesimal portion of M_θ anchored at the north pole. If meridian M_θ is tangent to a (bi)polar circle, $F_N(M_\theta)$ is undefined. But we denote $F_N(M_{\theta^-})$ (respectively $F_N(M_{\theta^+})$) the face containing the north pole for an infinitesimally smaller (respectively larger) value of θ. See Table 4.1 for initialization and update of $F_N(M_\theta)$.

4.2.4 Complexity Analysis

To conclude, we give the cost of constructing the HDS storing the arrangement. The analysis consists of counting the number of find and union operations used to maintain the topological data structures

Position of M_θ	**Operation(s) at north pole**
Initializing $\mathcal{V}$	$F_N(M_\theta)$ is set to be the face started by the upper hedge of the top arc of $\mathcal{V}$ at $\theta = 0^+$.
Bipolar or a north polar start event	$F_N(M_{\theta^+})$ is the face started by the inner hedge. $F_N(M_{\theta^-})$ is the face described by the outer hedge.
North polar end event	$F_N(M_{\theta^+})$ is the face described by the outer hedge. $F_N(M_\theta^-)$ is the face described by the inner hedge.
Bipolar end event	$F_N(M_{\theta^+})$ is the face started by the outer hedge. $F_N(M_\theta^-)$ is the face described by the inner hedge.
Normal start event or south polar start event	$F_N(M_\theta)$ is set to be the face started by the outer hedge(s) (if it is not already done).

TAB. 4.1 – Initializing and updating the face containing the north pole $F_N(M_\theta)$, and its relatives $F_N(M_\theta^-)$, $F_N(M_\theta^+)$. See text for details. Notice that hedge stands for half-edge.

i.e. SCH/CCB and faces. Denoting α the inverse of Ackermann's function, recall that the complexity of performing M union-find operations on a N elements set, with $M \geq N$, is $\Theta(M\alpha(M,N))$ [184]. $M \leq U$, then $\alpha(M,N) \leq \alpha(U,U)$.
Notation. Upon completion of the sweep, the union-find data structure features f connected components corresponding to the f faces of the arrangement. Denoting h the total number of holes, e the number of edges in the arrangement, n the number of circles and n_0 the number of arcs found in $\mathcal{V}$ at $\theta = 0^+$, the following theorems are proved in Appendix 4.4:

Theorem 3 *Constructing CCB has complexity* $O((e+n_0)\alpha(6e+8n_0, 6e+8n_0))$.

Theorem 4 *Grouping CCB into faces has complexity* $O((f+h)\alpha(f+h+20n, f+h+20n))$.

From the proof of lemma 2, one has:

Observation 2 *The memory space required to store the arrangement as a HDS is* $O(n+k+l)$.

4.3 Reporting Inclusion into Balls

Let B_i be the ball associated to the sphere S_i which generates the intersection circle $C_i = S_0 \cap S_i$. In this section, we describe an algorithm reporting the *covering list* of each face of the arrangement on S_0, that is the list of balls containing it.

4.3.1 Enclosing Balls and Unique Circles

In reporting the covering lists, two difficulties are faced.
First, a number of degeneracies must be accommodated. Notice that (i) if a sphere is tangent to S_0 the intersection reduces to a point, and if S_0 is covered by the associated ball, so are all the faces of the arrangement (ii) if two spheres intersect S_0 along the same circle, and their balls cover (respectively do not cover) the same part of S_0, a face covered by one is covered by the other (respectively is not covered by the other).
Second, the BO algorithm operates under the assumption that all circles are different. To meet this requirement, we sort the m intersecting spheres using a total ordering returning equality when two spheres intersect S_0 along the same circle. While sorting the spheres, each unique intersection circle is endowed with two special balls: the *primary* ball, which is associated to the sphere first defining such a circle; and the *opposite* ball, which is the first to intersect S_0 along the same circle, but is opposite to the primary w.r.t. the plane of the circle. Notice that the opposite ball may not exist. Each primary and opposite ball is attached a list of balls yielding the same intersection circle and covering the same part of S_0. We call these the lists of *friends* in the sequel.

Let us get back to the operator used to sort spheres. Two non-great circles are identical iff they have the same center: We first resort to a lexicographic sorting using intersection circle centers. Two great circles (their center being the center of S_0) are identical iff the centers of the two spheres defining these circles are aligned with the center of S_0—the spheres generating these circles belong to the same pencil. Thus, to distinguish great circles, we use the lexicographic order on a canonical vector describing the pencil of spheres yielding a given circle.

4.3.2 Inclusion into Balls

Outline. While running the sweep process, we construct an implicit encoding of the covering lists. In the following, the description focuses on the primary and opposite balls, as the remaining balls are accessible thanks to the lists of friends.

More precisely, the covering lists are represented by a tree, the *covering tree*. Each node in this tree corresponds to one face of the arrangement, and the edge connecting two nodes corresponds to an arc found in $\mathcal{V}$, stating which primary/opposite ball is added to/removed from the covering list of the father node. Since a face consists of one or several CCB and a CCB is incrementally built from SCH, the principle used to set the covering lists consists of setting one such list for each face master upon creation of the corresponding SCH. This strategy entails two things. First, upon extension at an intersection point of SCH describing a given face, the covering list does not change since the contribution (covering balls) of the corresponding circle has already been taken into account at the creation of the face using a father node. Second, consider the case of the union of two SCH having different face masters: The two face masters must be merged. The consequences are twofold: (i) one of the two covering lists can be discarded; (ii) in the covering tree, the sons of the node suppressed are attached to the node that remains.

Initialization. Let LNB_θ be the list of balls covering the face $F_N(M_\theta)$. This list is initialized at $\theta = 0^+$, as indicated in the first section of Table 4.3.2. The root of the covering tree is precisely LNB_θ at $\theta = 0^+$.

Updates. First, upon creation of a new face started by a new SCH, its covering list is created, using Observation 1, by updating that of its ancestor in the covering tree. This ancestor corresponds to the face pointed by the face master of the upper half-edge of the highest arc along $\mathcal{V}$ involved in the creation of the new SCH.

Second, consider the updates of LNB_θ. For a circle C, let $\texttt{CLP}(C)$ (respectively $\texttt{CSP}(C)$) be the function returning, if any, among the primary and opposite balls, the one Covering the Largest (respectively Smallest) Part of S_0 bounded by circle C. Operator + (respectively -) means that the ball is added to (respectively removed from) LNB_θ. The cases to be accommodated are listed in the second section of Table 4.3.2 and illustrated on Fig. 4.4.

Step		Type of event	Action(s) on $\mathbf{LNB_\theta}$
Filtering spheres	1a	$S \cap S_0 =$ a point and $S_0 \subset B$	$+ B$
	1b	B covers the largest part of S_0	$+ B$
Classifying circles	2a	North polar circle with $\theta_E < \theta_S$	+ $\texttt{CSP}(C)$ and - $\texttt{CLP}(C)$
	2b	North threaded circle	+ $\texttt{CSP}(C)$ and - $\texttt{CLP}(C)$
Handling events	3a	North (bi)polar circle Starting	+ $\texttt{CSP}(C)$ and - $\texttt{CLP}(C)$
	3b	North (bi)polar circle Ending	+ $\texttt{CLP}(C)$ and - $\texttt{CSP}(C)$

TAB. 4.2 – Updating the balls in LNB_θ: initialization and updates. The ball/sphere/circle processed is denoted $B/S/C$. See text for the definition of functions CLP and CSP, and Fig. 4.4 for the details.

At the end of the sweep process, the number of nodes is exactly the number of faces in the arrangement. Moreover, if n stands for the number of intersection circles, the maximum distance between the root and a node is $2n$ as $\mathcal{V}$ never contains more than $2n$ arcs.

An example of such a tree is presented on Fig. 4.5. Consider the case of face 6. This face got started at the start event of circle C_4 using its two inner half-edges. The highest one of the pair is the lower half-edge of the upper arc of C_4. The upper half-edge of that arc describes face 4. Therefore, the node of face 6 is a son of the node of face 4, and since the arc considered is an upper one, the edge between the

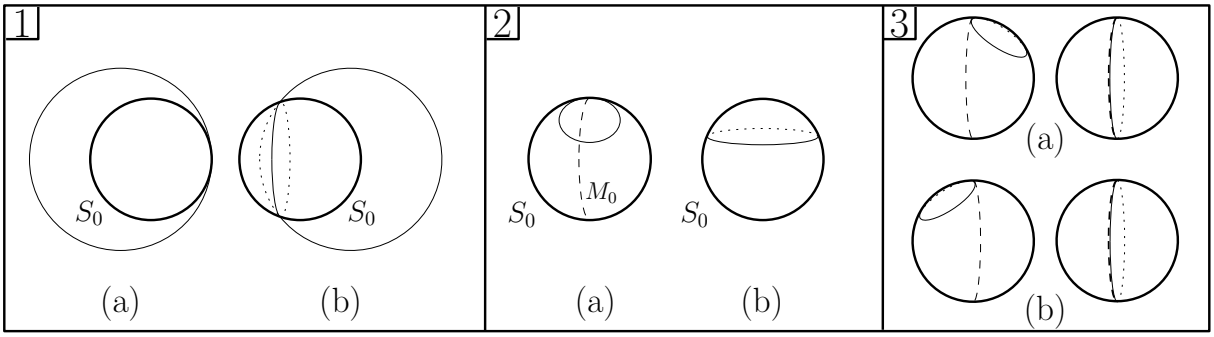


FIG. 4.4 – Updating the balls of LNB_θ: the six cases of Table 4.3.2. 1(a) S_0 is covered by a ball whose sphere is tangent to S_0; 1(b) A sphere intersects S_0, and the associated ball covers the largest part of S_0; 2(a) Meridian M_0 intersects a polar circle. The ball covering the smallest part of S_0 covers $F_N(M_0)$; 2(b) A north threaded circle. The north pole is always (respectively never) covered by a ball covering the smallest (respectively largest) part of S_0; 3(a) The dashed meridian M_θ stopped at a (bi)polar start event: $F_N(M_{\theta^+})/F_N(M_{\theta^-})$ is covered by a ball covering the smallest/largest part of S_0; 3(b) The dashed meridian M_θ stopped at a (bi)polar end event: $F_N(M_{\theta^+})/F_N(M_{\theta^-})$ is not covered by a ball covering the smallest/largest part of S_0.

two nodes indicates that the ball, among primary and opposite, covering the smallest (respectively the largest) part of S_0 according to C_4 must be added (respectively removed).

4.3.3 Complexity Analysis

To analyze the complexity of the implicit encoding, m denotes the number of input balls. The following is proved in Appendix 4.4:

Theorem 5 *Computing the covering tree has complexity $O(f+m)$ time and space. Denote $s_T(F)$ the total number of balls covering face F. Constructing the covering lists for all faces of the arrangement requires $O(\sum_{F\in faces} s_T(F)+f)$ time.*

4.4 Appendix

In this section, we first give the proofs of the three theorems stating the complexity for constructing the HDS and the covering lists. Then, we give more details on the handling of events at a pole.

4.4.1 Proofs of Section 4.2

Proof.[Thm. 3] The e edges of the arrangement yield $2e$ half-edges, while the n_0 arcs intersected at initialization yield $2n_0$ half-edges. The number of half-edges manipulated is thus $2e+2n_0$.

To count the number of operations, we proceed as follows: we count operations required by the initialization at $\theta = 0^+$; next, we enumerate operations on a vertex (of the arrangement) basis; finally, we count operations involved at the merging step at $\theta = 2\pi$.

First, the initialization step requires $2n_0$ *make_set* operations. Second, at a vertex of the arrangement, whenever two half-edges are merged to bound a face, three situations arises, as 2, 1 or 0 half-edges get created. The first case requires two *make_set* and one *union*. The second case requires one *make_set*, one *find*, and at most one *union*. The second case requires two *find* and at most one *union*. In any case, we have at most three operations. A vertex of the arrangement adjacent to m_p edges requires at most $3m_p$ operations, whence a total number of operations bounded by $6e$. Third, the merge steps at $\theta = 2\pi$ requires at most 3 operations for a pair of half-edges, whence $6n_0$ operations at most. Adding up these contributions yields an upper bound of $6e+8n_0$. □

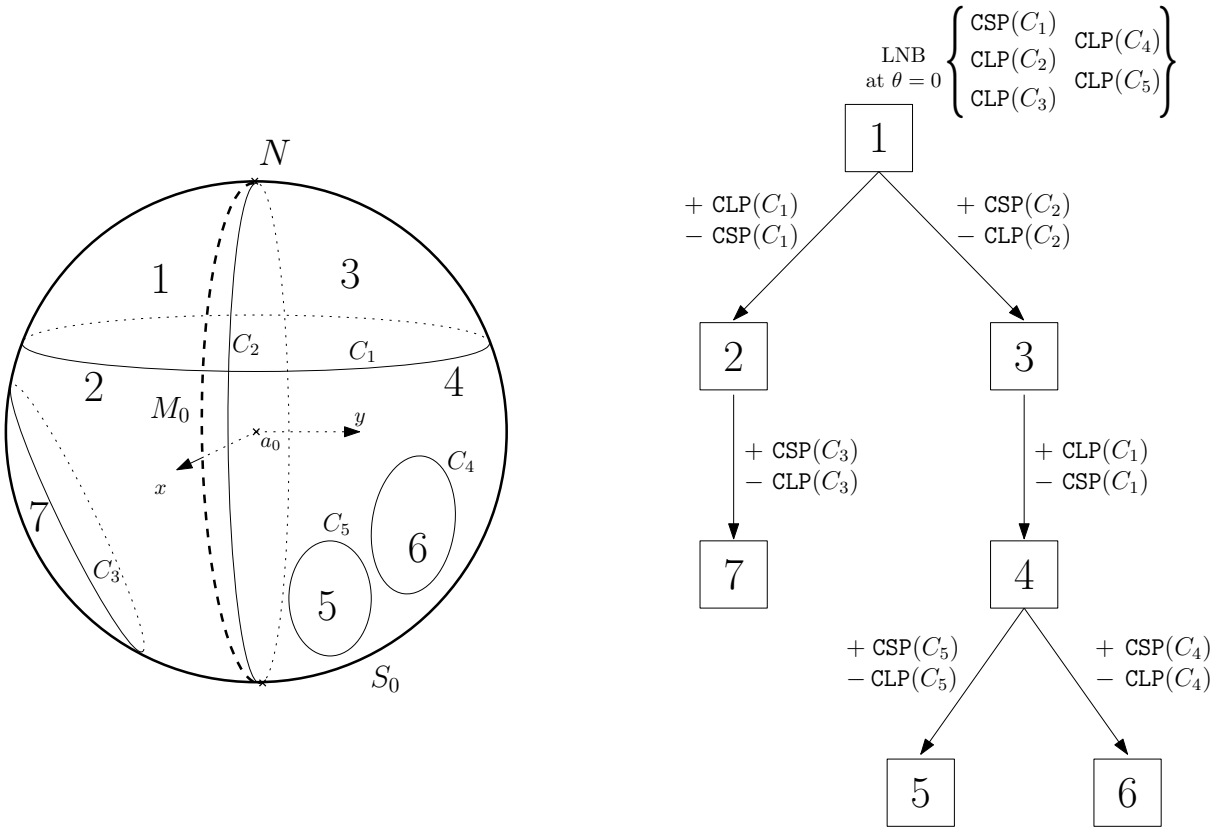


FIG. 4.5 – Implicit encoding of covering lists. Circle C_1 is north threaded, C_2 is bipolar, while C_3, C_4 and C_5 are normal circles. Faces of the arrangement depicted on the left are identified using numbers from 1 to 7. Notice that node associated to face 2 reflects the modification of LNB_θ induced by bipolar circle C_2.

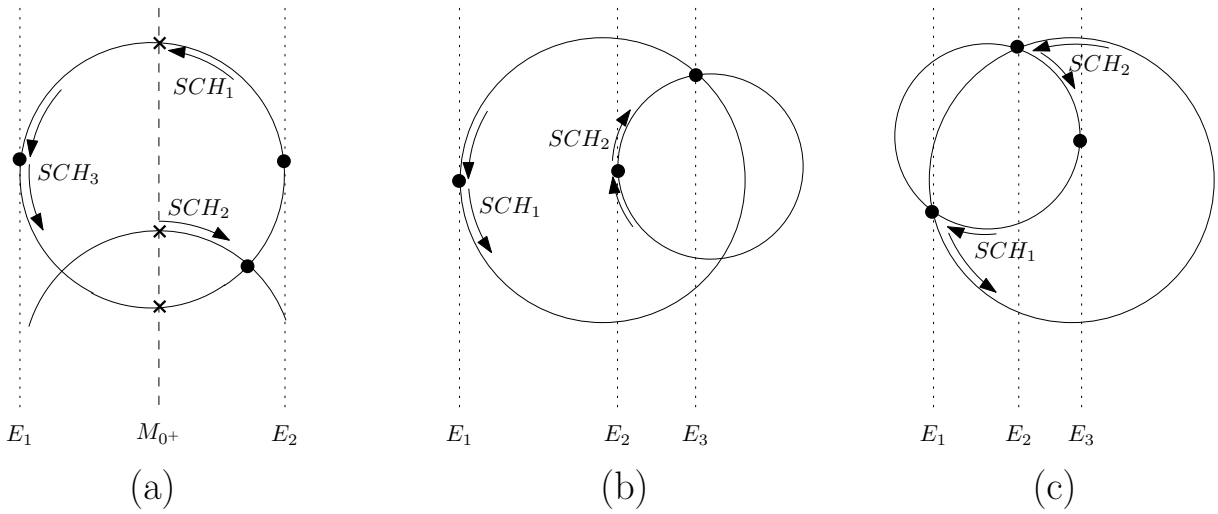


FIG. 4.6 – Counting superfluous SCH. (a) SCH_1 and SCH_2 are created at initialization and merged at E_2; SCH_3 is created at E_1; a single CCB remains at $\theta = 2\pi$ (b, c) SCH_1 (respectively SCH_2) is created at E_1 (respectively E_2); both are merged at E_3

Proof.[Thm. 4] Objects manipulated in this union-find algorithm are SCH. The total number of SCH created is bounded by $f + h + 2n_0 + n + n$. Let us examine the three overheads w.r.t. $f + h$: the first one, $2n_0$, corresponds to the SCH created upon initialization and merged upon completion of the sweep—Fig. 4.6(a); the second one, n, corresponds to the SCH started by the outer half-edges of the circle of largest radius involved in a normal event site featuring only start events—Fig. 4.6(b); the third one, n, corresponds to the SCH that get merged at the end point of the circle of largest radius involved in a normal event site featuring only end events—Fig. 4.6(c).

Apart from the N *make_set* operations, *union* and *find* operations are performed in three cases:
– first, when merging at $\theta = 2\pi$—Fig. 4.6(a). Since adjacent half-edges are always associated to the same face master, we have to do at most two *find* and one *union* operations per face defined, i.e. at most $3(n_0 + 1)$ operations.
– second, at a start point—Fig. 4.6(b). The SCH corresponding to the outer half-edge of the circle of largest radius is attached to a face in progress, which requires one *find*, and one *union*. Two extra find operations may be required in case the SCH created at the start point of a normal circle is merged to another SCH.
– third, at an end point of some normal circles—Fig. 4.6(c). Merging the different faces of two different SCH requires two *find* and one *union*.

Adding up these contributions results in $M \leq N+3(n_0+1)+7n$, which we rewrite as $M \leq f+h+20n$. □

4.4.2 Proof of Section 4.3

Proof.[Thm. 5] We recall that m is the number of input balls, and that n is the number of different intersection circles. First consider the initialization. Initializing LNB_θ requires $O(m)$ time and space, and $O(n)$ lists are created considering arcs intersected at $\theta = 0^+$. Next, consider the sweep. A constant time and space update is performed on the tree every time a face master is created. Moreover, one node is deleted every time a merge of two faces is performed. But since at most $O(n)$ merges are performed (the number of normal end events together with the number of faces intersected at initialization, see Fig. 4.6(a,c)), the total number of nodes created is $O(f + n)$. Finally, upon completion of the sweep, $O(n)$ merges occur to complete the faces started during the initialization. Adding up these costs and space requirements yields the conclusion.

To compute one covering list per face, we first deal with primary/opposite balls. From a depth-first traversal of the covering tree, the lists of primary/opposite balls are constructed for each node. Traversing an edge of the tree consists of copying the list of the father node, from (respectively to) which one ball may be removed (respectively added). Next, the remaining balls are recovered from the lists of friends. □

4.4.3 Handling the Topology at (Bi)polar Events

This section describes operations handled at a pole by providing the pseudo-code for algorithms `topo_handle_polar_event_site` and `topo_handle_bipolar_event_site`. In doing so, we assume functions `upper_halfedge`(A_i) and `lower_halfedge`(A_i) return the named half-edges for a given arc A_i. For a polar or a bipolar circle C, `in`(C) (resp. `out`(C) refers to the current half-edge associated to the inner (outer) half-edge.

We describe how to proceed for one pole. For each pole, we consider a pair of global variables L_h, P_h pointing on half-edges, standing respectively for Last and Previous half-edges. These two pairs are initialized to `NULL`. When the sweep process is over, i.e. $\mathcal{E}$ is empty, and before launching `topo_merge_virtual_faces`, we merge L_h P_h if they are not still `NULL`.

Polar circle. To handle topological operations when a polar circle starts or ends, we have to manage the half-edges passing by its corresponding pole. This can be done using the function `topo_handle_polar_event_site` when starting and ending a polar circle.

Bipolar circle. Let us consider an event site of a bipolar circle C. Topological operations to correctly handle this event can be decomposed in three steps.
▷ **1.** We launch the routine `topo_handle_polar_event_site` for the north pole, in order to anchor at the north pole half-edges of the bipolar circle, and to connect with previously encountered half-edges of other (bi)polar circles.
▷ **2.** We manage intersection of arcs in $\mathcal{V}$ by C
▷ **3.** We relaunch the routine `topo_handle_polar_event_site` but for the south pole.

These operations are summarized in algorithm `topo_handle_bipolar_event_site`. To simplify its presentation, during intersection of arcs of $\mathcal{V}$ by C we consider two functions that create a new half-edge and return a pointer to it:
– `topo_new_left_halfedge`: returns a pointer to the new half-edge which is `out`(C) (respectively `in`(C)) at a start event (respectively end event), with the correct intersection point associated to one extremity of the half-edge.
– `topo_new_right_halfedge`: returns a pointer to the half-edge which is `in`(C) (respectively `out`(C)) at a start event (respectively end event), with the correct intersection point associated to one extremity of the half-edge.

Algorithm 3 `topo_handle_polar_event_site`

if We handle a start point **then**
  if (P_h=NULL) **then**
    L_h=`out`(C)
  else
    merge(P_h,`out`(C))
  end if
  P_h=`in`(C)
else
  if (P_h=NULL) **then**
    L_h=`in`(C)
  else
    merge(P_h,`in`(C))
  end if
  P_h=`out`(C)
end if

Algorithm 4 Algorithm `topo_handle_bipolar_event_site`

L_h=`topo_new_left_halfedge`
R_h=`topo_new_right_halfedge`
`topo_handle_polar_event_site`(NORTH_POLE)
event_site *evt_pol*=$\mathcal{E}$.`pop`{Bipolar event site}
current = pointer on the successor of the north pole in $\mathcal{V}$
stop = pointer on the south pole
while (*current* is not pointing on south pole) **do**
  if (*stop* is not pointing on south pole) **then**
    `handle_event_site`(L_h,R_h)
    current = pointer on the lower bounding arc of the block defined by lists of $\mathcal{E}$.`top`
  end if
  if (($\mathcal{E} \neq \emptyset$) **AND** ($\mathcal{E}$.`top` and *evt_pol* have same θ value) **AND** ($\mathcal{E}$.`top` is not a polar event site)) **then**
    stop =pointer on the top arc of the block defined by lists of $\mathcal{E}$.`top`
  else
    stop = pointer on the south pole
  end if
  while (*current* $\neq$*stop*) **do**
    A=arc pointed by *current*
    current=successor of *current* in $\mathcal{V}$
    merge(`upper_halfedge`(A),L_h)
    L_h=`topo_new_left_halfedge`
    merge(`lower_halfedge`(A),L_h)
    `upper_halfedge`(A)=new halfedge
    merge(R_h,`upper_halfedge`(A))
    `lower_halfedge`(A)=new halfedge
    R_h=`topo_new_right_halfedge`
    merge(R_h,`lower_halfedge`(A))
  end while
end while
while (($\mathcal{E} \neq \emptyset$) **AND** ($\mathcal{E}$.`top` and *evt_pol* have same θ value) **AND** ($\mathcal{E}$.`top` is not a polar event site)) **do**
  `handle_event_site`(L_h,R_h){Only start points in the event site}
end while
`topo_handle_polar_event_site`(SOUTH_POLE)

Chapitre 5

Design of the CGAL Spherical Kernel

Chapter Overview

This chapter presents the design of the CGAL 3D Spherical Kernel dedicated to operations involving circles, circular arcs and spheres in 3D. Section 5.1 provides the mathematics underlying two non trivial predicates. Section 5.2 presents the design of the 3D Spherical Kernel and of the Algebraic Kernel. Section 5.3 bridges the gap between the geometric calculations presented in Section 5.1 and the functionality of the kernel. The link between the computation of an arrangement of circles on a sphere and the 3D Spherical Kernel is presented in Section 5.4. Experiments on random arrangements and molecular models are reported in Section 5.5. For completeness, we include additional details regarding some constructions and predicates in Appendix 5.6.

This chapter corresponds to a paper accepted for publication in the journal *Computational Geometry: Theory and Applications*. This is a joint work with Frédéric Cazals, Pédro M. M. de Castro and Monique Teillaud. Appendix 5.6 was presented in a technical report [36].

5.1 Mathematical Background

This section focuses on basics underlying the 3D Spherical Kernel, as well as on the mathematics needed for two non-trivial operations: The construction of the θ-extremal points of a circle on a given sphere, and the computation of the relative orientation of the tangents of two intersecting circular arcs on a given sphere.

5.1.1 Preliminaries and Notation

The center of a sphere S_i is denoted $c_i = (x_i, y_i, z_i)$, and its radius r_i. We assume the Cartesian coordinates of the center and the squared radius to be rational numbers. All objects (planes, lines, spheres, circles) whose equations have rational coefficients are termed *rational*.

The x-coordinate of point p and vector u are denoted p_x and u_x, and similarly for y- and z-coordinates. The dot and vector products of two vectors u and v are respectively denoted $< u, v >$ and $u \wedge v$. The squared norm of vector u is denoted $u^2 =< u, u >$, and its norm $\|u\|$. The sign of a real number, denoted $\mathrm{Sign}(x)$, is such that $\mathrm{Sign}(x) \in \{-1, 0, 1\}$. Given two points p and q, the vector $q - p$ is denoted pq. If o stands for the origin, and p is a point, the vector op is denoted $\overline{p}$.

The power of point p w.r.t. sphere S_i is defined by $\pi(p, S_i) = pc_i^2 - r_i^2$. The radical plane RP_{ij} of any two spheres S_i and S_j is the plane consisting of the points having equal power with respect to the two spheres. Its equation is $2 < \overline{p}, c_i c_j > + \overline{c_i}^2 - \overline{c_j}^2 + r_j^2 - r_i^2 = 0$. Whenever these two spheres intersect, their intersection circle C_{ij} is also defined as the intersection between either sphere and their radical plane. The center and radius of C_{ij} are denoted c_{ij} and r_{ij}. It can be checked that $c_i c_{ij} = \frac{c_i c_j^2 - r_j^2 + r_i^2}{2 c_i c_j^2} c_i c_j$ and $r_{ij}^2 = r_i^2 - c_i c_{ij}^2$.

A root of a degree n polynomial with rational coefficients is called an algebraic number of degree n. The following observation shows that algebraic numbers of degree two play a central role in our predicates and constructions:

Observation 3 *The Cartesian coordinates of the intersection points of a rational sphere and a rational line are algebraic numbers of degree two in the same extension.*

We also recall the following properties that will be used in our computations:

Observation 4 *Consider two algebraic numbers a and b, of degree at most two in the same algebraic extension. Then $a+b$, $a \times b$ and $1/a$ (if $a \neq 0$) are algebraic numbers of degree at most two, and they also belong to the same algebraic extension.*

5.1.2 Computing the θ-extremal Points of a Normal Circle

We start with a proposition concerned with the z-coordinate of θ-extremal points:

Proposition 1 *Consider a normal circle C_{0i} defined by the intersection of two rational spheres. The z-coordinate of its two θ-extremal points is the following rational number:*

$$z = \frac{2 z_i r_0^2}{\overline{c_i}^2 + r_0^2 - r_i^2}.$$

Proof. Recall that the reference sphere S_0 is centered at the origin, and let C_{0i} be a normal circle intersection of S_0 and S_i. If the z-coordinate c_{0iz} of its center c_{0i} is equal to 0, a symmetry argument with respect to the plane $z=0$ imposes that the z-coordinate of the θ-extremal points is also null. In the following, we therefore assume that $c_{0iz} \neq 0$.

Consider the zero-dimensional intersection set between the circle C_{0i} and the half-plane $P(\theta)$ defining the meridian M_θ (i.e. $S_0 \cap RP_{0i} \cap M_\theta$, where $M_\theta = S_0 \cap P(\theta)$). The θ-extremal points of the circle correspond to the case where this intersection set reduces to one point. Denoting $p = (x, y, z)$ such an intersection point, the corresponding polynomial system reads as:

$$\begin{cases} P(\theta): & -x \sin\theta + y \cos\theta = 0 \\ RP_{0i}: & 2 < \overline{p}, c_0 c_i > + \overline{c_0}^2 - \overline{c_i}^2 + r_i^2 - r_0^2 = 0 \\ S_0: & x^2 + y^2 + z^2 - r_0^2 = 0 \end{cases} \tag{5.1}$$

Assuming $x \neq 0$, or equivalently $\theta \neq \frac{\pi}{2}\,[\pi]$, we investigate this system using $\tan\theta$. (If $x = 0$, we work with $\cot\theta$ by swapping the roles of x and y. Notice that x and y cannot vanish simultaneously for a normal circle, as such a circle does not contain a pole.) The previous system is tantamount to:

$$\begin{cases} y = x \tan\theta \\ z = \frac{\overline{c_i}^2 + r_0^2 - r_i^2 - 2x(x_i + y_i \tan\theta)}{2 z_i} \\ x^2 + x^2 \tan^2\theta + \frac{(\overline{c_i}^2 + r_0^2 - r_i^2 - 2x(x_i + y_i \tan\theta))^2}{4 z_i^2} - r_0^2 = 0 \end{cases} \tag{5.2}$$

Rewrite the last equation of the previous system

$$Ax^2 + Bx + C = 0, \tag{5.3}$$

with

$$\begin{cases} A = (1 + \tan^2\theta) + \frac{(x_i + y_i \tan\theta)^2}{z_i^2} \\ B = -\frac{(x_i + y_i \tan\theta)(\overline{c_i}^2 + r_0^2 - r_i^2)}{z_i^2} \\ C = \frac{(\overline{c_i}^2 + r_0^2 - r_i^2)^2}{4 z_i^2} - r_0^2 \end{cases} \tag{5.4}$$

The values of θ sought are such that Eq. (5.3) has a single solution, namely $x = -B/(2A)$. Imposing that the discriminant of this polynomial vanishes yields the condition $D = 0$, with $D = B^2 - 4AC$, thus $\tan\theta$ is an algebraic number of degree two. Letting T stand for $\tan\theta$ and denoting $\lambda_i = \overline{c_i}^2 + r_0^2 - r_i^2$, we have:

$$D = \overbrace{(-\lambda_i^2 + 4r_0^2 z_i^2 + 4r_0^2 y_i^2)}^{D_2} T^2 + \overbrace{(8x_i y_i r_0^2)}^{D_1} T + \overbrace{(-\lambda_i^2 + 4r_0^2 z_i^2 + 4r_0^2 x_i^2)}^{D_0} \tag{5.5}$$

Let us investigate the cases where polynomial D has degree two and one. As we shall see, in the first case, no θ-extremal point lies in the plane $x = 0$, while the second one corresponds to the situation where one θ-extremal point lies in plane $x = 0$. The latter case is left to the reader and we describe here only the former one.

We assume that $D_2 \neq 0$: Polynomial D is of degree two. We first make the connexion between the discriminant δ of D and the geometry, and proceed with the value of the z-coordinate of θ-extremal points. The discriminant δ of D may be written as:

$$\delta = 4\overbrace{(-2r_0 z_i + \lambda_i)}^{\delta_1}\overbrace{(2r_0 z_i + \lambda_i)}^{\delta_2}\overbrace{(4r_0^2\overline{c_i}^2 - \lambda_i)}^{\delta_3} \tag{5.6}$$

Since the power of a pole of S_0 w.r.t. to sphere S_i reads as $(c_i - (0,0,\pm r_0))^2 - r_i^2 = \overline{c_i}^2 + r_0^2 - r_i^2 \pm 2z_i r_0 = \pm 2r_0 z_i + \lambda_i$, polynomial δ_1 (respectively δ_2) is the power of the north (respectively south) pole w.r.t. S_i. The power of a point w.r.t. to a sphere is negative (respectively positive, zero) iff the point is inside (respectively outside, on) the sphere. Therefore, δ_1 is negative (respectively positive, zero) if the north pole is inside (respectively outside, on) S_i. The same observation holds for δ_2 and the south pole. Finally, for δ_3, observe that:

$$\delta_3 = \overbrace{(-\overline{c_i}^2 + (r_0 + r_i)^2)}^{\delta_{3_1}}\overbrace{(\overline{c_i}^2 - (r_0 - r_i)^2)}^{\delta_{3_2}}. \tag{5.7}$$

The combinations of signs of δ_{3_1} and δ_{3_2} are reported in Table 5.1, and the discussion of the sign of δ is summarized in Table 5.2. These tables prove that as expected, one has $\delta > 0$ for all normal circles, so that Eq. (5.5) can be canceled in two different ways corresponding to the two θ-extremal points.

Finally, substituting any solution of Eq. (5.5) into the expression of z from the system (5.2) proves the claim under the assumption $D_2 \neq 0$. Notice that $\overline{c_i}^2 + r_0^2 - r_i^2 = 0$ corresponds to the case where the circle is bipolar. □

δ_{3_1}	δ_{3_2}	δ_3	**sphere configuration**
+	+	+	S_0 and S_i intersect along a non-degenerate circle
+	−	−	S_i inside S_0 or S_0 inside S_i, no intersection
−	+	−	S_i outside S_0, no intersection
−	−	+	impossible
0	+	0	S_i and S_0 tangent, no inclusion
0	−	0	impossible
+	0	0	S_i and S_0 tangent, S_i inside S_0 or S_0 inside S_i
−	0	0	impossible

TAB. 5.1 – Sign of δ_3 from Eq. (5.7). Abusing terminology, S_i *inside* S_j stands for S_i *inside the ball associated to* S_j, while S_i *outside* S_j stands for the *balls associated to* S_i *and* S_j *are disjoint.*

δ_1	δ_2	δ_3	δ	**circle type**
$+$	$+$	$+$	$+$	normal circle
$+$	$+$	$-$	$-$	S_0 and S_i do not intersect
$+$	$-$	$+$	$-$	threaded circle
$-$	$+$	$+$	$-$	threaded circle
$+$	$-$	$-$	$+$	impossible
$-$	$+$	$-$	$+$	impossible
$-$	$-$	$+$	$+$	normal circle
$-$	$-$	$-$	$-$	S_0 and S_i do not intersect
0	$\neq 0$	≥ 0	0	polar circle if $\delta_3 > 0$, S_0 and S_i are tangent otherwise
$\neq 0$	0	≥ 0	0	polar circle if $\delta_3 > 0$, S_0 and S_i are tangent otherwise
0	$\neq 0$	$-$	0	impossible
$\neq 0$	0	$-$	0	impossible
$\pm$	$\pm$	0	0	tangent spheres if $\delta_1\delta_2 > 0$, impossible otherwise
0	0	$\{\pm, 0\}$	0	bipolar circle if $\delta_3 > 0$, impossible otherwise

TAB. 5.2 – Sign of δ from Eq. (5.6). δ_1 (respectively δ_2) is the power of the north (respectively south) pole w.r.t. S_i, while δ_3 indicates whether S_0 intersects S_i.

Proposition 1 has an interesting consequence:

Corollary 1 *The θ-extremal points of a normal circle C_{0i} are defined by the intersection between the reference sphere, and the intersection line between the radical plane RP_{0i} and the horizontal plane defined by $z = \frac{2z_i r_0^2}{\overline{c_i}^2 + r_0^2 - r_i^2}$. Therefore, by Observation 3, the x- and y-coordinates of these points are algebraic numbers of degree two in the same extension.*

5.1.3 Computing the Relative Orientation of the Tangents of two Circles

Consider two circles, each supported by a rational plane, intersecting transversally at point p on S_0. Under the assumptions that none of the circles is bipolar and that p is neither a pole nor a θ-extremal point of a normal circle, we wish to compute the relative orientation of the tangents of these two circles at p.

Under the previous assumptions, denote t_k, $k = i$ or j, the tangent vector associated to circle C_{0k} at point p, as illustrated on Fig. 5.1. Tangents vectors t_i and t_j are chosen so as to point towards *increasing* values of θ, locally in the tangent space of S_0 at p. Because t_i and t_j are orthogonal to $\overline{p}$, the relative orientation of the tangents is given by Sign(Δ), with

$$\Delta =< t_i \wedge t_j, \overline{p} > . \tag{5.8}$$

The details are as follows. The tangent to a circle is supported by the line intersection of the plane of the circle together with the tangent plane of S_0 at p. More precisely, consider the vectors $n_i = c_{0i}p$ and $n_j = c_{0j}p$. For $k = i, j$ the tangent vectors are obtained as follows: For a normal or a polar circle, $t_k = \beta_k \overline{c_{0k}} \wedge n_k$, with $\beta_k = -1$ (respectively 1) if the circular arc containing p is an upper (respectively a lower) one; For a threaded circle, let m_k be $\overline{c_{0k}}$ if the circle is not a great circle and otherwise any normal vector of the plane of the circle, then $t_k = \gamma_k(m_k \wedge n_k)$, with $\gamma_k = \pm 1$ such that $\gamma_k m_{kz} > 0$.

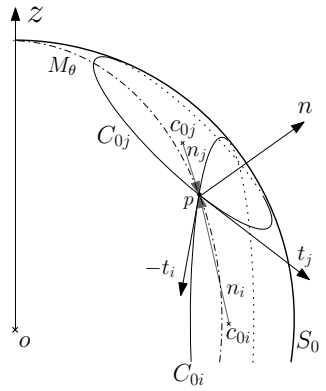


FIG. 5.1 – Tangent vectors t_i and t_j to circles C_{0i} and C_{0j} at one of their intersection points p. Vector n is the normal to S_0 at p.

The Cartesian coordinates of p are three algebraic numbers of degree two in the same extension. Computing tangent vectors using the formulation $t_k = \pm(c_{0ky}p_z - c_{0kz}p_y, c_{0kz}p_x - c_{0kx}p_z, c_{0kx}p_y - c_{0ky}p_x)^t$, we now give the cost of computing Δ in term of products $\Pi_{T;T'}$ and sums $\Sigma_{T;T'}$, where T and T' are either algebraic numbers of degree two (denoted AN_{op}) or rational numbers (denoted R). AN_{op} is a trivariate rational polynomial expression in the coordinates of p. This ensures that it is an algebraic number of degree two lying in the same extension as p. If considering the tree of arithmetic operations representing such an expression, the subscript op is equal to the maximum number of edges between the final expression and a coordinate of p. For instance each coordinate of t_k is a AN_2 obtained through two $\Pi_{R;AN_0}$ followed by one $\Sigma_{AN_1;AN_1}$. A naive computation of Δ requires two $\Pi_{AN_2;AN_2}$ and one $\Sigma_{AN_3;AN_3}$ for each coordinate of the vector product; three $\Pi_{AN_4;AN_0}$, one $\Sigma_{AN_5;AN_5}$ and one $\Sigma_{AN_6;AN_5}$ for the dot product. In Observation 6, omitting products of rationals, we show that if no circle is a great circle, we can actually compute the sign of Δ with only two $\Pi_{AN_2;AN_2}$, two $\Pi_{R;AN_3}$ and one $\Sigma_{AN_4;AN_4}$. We use the following elementary observation:

Observation 5 *Let $\Delta' = \frac{t_{j_z}}{\|t_j\|} - \frac{t_{i_z}}{\|t_i\|}$. The sign of Δ matches that of Δ'.*

Notice that the sign of Δ' is trivially inferred in the following three cases: $t_{i_z} = 0$ and $t_{j_z} = 0$; $t_{i_z} = 0$ or $t_{j_z} = 0$; t_{i_z} and t_{j_z} have different signs.

Otherwise we have:

Observation 6 *If none of the circles involved is a great circle and the signs of t_{i_z} and t_{j_z} are identical, the sign of Δ' is given by:*

$$Sign(\Delta') = Sign(t_{j_z})Sign\left(\overline{c_{0i}}^2 r_{0i}^2 (t_{j_z})^2 - \overline{c_{0j}}^2 r_{0j}^2 (t_{i_z})^2\right)$$

Proof. In the following, we suppose without loss of generality that the signs of t_{i_z} and t_{j_z} are identical and non-null. The sign of Δ' is the same as that of $\|t_i\|t_{j_z} - \|t_j\|t_{i_z}$. and we have:

$$t_i^2 t_{j_z}^2 - t_j^2 t_{i_z}^2 = (\|t_i\|t_{j_z} + \|t_j\|t_{iz})(\|t_i\|t_{j_z} - \|t_j\|t_{iz}) \tag{5.9}$$

And since we supposed that $\text{Sign}(t_{j_z}) = \text{Sign}(t_{i_z})$, we also have

$$\text{Sign}(\Delta') = \text{Sign}(t_{j_z})\text{Sign}(t_i^2 (t_{j_z})^2 - t_j^2 (t_{i_z})^2). \tag{5.10}$$

Recall that the Cartesian coordinates of the center of circle $C_{0k}, k = i, j$, are rational numbers. As $\overline{c_{0k}}$ and $c_{0k}p$ are orthogonal, the sine of the angle between the vectors is 1. Since $t_k = \pm\overline{c_{0k}} \wedge c_{0k}p$, we have $t_k^2 = \overline{c_{0k}}^2 c_{0k}p^2$, which we can rewrite $t_k^2 = \overline{c_{0k}}^2 r_{0k}^2$.

□

5.2 Software Design

In this section, we sketch the general design of CGAL kernels, and focus on the 3D Spherical Kernel.

5.2.1 CGAL Kernels: Rationale

We recall that in CGAL, geometric algorithms are generic: a geometric class has a template parameter called a *traits* class, providing the minimum set of functionality required. A traits class is documented as a *concept*, that is to say as a set of functionality with prescribed signatures.

The general design of a kernel concept for curved objects initially proposed in [84] is driven by the following goals: (i)interoperability of any model of the kernel concept with CGAL geometric algorithms; (ii)re-usability of the existing CGAL kernel models for linear objects; (iii)genericity and flexibility: ability to use other linear kernels than those in CGAL and independence from a particular implementation of the algebraic operations needed. The declaration of a kernel for curved objects is the following:

```
template < typename LinearKernel, typename AlgebraicKernel >
class Curved_kernel;
```

Practically, to indicate whether an object is a member of the 3D Spherical Kernel or of the Algebraic Kernel, we use the prefixes `SK::` and `AK::` respectively.

Let us now comment briefly on the two template arguments. Using the extensibility and adaptability scheme of the CGAL kernel [110], the `Curved_kernel` inherits from the model of `LinearKernel` given as template parameter, which allows it to directly benefit from all the functionality on linear objects. The `LinearKernel` concept will not be presented here. It coincides with the basic CGAL kernel concept extensively documented in the CGAL manual.

The requirements on the `AlgebraicKernel` concept listed in Section 5.2.3 are guided by the functionality offered by the 3D Spherical Kernel (Section 5.2.2). Notice that this Algebraic Kernel concept is actually called `AlgebraicKernelForSpheres` in the CGAL package because it is restricted to functionality required by the 3D Spherical Kernel. Nevertheless, we use the shorter name `AlgebraicKernel` in this chapter.

At the embedding level, the number type used to represent the objects of the linear kernel (coordinates for points, coefficients of equations for planes, spheres) is supposed to lie in a *field number type* (a type providing elementary operations $+, -, \times, /$), that will typically be the rationals. From now on and for the sake of simplicity, we shall refer to this basic number type as *rational*.

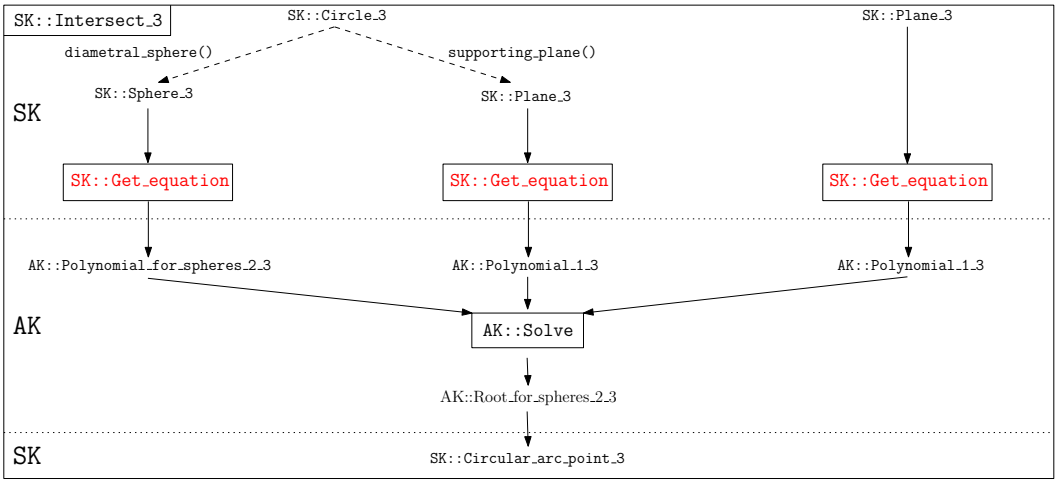
Regarding algebra, the key idea in this design is that each primitive object of the 3D Spherical Kernel is internally represented by a corresponding object of the Algebraic Kernel–its equation or coordinates. Moreover, each functionality of the 3D Spherical Kernel is obtained by a combination of calls of functionality of the Algebraic Kernel on the algebraic representations of its arguments. Fig. 5.2 illustrates how algebra and geometry communicate, on the example of the computation of the intersection of a circle and a plane: We first retrieve the underlying primitive geometric objects, get their equations as polynomials of the Algebraic Kernel, and solve the system of corresponding equations using the solver provided by the Algebraic Kernel. Finally, the root of the system is used to retrieve the geometric intersection.

5.2.2 3D Spherical Kernel

To list the main geometric functionality offered upon instantiation of a model of the 3D Spherical Kernel concept, we discuss in turn the types provided, the predicates and constructions, and the connexion between geometry and algebra.

Types. Among the object types provided by the 3D Spherical Kernel, one finds the twelve listed in Table 5.3. These object types can be classified into three groups of four types: objects inherited from the linear kernel, general objects and objects lying on a given reference sphere—as evidenced by the suffix `ORS` which stands for `OnReferenceSphere`. All but one of these objects types are provided accessor(s) by the concept: Two objects of type `ThetaRep` (designed to represent the θ-coordinate of a point) must only be comparable through function object `CompareTheta_3`.

Note that a circular arc can be defined by a supporting circle and two endpoints. It is unambiguously defined as the set of points lying on the circle, when walking counterclockwise from its source to its target



Names in boxes represent 3D Spherical Kernel or Algebraic Kernel functors, while names on dashed arrows are access functions.

FIG. 5.2 – An example of an implementation of a geometric construction: computing the intersection of a circle and a plane.

Type	Basic accessors
`SK::Point_3`	`x()`, `y()`, `z()`
`SK::Line_3`	`point(int)`, `to_vector()`
`SK::Plane_3`	`a()`, `b()`, `c()`, `d()`
`SK::Sphere_3`	`center()`, `squared_radius()`
`SK::Circle_3`	`center()`, `squared_radius()`, `supporting_plane()`, `diametral_sphere()`
`SK::CircularArcPoint_3`	`x()`, `y()`, `z()`
`SK::CircularArc_3`	`source()`, `target()`, `supporting_circle()`
`SK::LineArc_3`	`source()`, `target()`, `supporting_line()`
`SK::CircleORS_3`	`type_of_circle_on_reference_sphere()`, `reference_sphere()`
`SK::CircularArcPointORS_3`	`reference_sphere()`
`SK::CircularArcORS_3`	`reference_sphere()`, `theta_rep()`
`SK::ThetaRep`	

TAB. 5.3 – Types of the 3D Spherical Kernel. The three groups of the table read as follows: inherited from the linear kernel; general objects; objects on a reference sphere. Notice that objects from the third group inherit the functionality of objects of the second group.

Predicate	Returns	Arg.	Meaning
`SK::CompareX_3`	{<,=,>}	(p_i, p_j)	p_i.x() vs. p_j.x()
`SK::CompareY_3`	{<,=,>}	(p_i, p_j)	p_i.y() vs. p_j.y()
`SK::CompareZ_3`	{<,=,>}	(p_i, p_j)	p_i.z() vs. p_j.z()
`SK::CompareXY_3`	{<,=,>}	(p_i, p_j)	Lexicographic order
`SK::CompareXYZ_3`	{<,=,>}	(p_i, p_j)	Lexicographic order
`SK::Equal_3`	`bool`	(w_i, w_j)	Geometric equality
`SK::HasOn_3`	`bool`	(w_i, w_j)	$w_j \subseteq w_i$
`SK::HasOnBoundedSide`	`bool`	(p, S)	$p \in$ ball bounded by S
`SK::CompareZAtXY_3`	{<,=,>}	(p, P)	p vs. P $(a, b, d \neq 0)$
`SK::CompareYAtXZ_3`	{<,=,>}	(p, P)	p vs. P $(a, c, d \neq 0)$
`SK::CompareXAtYZ_3`	{<,=,>}	(p, P)	p vs. P $(b, c, d \neq 0)$
`SK::DoOverlap_3`	`bool`	(A_i, A_j)	Overlap test
`SK::DoIntersect_3`	`bool`	(w_i, w_j)	Intersection test
`SK::CompareTheta_3`	{<,=,>}	(p_i^r, p_j^r)	Compare θ-coordinates
`SK::CompareTheta_3`	{<,=,>}	(θ_i, θ_j)	Compare θ-values
`SK::CompareThetaZ_3`	{<,=,>}	(p_i^r, p_j^r)	Lexicographic order
`SK::CompareZAtTheta_3`	{<,=,>}	$(A_i^m, A_j^m, \mathbb{M})$	Order along $\mathbb{M}$
`SK::CompareZAtTheta_3`	{<,=,>}	(p^r, A_i^m)	p^r vs. A_i^m along $M_\theta \ni p^r$
`SK::CompareZToRight_3`	{<,=,>}	(A_i^m, A_j^m)	Right of common pt order

Notation: p, A, P, $\mathbb{M}$, S, θ, w, stand respectively for a point, a circular arc, a plane, a meridian included in a rational plane, a sphere, a θ-value of type `ThetaRep`, any object of the 3D Spherical Kernel. Two instances of any type are distinguished using a subscript i and j. A superscript m on a circular arc indicates that this arc is θ-monotone—obviously on a reference sphere. A superscript r on a point indicates that such a point is on a reference sphere. For equality test, w_i and w_j are of the same type.

TAB. 5.4 – Predicates of the 3D Spherical Kernel.

in the positive plane containing the circle—a plane is positive if its equation is of the form $ax+by+cz+d = 0$ with $(a, b, c) > (0, 0, 0)$ according to the lexicographic order.

Predicates and Constructions. Predicates and constructions are provided by the kernel as C++ functions objects (called functors). Note that a functor can be used as a template argument.

A model of the 3D Spherical Kernel must provide the functionality described in the sequel. We list only the most important requirements in Table 5.4 (predicates) and Table 5.5 (constructions). A construction returns an iterator over a (possibly empty) set of solutions associated to the problem solved by the function.

Communication with Algebra. Functor `SK::GetEquation` returns a polynomial of the Algebraic Kernel providing an equation of a plane or a sphere. As explained at the beginning of Section 5.2, this is the essential link between the 3D Spherical Kernel and the Algebraic Kernel.

Construction	Arguments	Returned objects
`SK::Intersect_3`	Two or three primitives	Their intersection(s)
`SK::ThetaExtremePoint_3`	A circle or a circular arc, on a reference sphere	Its θ-extremal point(s)
`SK::MakeThetaMonotone_3`	A circle or a circular arc, on a reference sphere	Its θ-monotone sub-arc(s)

TAB. 5.5 – Constructions of the 3D Spherical Kernel.

Type	Represents
`AK::PolynomialForSpheres_2_3`	Equation of a sphere; degree 2, 3 variables
`AK::Polynomial_1_3`	Equation of a plane; degree 1, 3 variables
`AK::RootForSpheres_2_3`	Root of a system of 3 polynomials of the previous types

The `AK::RootForSpheres_2_3` *type is used to represent the coordinates of a point of the 3D Spherical Kernel.*

TAB. 5.6 – Types of the `AlgebraicKernel`.

5.2.3 Algebraic Kernel for Spheres

The operations provided by the CGAL 3D Spherical Kernel make heavy use of algebraic operations. The `AlgebraicKernel` parameter has a crucial role in particular for the robustness of the 3D Spherical Kernel.

Types. As already mentioned in [84], the Algebraic Kernel must provide basic types (polynomials and roots of systems, see Table 5.6), and basic functionality on them. In the sequel, we focus on the most important requirements of the `AlgebraicKernel` concept only.

Main Algebraic Predicates and Constructions. Several elementary functors reflecting the 3D Spherical Kernel user interface are provided for predicates, for instance to compare the coordinates of two `AK::RootForSpheres_2_3`. Moreover, the functor `AK::SignAt` allows one to evaluate the sign of a polynomial of type `AK::PolynomialForSpheres_2_3` or `AK::Polynomial_1_3` at a `AK::RootForSpheres_2_3`.

The main construction is provided by the functor `AK::Solve`, which solves a zero-dimensional polynomial system featuring polynomials whose types are listed in Table 5.6. The return value is an iterator over the solution set (possibly empty), each solution being given as a `AK::RootForSpheres_2_3`.

5.3 Implementation Details

In this section, we bridge the gap between the geometric calculations presented in Section 5.1 and the functionality of the kernel: We describe the way concepts developed in the previous section are implemented.

Operations Involving Points. In the current CGAL implementation, an algebraic number A of degree two is represented as a triple of rationals (a, b, c) such that $A = a + b\sqrt{c}$. By Corollary 1 and Observation 3, the Cartesian coordinates of an intersection point and of a θ-extremal point of a normal circle are algebraic numbers of degree two. Comparison of Cartesian coordinates of such points is easily handled using the CGAL representation. A similar observation holds for the evaluation of a degree two polynomial at such a point—a calculation involved in predicates `AK::SignAt`, `SK::CompareZAtXY_3`, `SK::CompareYAtXZ_3`, `SK::CompareXAtYZ_3`.

Predicates `SK::CompareTheta_3` and `SK::CompareThetaZ_3` compare two θ-coordinates using a partition of the interval $(0, 2\pi]$ into eight open intervals [53] of length $\frac{\pi}{4}$ and eight value $\frac{k\pi}{4}$ with $k = 0 \ldots 7$. The comparison of θ-coordinates of points falling in different intervals is trivial, while that of points falling in the same interval requires comparing $\tan\theta$ or $\cot\theta$. This boils down to comparing two algebraic number of degree two—each constructed as a quotient (Observation 4) of two degree two algebraic numbers in the same extension using the x- and y-coordinates of each point. This strategy is valid even for events at a pole, using $(y_i, -x_i, 0)$ and $(-y_i, x_i, 0)$ as fictitious points whose θ-coordinates match the θ-extremal values of a polar or bipolar circle C_{0i}. Indeed, one can see that a plane containing the poles and these two points is tangent to or contains C_{0i}.

[53] This predicate is presented more in details in Appendix 5.6.2.

Identifying Intersection Points. The intersection points of two circles and the θ-extremal values of a circle come into pairs, so that finding the element of the pair with smallest θ-coordinate is an important primitive. Using the afore-mentioned partition of $(0, 2\pi]$, this is trivial if the two points fall within different intervals. If not, let p and q be two such points. In such a setting, identifying the smallest θ-coordinate, is equivalent to computing $\mathrm{Sign}(p_y q_x - p_x q_y)$. By Observation 3 and Corollary 1, there exist four rational numbers a, b, c, d and a rational polynomial of degree two P such that:

$$\begin{cases} p_x = aR_1 + b & p_y = cR_1 + d, \text{ with } P(R_1) = 0 \\ q_x = aR_2 + b & q_y = cR_2 + d, \text{ with } P(R_2) = 0 \end{cases} \tag{5.11}$$

Therefore, $\mathrm{Sign}(p_y q_x - p_x q_y) = \mathrm{Sign}((bc-ad)(R_1-R_2))$. Since R_1 and R_2 are roots of the same polynomial, upon creation of p and q we only have to evaluate $\mathrm{Sign}(bc - ad)$.

Sorting Circular Arcs at a Common Point. Let p be an intersection point of two θ-monotone circular arcs and suppose that p is on a meridian M_θ. The predicate `SK::CompareZToRight_3` finds the relative position (above, on or below) of the two arcs to the right of p, i.e. for the angle value $\theta + \varepsilon$ with ε arbitrarily small. The predicate is not defined if point p is a pole, and the associated circles are of any type except bipolar as no θ-monotone circular arc is defined on a bipolar circles. The two supporting circles are denoted C_{0i} and C_{0j}. We now consider three exclusive cases:
– If p matches a θ-extremal point of at least one of the two circles, then this circle is normal. The ordering is trivial resorting to the radii of circles and upper/lower status of θ-monotone circular arcs.
– If the circular arcs intersect transversally, we use the sign of Δ in Eq. (5.8).
– If the circular arcs are tangent, for $k = i$ or j, we define z_k to be either the z-coordinate of the θ-extremal points of C_{0k} if it is a normal circle, or the z-coordinate of the pole C_{0k} goes through if it is a polar circle. We have two sub-cases:
(i)If one of the two circles is threaded, say C_{0i}, we conclude from the sign of $c_{0iz} - c_{0jz}$ if C_{0j} is threaded too, and from the sign of $p_z - z_j$ otherwise.
(ii)If both circles are not threaded, we conclude from the radii of the circles and the sign of $p_z - z_k$, $k = i, j$.
Note that if $p_z - z_k$ is negative (respectively positive), the circular arc of C_{0k} involved is a lower (respectively upper) one.

Ordering θ-monotone Circular Arcs. We consider two problems: ordering two circular arcs intersecting M_θ included in a rational plane, and positioning a point p along a meridian with respect to a circular arc intersected by any given meridian. These two operations are provided by the functor `SK::CompareZAtTheta_3`.

For the first predicate, meridian M_θ is included in a rational plane. We explicitly construct and compare the z-coordinates of the intersection points between the meridian and the circular arcs.
For the second predicate, observe that a non-great circle decomposes S_0 into two regions of unequal areas—the region of largest area and the center of S_0 are on the same side of the plane containing the circle. To begin with, we look for the position of the point p w.r.t. the supporting plane of the circle of the circular arc. This information is sufficient to conclude if p lies on the plane or if the corresponding circle is threaded or polar. When the circular arc lies on a normal circle: If p lies inside the cap of least area defined by the circle on S_0 then p is below the upper circular arc and above the lower one; otherwise the position of p relatively to the circular arc is given by the sign of the difference of p_z and the z-coordinate of the θ-extremal points of the circle.

Misc. For testing equality of two geometric objects, we compare either their parameters or their algebraic representations. Overlapping tests are performed using underlying geometric objects and inclusion of endpoints. Intersection tests and computations are based on classical mathematical inequalities. Predicate `SK::HasOnBoundedSide` obviously relies on evaluation of the sign of sphere equation at the point considered. The inclusion tests performed by predicate `SK::HasOn_3` are easily triggered using the underlying geometric or algebraic representation. For a point on a circular arc, we additionally need some orientation tests.

5.4 Application: Computing Spherical Arrangements

In this section, we present the connexion between the 3D Spherical Kernel and a Bentley-Ottmann like algorithm [23] computing the exact arrangement of circles on a sphere (chapter 3). In a nutshell, the algorithm takes as input a collection of circles, and returns a decomposition of the sphere into regions whose interiors are connected—the decomposition being stored in an extended half-edge data structure handling holes in faces. In the following, we explain how primitives from the 3D Spherical Kernel are used, and refer the reader to chapter 3 for the details on the algorithm.

Algorithm Description. As recalled in chapter 3.1, the BO sweep-line algorithm requires (i)the intersection of the sweep-line with an object to be at most one point, (ii) a predicate to give the position of two objects along the sweep-line, and, (iii)upon adjacency of two objects along the sweep-line, a predicate to test if the objects intersect and a construction to build their possible intersection(s). The sweep-line stops at events to update the order of swept objects along the sweep-line. In the spherical case, the algorithm sweeps a reference sphere using a meridian M_θ anchored at the poles, moving it from 0 to 2π using cylindrical coordinates. Given a collection of circles on a reference sphere, the algorithm decomposes them into θ-monotone circular arcs, using θ-extremal points. An event corresponds to an intersection point (either a transverse intersection or a tangency point) between two circular arcs, or to a θ-extremal point of a circle. Degeneracies occur when several events are associated to the same point of the reference sphere. To handle them, events are gathered into a data structure called the *event site*. The *vertical ordering* $\mathcal{V}$ stores the order of θ-monotone circular arcs along the meridian during the sweep. The *event queue* $\mathcal{E}$ stores event sites, which are created upon insertion of intersection and θ-extremal events.

Connexion between the BO Algorithm and the 3D Spherical Kernel. The algorithm features three main constructions. The first two construct θ-extremal points and the create θ-monotone circular arcs from input circles using `SK::ThetaExtremePoint_3` and `SK::MakeThetaMonotone_3` respectively. The third one constructs intersection points using `SK::Intersect_3`.

Predicates manipulates $\mathcal{V}$ and $\mathcal{E}$. Let us first examine the vertical ordering $\mathcal{V}$:
(i)The initialization of $\mathcal{V}$ compares the position of two circular arcs along meridian M_θ with $\theta = 0$, using predicate `SK::CompareZAtTheta_3`. For intersections occurring on M_0, one further compares the circular arcs to the right of this point using predicate `SK::CompareZToRight_3`.
(ii)To update $\mathcal{V}$, the only predicate involved is that required to insert the circular arcs of a normal circle starting to be swept at a given θ-extremal point p. To do so, we locate p along the meridian amongst circular arcs present in $\mathcal{V}$, the predicate involved being `SK::CompareZAtTheta_3` with one point and one circular arc. The case of several circles having p as θ-extremal point is easily handled once the positions of the upper and lower arcs of the circle of greatest radius have been determined—this a pencil of circles tangent at p.
(iii) To maintain a linear size event queue [34], within an event site, only intersection events corresponding to a pair of circular arcs adjacent along $\mathcal{V}$ are stored. This also prevents re-inserting a given event—which is harmful since such a re-insertion causes arithmetic filter failures while seeking this event in the queue. Finally, appropriately concatenating blocks of arcs involved within an event site allows one to maintain $\mathcal{V}$ without any numerical operation.

Let us now analyze the event queue $\mathcal{E}$. Its initialization requires all θ-extremal points. Given all but threaded circles using access function of circles `type_of_circle_on_reference_sphere_3()`, such points are constructed using functor `SK::ThetaExtremePoint_3`. The detection of new intersection points from new adjacencies along $\mathcal{V}$ uses predicate `SK::DoIntersect_3`. All intersection and θ-extremal points are sorted using `SK::CompareTheta_3` and `SK::CompareZ_3`.

The algorithm is summarized in Table 5.7, with primitives from the 3D Spherical Kernel in typewriter font. From this algorithm, the construction of the half-edge data structure storing the arrangement uses two Union-Find processes, as explained in chapter 4.

0. Classify circles as normal/polar/bipolar/threaded.
♦`SK::CircleORS_3::type_of_circle_on_reference_sphere_3()`
Compute θ-extremal points and decompose circles into θ-monotone arcs.
♦`SK::ThetaExtremePoint_3`, `SK::MakeThetaMonotone_3`
1. Initialize $\mathcal{V}$: Fill $\mathcal{V}$ with circular arcs intersected by the meridian at $\theta = 0$.
♦`SK::CompareZAtTheta_3`, `SK::CompareZToRight_3`
2. Initialize $\mathcal{E}$:
(a) Look for intersections between circular arcs adjacent in $\mathcal{V}$ at $\theta = 0$,
♦`SK::DoIntersect_3`
and insert the corresponding intersection points into $\mathcal{E}$.
♦`SK::Intersect_3`, `SK::CompareTheta_3`, `SK::CompareZ_3`
(b) For all but threaded circles, insert θ-extremal points into $\mathcal{E}$.
♦`SK::CompareTheta_3`, `SK::CompareZ_3`
3. While $\mathcal{E}$ is not empty do
(a) Remove from $\mathcal{V}$ the circular arcs of the event(s) ending.
(b) Insert into $\mathcal{V}$ the circular arcs of the event(s) starting.
♦`SK::CompareZAtTheta_3`
(c) Swap in $\mathcal{V}$ the circular arcs intersecting transversally at the event.
(d) Insert into $\mathcal{E}$ the intersection detected from the new adjacencies along $\mathcal{V}$.
♦`SK::DoIntersect_3`, `SK::Intersect_3`, `SK::CompareThetaZ_3`

TAB. 5.7 – The Bentley-Ottmann algorithm for circles on a sphere, together with the primitives of the 3D Spherical Kernel involved.

5.5 Experiments on Spherical Arrangements

In this section we present two types of results for the algorithm of section 5.4: On the one hand, we investigate the practical complexity of the algorithm, and on the other hand, we compare to previous work using molecular models.

Variants. The arrangement algorithm was implemented as a generic C++ class, allowing us to investigate the following three variants:
– The *double variant* instantiates the algorithm using a plain `double` number type as rational type. The main interest is to estimate the overhead imposed by the certification of the results. No guarantee is provided either on the termination or on the correctness of the arrangement.
– The *exact variant* instantiates the algorithm using `CGAL::Lazy_Exact_NT<Gmpq>` as rational type. This is a filtered version of `Gmpq` [102] using the double interval type `CGAL::Interval_nt`. It uses the `Gmpq` exact rational number type when intervals are not sufficient to certify the answer to a predicate. This number type stores for each number an approximated value and the exact value encoded as a DAG of the arithmetic operations needed to construct it—the value is computed upon request.
– The *exact filtered variant* instantiates the algorithm twice, with `CGAL::Interval_nt` and `CGAL::Lazy_Exact_NT<Gmpq>`. as follows. We first launch the instantiation based upon `CGAL::Interval_nt`. If the certification of a predicate fails, we restart the whole arrangement calculation using the instantiation based upon `CGAL::Lazy_Exact_NT<Gmpq>`. While the standard filtering strategy consists of re-computing a quantity at the predicate level; we do the same but for a whole arrangement. This strategy makes sense if the first calculation fails with low probability.
Running Times of an Arrangement of Circles. As shown in section 3.5, the Bentley-Ottmann algorithm on a sphere has complexity $c(n + k + l) \log n$, with n the number of circles, k the number of intersection points, and l the number of faces bounded by exactly two arcs ($l = O(k)$ in non-degenerate

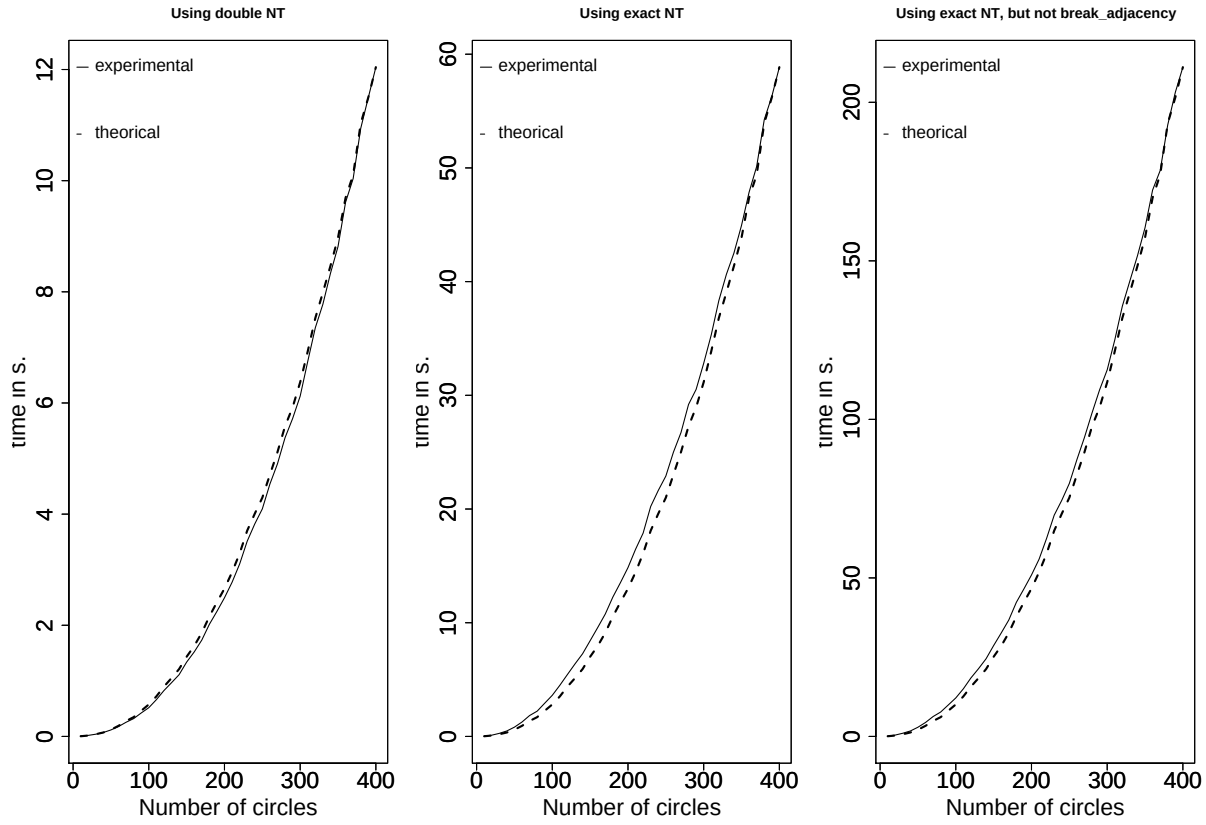


FIG. 5.3 – Assessement of the variants: observed vs theoretical complexities for random arrangements.

cases). To scale the running time with the theoretical complexity, we wish to measure to constant c of the algorithm. To investigate this question, tests were run on collections of random circles on the sphere S_0 centered at the origin, each circle being defined by its center picked uniformly at random within the ball bounded by S_0. Notice this strategy does not yield great circles. The number of circles chosen varies from 10 to 400 by step size of 10. Computations were run on a bi-proc Pentium IV(R) 3.06Ghz with 2.5GB of RAM.

We fit a curve on running times. As illustrated on Fig. 5.3, the theoretical curve is in excellent agreement with the experimental curves, for the double and exact variants. Setting the constant c to match the experimental time obtained for $n = 400$ and the measured k and l, we observe that the constant factor is 2.1×10^{-5} for the double variant and 10.2×10^{-5} for the exact variant.

Running Times on Molecular Models. Given a molecular model, we wish to compute one surface arrangement for each atom. This arrangement features the circles defined by the intersections between this atom and its neighbors. These neighbors are retrieved using a regular grid that partitions the axis aligned bounding box of the model into *cells*. The length of each edge in the grid is the diameter of the largest sphere of the model. Each atomic sphere is associated with a cell in the grid and all the spheres intersecting that sphere are either in the same cell or in adjacent ones.

The only reported running times we are aware of to compute an arrangement of circles on a sphere are those obtained with the algorithm based on explicit controlled perturbations of spheres [86]. Because the perturbation used is global, the running times reported correspond to the cumulative cost of the arrangements on all spheres. Four protein models from the Protein Data Bank [194] were used—PDB codes: `1bzm`, `1jky`, `7at1`, `1l7x`. Table 5.8 lists our results on these models, using a Pentium III(R) 1Ghz

Input file	#atoms	From [86]	Double	Exact	Exact filtered
1bzm	2049	8.31	4.98	45.51	9.47
1jky	5734	26.94	15.75	149.44	29.73
7at1	7169	28.40	17.00	162.70	32.89
1l7x	12912	54.50	33.00	311.17	64.32

TAB. 5.8 – Tests on 4 macro-molecular structures. Total time (in seconds) for computing the Van der Waals surface, including the perturbation, taken from [86] vs. total time of computing the arrangement of circles on each atomic sphere for the three variants.

NT	**neighbor**	**argt**	**area**	**total**
double	0.11s	2.14s	0.71s	2.96s
exact	0.38s	21.47s	7.57s	29.42s
exact filtered	0.23s	4.01s	1.67s	5.91s

TAB. 5.9 – Comparing Number Types (NT) for complex `1acb` (2433 atoms): Run-times to report the neighbors, compute the arrangements, and compute the surface areas of the faces on each atomic sphere.

with 1GB of RAM in our case, as opposed to a bi-proc Pentium III(R) 1GHz with 2GB of RAM for [86]. On these examples, our code is about 65% faster using the double variant and 20% slower using the exact filtered variant.

Consider Tables 5.8 and 5.9. The ratio between the double variant and the exact filtered variant is 2. But calculations in double fail on (nearly-)degenerate inputs. One such highly degenerate example is displayed in Fig. 5.4. Failures also happen for molecular models, since 6 atoms had to be removed to get the *double* row in Table 5.9.

Maintenance of a linear event queue. In chapter 3, we described the function `break_adjacency`, which keeps the size of the event queue linear. To investigate the impact of this function on the running time, we modified our algorithm to no longer remove non-valid intersection events. Using the aforementioned framework for collections of random circles, and using the exact variant, the running time when using function `break_adjacency` is better (see Fig. 5.3). This actually corresponds to a much smaller priority queue (Fig. 5.5(Left)), most of the events found in the non-cleaned up queue being actually non-valid (Fig. 5.5(Right)).

The reason for this difference is due to repeated insertion tries. Numerically, comparing the cylindrical coordinates of event points relies on the comparison of algebraic numbers of degree two. While using the exact variant, the underlying number type always resorts to exact computations when trying to insert an event already present. As an illustration, we considered a random collection of 200 circles on a sphere, forming an arrangement with 14,946 intersection points. During the run of the algorithm, 17,153 intersection events became non-valid (we have no information on the distribution of these 17,153 events amongst the 14,946 intersection points.) Using the `CGAL_PROFILE` flag coming from the `CGAL::Lazy_Exact_NT` class, we observed that when non-valid intersection events are removed of $\mathcal{V}$, 343,247 calls to predicates comparing two algebraic numbers are done, only 4 need exact values. If non-valid intersection points are not removed, the number of calls increases to 432,572, and the number of filter failures to 34,310. Observe that the increase of failures is exactly twice the number of non-valid events, as any non-valid event yields filter failures for the comparison of the θ and z coordinates [54].

For the sake of completeness, we equipped our algorithm with a map (denoted *intersection map*), to test a naive approach to avoid unnecessary insertion tries. When using the function `break_adjacency`, this structure maps a pair of non-valid arcs to an intersection point (to avoid computing twice coordinates

[54] To be precise, we get one failure for the comparison of $\tan\theta$ or $\cot\theta$, and one for that of z. See the section 5.3 for the strategy used to compare the cylindrical coordinates of two event points.

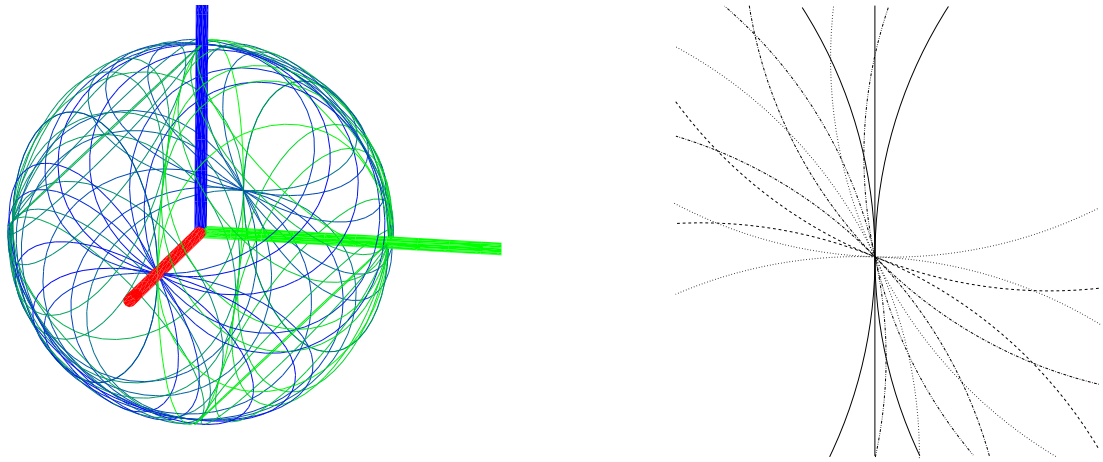


FIG. 5.4 – Left: degenerate arrangement of 47 circles on S_0. Right: zoom about the point $(r_0, 0, 0)$: six different directions of tang ency for thirteen circles. This point is a θ-extremal point for two circles. Ten circles (two among them being polar) intersect at that point and a bipolar circle passes through it. This arrangement features 674 vertices, 1384 edges, 712 faces and no holes. The Euler characteristic is $674 - 1384 + 712 = 2$.

of such a point); otherwise it maps a pair of arcs to the boolean value `TRUE` (to avoid insertion try of events already in $\mathcal{E}$). As can be seen on Fig. 5.6 and Fig. 5.3, the performances of these strategies are quite similar to those of the exact version using function `break_adjacency`. Looking at the maximum amount of memory used without function `break_adjacency`, we observe that on average 35% more memory than the original version (no intersection map and using function `break_adjacency`) is needed when using the intersection map, against 17% when not using it.

5.6 Appendix

For the sake of completeness, we detail here the (trivial) mathematics, used by the constructions and predicates introduced by the 3D Spherical Kernel.

5.6.1 Constructions

Computing intersections

The developments presented in this section are naturally involved in the implementation of `SK::Intersect_3`, but also in those of a number of constructors (that of a circle from a sphere and a plane, for example), and access functions.

Given the primitives introduced in section 5.2.2, we shall consider the following six three dimensional objects: plane, sphere, circle, circular arc, segment and line. We examine the pairwise intersection of any two such primitives, as the intersection of an arbitrary number of them can always be computed from pairwise intersections. When dealing with two objects, we shall assume without loss of genericity [55] that (i) they are different, (ii) they do not overlap (for circular arcs and for segments only), (iii) they do not include one another (circle on sphere and line on a plane), (iv) their intersection is non empty. The following 21 cases, classified into four classes as a function of the expected intersection type, have to be accommodated:

[55] Excluded cases are particular cases easily detected and handled using predicates provided by the 3D Spherical Kernel.

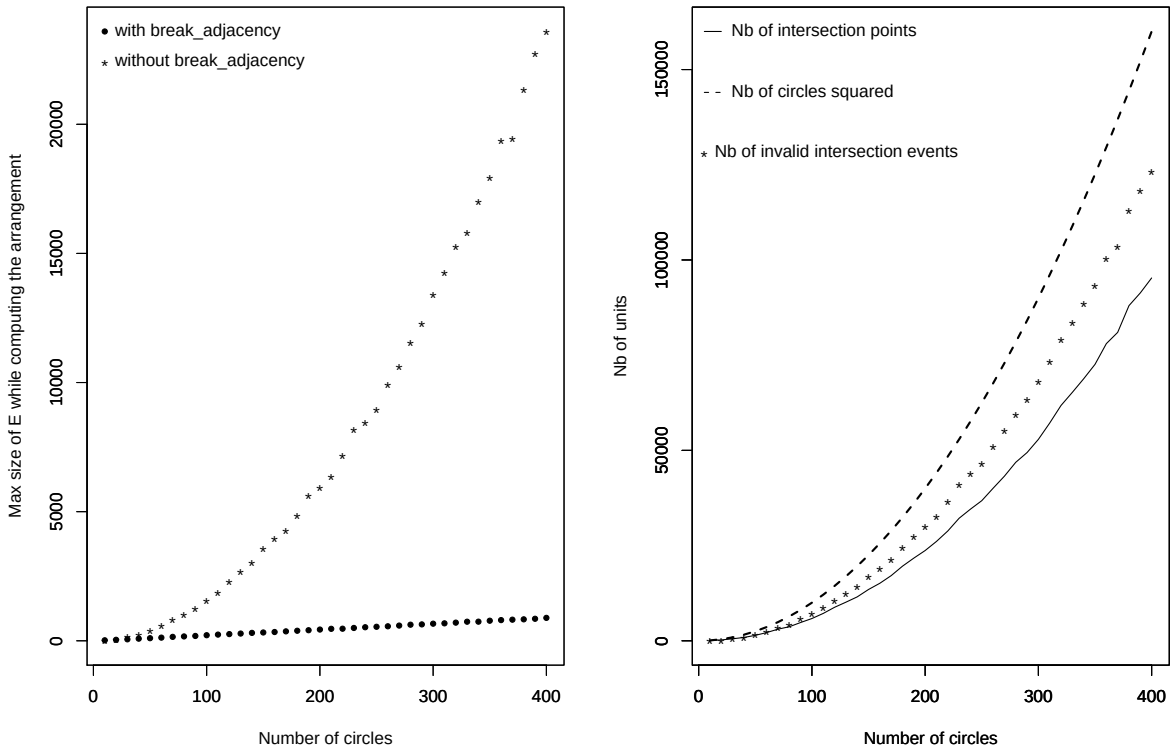


FIG. 5.5 – On the importance of maintaining a linear size event queue. (Left) Maximum number of event sites in $\mathcal{E}$ during the computation of the arrangement. (Right) Number of intersection points for each distribution of circles versus number of intersection events that become non-valid along the algorithm.

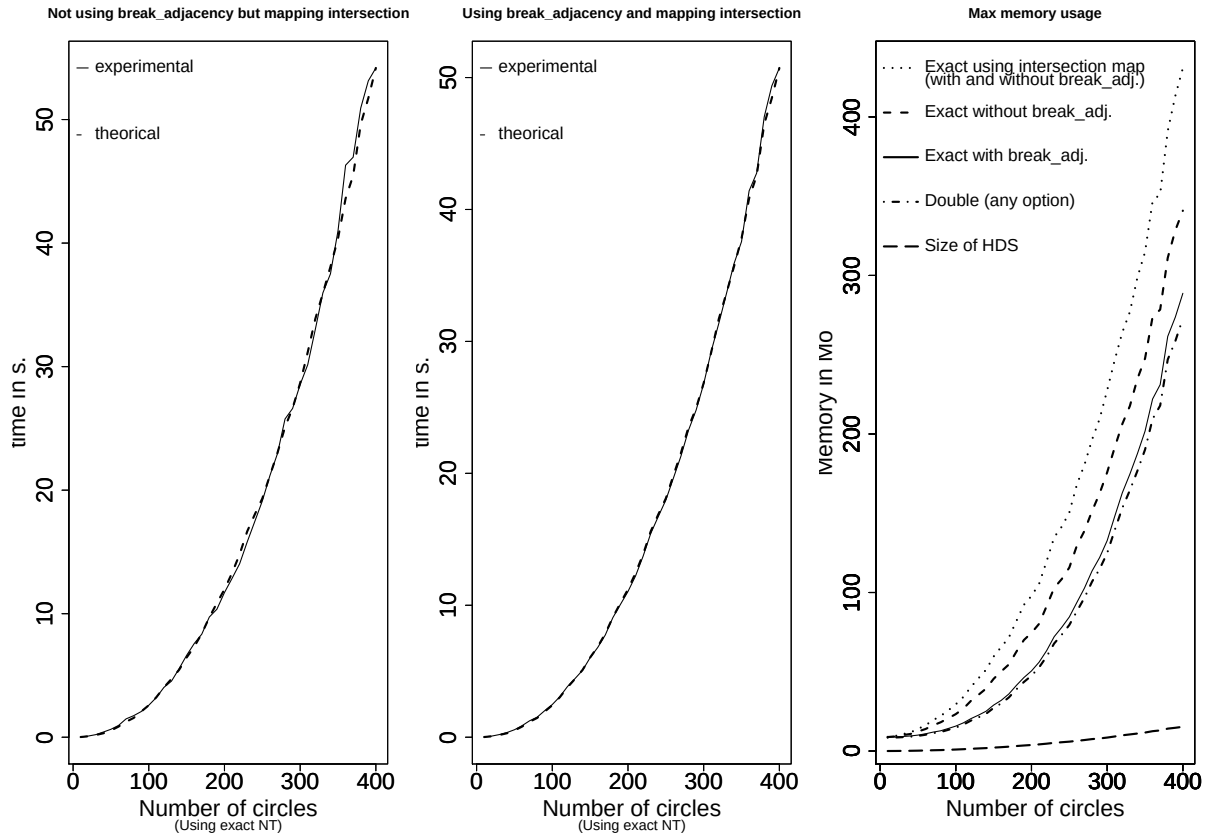


FIG. 5.6 – Assessment of the impact of the linear size event queue: observed vs theoretical complexities for random arrangements.

– **Line**: {plane ∩ plane}
– **Point with rational Cartesian coordinates**: {plane ∩ line, line ∩ line, plane ∩ segment, segment ∩ segment, line ∩ segment }
– **Circle**: { plane ∩ sphere, sphere ∩ sphere}
– **Point(s) whose Cartesian coordinates are algebraic numbers of degree two:** { sphere ∩ line, sphere ∩ segment, circle ∩ circle, sphere ∩ circle, plane ∩ circle, plane ∩ circular arc, sphere ∩ circular arc, circle ∩ circular arc, circular arc ∩ circular arc, circle ∩ line, circle ∩ segment, circular arc ∩ segment, circular arc ∩ line}.

If we detail one construction per class (the most generic one), then other intersections can be inferred from that construction.

▷**Plane ∩ Plane.** To parametrize the intersection line of two planes, we compute the director vector of the line (the cross product of the normal vectors of the planes), and a common point of the two planes.

▷**Plane ∩ Line.** A line can be parametrized by one parameter. So finding the intersection point of a plane and a line is equivalent to solving an equation of degree one.

▷**Sphere ∩ Sphere.** The characterization of the center and the squared radius of the intersection circle of two spheres relies on the following:

Observation 7 *The center of the intersection circle of two spheres S_i and S_j is a linear combination of the two spheres' centers, whose coefficients are quotients of polynomial expressions of degree two of the parameters of the spheres.*

The squared radius of an intersection circle is a quotient of polynomials of the same parameters.

Proof. To find the coordinates of the center c_{ij} of the intersection circle, we seek the intersection between the line joining the centers, and the radical plane of the two spheres.
If $p = (x, y, z)$, we have the following system defining the intersection of a line and a plane:

$$\begin{cases} c_i c_{ij} = \alpha c_i c_j \qquad \forall \alpha \in \mathbb{R} \\ 2 < \overline{p}, c_i c_j > + \overline{c_i}^2 - \overline{c_j}^2 + r_j^2 - r_i^2 = 0 \end{cases}$$

The fact that c_{ij} lies on the radical plane can be written as

$$2 < \overline{c_{ij}}, c_i c_j > + \overline{c_i}^2 - \overline{c_j}^2 + r_j^2 - r_i^2 = 0 \Longleftrightarrow \alpha = \frac{c_j c_j^2 - r_j^2 + r_i^2}{2 c_i c_j^2}$$

Whence

$$c_i c_{ij} = \frac{c_i c_j^2 - r_j^2 + r_i^2}{2 c_i c_j^2} c_i c_j \tag{5.12}$$

For the squared radius r_{ij}^2, we use Pythagorean theorem: $r_{ij}^2 = r_i^2 - c_i c_{ij}^2$. □

Notice that the radical plane is orthogonal to the line joining the centers.

▷**Sphere ∩ Line.** An intersection point between a sphere and a line is characterized by:

Observation 8 *The Cartesian coordinates of intersection points of a sphere and a line are algebraic numbers of degree two in the same extension.*

Suppose the line is defined by $(a_1 t + b_1, a_2 t + b_2, a_3 t + b_3)$ and the sphere by $(x - a)^2 + (y - b)^2 + (z - c)^2 - r^2 = 0$. The Cartesian coordinates of the intersection points of a sphere and a line are $(a_1 t' + b_1, a_2 t' + b_2, a_3 t' + b_3)$ with t' solution of $(a_1 t + b_1 - a)^2 + (a_2 t + b_2 - b)^2 + (a_3 t + b_3 - c)^2 - r^2 = 0$.

Decompositions into θ-monotone circular arcs

Functor `SK::MakeThetaMonotone_3` decomposes a circular arc or a circle into θ-monotone circular arcs. This is trivial for circular arcs on threaded circles. For a circular arc on a normal circle, this operation requires computing the critical points of the circle, and checking whether these points lie on the circular arc. This latter test requires comparing the z coordinate values of the extremities of the circular arc with the values of the critical points. For a circular arc on a polar circle the operation is similar, considering the pole the circle goes through as critical point. For circles, the inclusion test is not necessary.

5.6.2 Predicates

Testing equality

Functor `SK::Equal_3` tests the equality of two objects of the same type from the 3D Spherical Kernel. For objects whose representation is unique, i.e. points and spheres, the test is trivial. For segments, we test equality of endpoints. For planes and lines, we test the non-independence of coefficients of the equations. For circles, we successively test the equality of containing planes, circle centers and squared radii. For circular arcs, we test the equality of the circles and that of the endpoints.

Testing inclusion

Functor `SK::HasOn_3` tests among two objects of the 3D Spherical Kernel assumed to be topologically closed, whether one is included into the second. We examine separately the case of points and the case of remaining primitives.

▷**The case of points.** We wish to test whether a point lies on a sphere, a plane, a line, a segment, a circle or a circular arc. For a sphere and a plane, this operation is equivalent to the sign evaluation of their associated polynomials using `AK::SignAt` at the input point.

For a line, this is trivial using either two planes defining the line or its parametrization. For a segment, we compute the signs of the dot products of the vectors involving the point and the segment endpoints. For a circle, we test if the input point belongs to the plane of the circle and to the diametrical sphere associated to the circle. For a circular arc we test whether the input point belongs to the supporting circle first, and then check whether it belongs to the arc itself. For this latter test, denote s, t, c the source, target, and center of the circular arc and let N stands for the unit normal vector of the positive plane containing the circle. We suppose that $s \neq t$ else testing inclusion into the associated circle is enough. A circular arc is such that we go counter clockwise from its source to its target as imposed by the normal vector N. For two points g and h on that circle such that $g \neq h$, let $S_{gh} = \text{Sign}(< cg \wedge ch, N >)$. S_{gh} indicates if the shortest path to go from g to h is counter clockwise relative to N. If c, s, p or c, t, p are collinear, the input point lies on the circular arc iff $S_{st} \leq 0$. If c, s, t are collinear, the input point lies on the circular arc iff $S_{sp} \geq 0$. If there is no triple of collinear points, all possible cases are summarized in table 5.10.

S_{st}	S_{sp}	S_{pt}	`SK::HasOn_3`
1	1	1	true
1	−1	±1	false
1	1	−1	false
−1	1	± 1	true
−1	−1	1	true
−1	−1	−1	false

TAB. 5.10 – Deciding whether point p belongs to the closed circular arc delimited by s and t.

▷**Other primitives.** The test is trivial using `SK::Equal_3` for circle or a circular arcs with respect to a plane, and for a circular arc on a circle. A plane contains a line, iff the normal vector of the plane and the director vector of the line are orthogonal and the plane contains (`SK::HasOn_3`) one point of the line. A line or a plane contains a segment if it contains its two endpoints. A circle belongs to a sphere S, iff sphere S belongs to the pencil of spheres defined by the diametrical sphere of the circle and its supporting plane.

Testing overlap

Functor `SK::DoOverlap_3` tests whether two circular arcs or two segments overlap—i.e. if they have more than two points in common. The test first checks whether the circles (lines) associated to the arcs (segments) are identical, and if so, of checking the position of the endpoints of one arc (segment) with respect to the other arc using predicate `SK::HasOn_3`.

Testing intersection

Functor `SK::DoIntersect_3` tests if two objects of the 3D Spherical Kernel intersect.

In the following, we suppose that the objects handled are different and that they do not overlap. Similarly to section 5.6.1.0, we focus on the description of one case per class of intersection result, the other cases being handled from the base case together with predicate `SK::HasOn_3`. The base cases are:

– **Plane** $\cap$ **Plane** $\neq \emptyset \Longleftrightarrow$ Normal vectors are not collinear.

– **Plane** $\cap$ **Line** $\neq \emptyset \Longleftrightarrow$ Normal vector and director vector are not orthogonal.

– **Sphere** $\cap$ **Line** $\neq \emptyset \Longleftrightarrow$ Number of real roots of the polynomial specified in Observation 3 must be greater or equal to one.

– **Sphere** $\mathbf{S_i} \cap$ **Sphere** $\mathbf{S_j} \neq \emptyset \Longleftrightarrow \left((c_i - c_j)^2 - r_i^2 - r_j^2\right)^2 \leq 4r_i^2 r_j^2$.

Comparing cylindrical coordinates of points on the reference sphere

The following provides more detail on the strategy used within predicate `SK::CompareTheta_3` and constructor of the class `SK::ThetaRep`. Predicate `SK::CompareTheta_3` is available for points represented under the type `SK::CircularArcPointOnReferenceSphere_3` only. The comparison strategy compares $\tan\theta$ or $\cot\theta$ based on the Cartesian coordinates of the points. We first present the general principle, and then indicate how to accommodate particular cases i.e. the poles.

▷**Quadrants, tangents and cotangents.** Assume the reference sphere is endowed with cylindrical coordinates —which is most convenient when sweeping the sphere with a rotating meridian as in section 5.4. Comparing the values of θ of two points involves inverse trigonometric functions and is non-trivial. To avoid such calculations, we resort to a decomposition of the punctured disk $x^2 + y^2 \leq r_0^2$ into open half-quadrants and line-segments, as depicted on Fig. 5.7. Puncturing the disk corresponds to removing the origin, as poles only project onto it. As explained in the next paragraph, poles deserve a special treatment. More precisely, we decompose the punctured disk into 16 cells: eight open half quadrants, and eight line-segments. Each quadrant is associated an index in the range 1..8 in increasing order, while the segments are assigned a rational index equal to the average of the two neighbors' indices. (For the segment between half quadrants 1 and 8 we take 1/2 as a convention.)

To compare the θ values of two points, we first assign each point to its cell. Computing the sign of x and y, we can easily isolate the containing segment or the two candidate adjacent half quadrants. In this latter case, comparing $\mid x \mid$ and $\mid y \mid$ yields the solution.

Two points falling in different cells are compared from the cells' indices. Inside a cell, two points may have the same θ —all points on a ray meet this condition. To check this condition or find the relative ordering of the points, we have $\tan\theta = x/y$ or $\cot\theta = y/x$. More precisely, as $\tan\theta$ is not defined for $\theta = \pi/2 \mod \pi$, while $\cot\theta$ is not so for $\theta = 0 \mod \pi$, comparison inside two-dimensional cells consists of comparing

$$\begin{cases} \tan\theta \text{ for cells of index } 8, 1, 4, 5, \\ \cot\theta \text{ for cells of index } 2, 3, 6, 7. \end{cases} \tag{5.13}$$

The Cartesian coordinates of a point being algebraic numbers of degree two in the same extension, so are the ratios x/y and y/x —if defined.

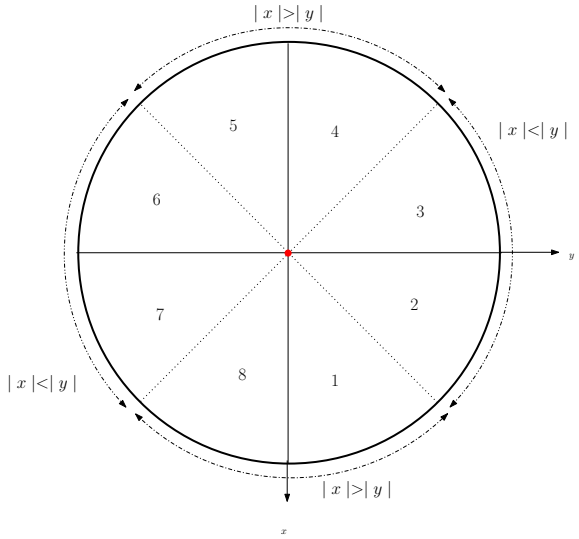


FIG. 5.7 – Decomposition of the punctured disk $x^2 + y^2 \leq r_0^2$ into open half quadrants and line segments.

▷θ-extremal values for polar and bipolar circles. The comparison strategy of event points relying on the comparison of $\tan\theta$ or $\cot\theta$ is valid away from poles but poses a problem at the poles. However, the θ-extremal values of polar and bipolar circles have been defined in section 3.2.2.

In order to have a homogeneous comparison method based on Cartesian coordinates for all critical and intersection points, we associate to each θ-extremal value of a polar or bipolar circle a point away from the pole, whose x and y are such that $\tan\theta = x/y$ and/or $\cot\theta = y/x$. For circle C_{0i}, the two points associated to the two θ-extremal values are $p_1 = (y_i, -x_i, 0)$ and $p_2 = (-y_i, x_i, 0)$. The start and end point of a bipolar circle are such that the θ value associated to the start point is smaller. For a polar circle, while sweeping from its start point to its end points, the half-plane defining the meridian —as intersection with S_0, encounters the center of the circle.

Chapitre 6

Characterization and Selection of Diverse Conformational Ensembles, with Applications to Flexible Docking

Chapter Overview

Having developed an algorithm to compute arrangements of circles on a sphere, we now discuss an application to the problem of selecting diverse conformational ensembles, with applications to flexible-loop docking. The goal of this chapter is to show that a geometric criterion is suitable to select diverse conformers (vs. a classic technique), and that this diversity is a key parameter for one flexible-loop docking algorithm.

Two conformer selection problems phrased as geometric optimization problems are presented in section 6.1, together with a general strategy to solve them, the *greedy strategy*. In section 6.2, we focus on one such problem, namely that of reporting a selection maximizing the VdW surface area of the union of the conformers, and present the protein-protein complexes used for the validation. A geometric and topological assessment of the diversity is presented in section 6.3, while an assessment of the quality of the conformers selected for flexible protein docking is presented in section 6.4. These assessments are conducted by comparing our algorithm, `Greedy`, to a contender named `HClust` based on a hierarchical clustering strategy. Finally, we provide the proofs of the theorems presented in the main text in appendices 6.5 and 6.6, and further discuss the conformer generation methods used in Appendix 6.7.

This chapter corresponds to a paper submitted to *IEEE/ACM Transactions on Computational Biology and Bioinformatics*. This is a joint work with Karine Bastard, Frédéric Cazals, Chantal Prévost and Sushant Sachdeva. In particular, the statement of the combinatorial problems, together with approximation guarantees proved in appendices 6.6 and 6.5 have been worked out by Sushant Sachdeva (IIT Bombay) during an internship under the supervision of Frédéric Cazals.

6.1 Selecting Conformers: the Combinatorial Viewpoint

6.1.1 Arrangements of Balls and Spheres: Volume and Surface Decompositions

The spheres bounding the balls of a collection of conformers induce two decompositions: a decomposition of the volume occupied by the balls; and a decomposition of each sphere into spherical patches. More precisely, consider the three-dimensional domain spanned by the conformers that is the union of their defining balls. The decomposition of this volume induced by the spheres is called a *volumetric arrangement* (or *volumetric decomposition*). This arrangement consists of a collection of cells $\mathcal{A} = \{A_i\}$ such that the interior of each cell is connected. Each such cell is bounded by $2D$ cells, called surface patches, found on

the spheres bounding the balls. On a given sphere, these patches are induced by the intersection circles with neighboring spheres. The collection $\mathcal{P} = \{P_i\}$ of all such patches defines a *surface arrangement* (or *surface decomposition*). See Fig. 2.16 for an illustration.

6.1.2 Optimization Problems

Problems statements. We shall be concerned with two classes of combinatorial optimization problems arising from geometric representations of molecular shapes. To state these problems from the combinatorial viewpoint (see section 6.1.3 for the connexion with conformers), assume we are given a base set $\mathcal{U} = \{U_i\}_{i=1,\ldots,m}$ of interior disjoint *cells* (think cells of the volume or surface arrangement), and a collection of *sets* $\mathcal{C} = \{C_i\}_{i=1,\ldots,n}$ called the *pool* (think conformers), where each set is a union of cells. For a subset $\mathcal{S} \subset \mathcal{C}$, denote $\cup_{\mathcal{S}} C_j$ the union of the sets in $\mathcal{S}$. Cells and sets shall be subsets of $\mathbb{R}^3$, so that the inclusion of a cell U_i in a set C_j is naturally defined.

For the first class of problems, assume we are given a weight function w, i.e. a real valued function defined over the cells. Denote $\binom{\mathcal{C}}{s}$ the subsets of $\mathcal{C}$ of size s. We define:

Problem 1 *Given a weight function w, find a subset $\hat{\mathcal{S}}$ of $\mathcal{C}$ of size s, called the* selection*, such that:*

$$\hat{\mathcal{S}} = \arg\max_{\mathcal{S} \in \binom{\mathcal{C}}{s}} w(\mathcal{S}), \text{ with } w(\mathcal{S}) = \sum_{U_i \subset \cup_{\mathcal{S}} C_j} w(U_i). \tag{6.1}$$

For the second class of problems, assume the weight function depends not only on the cells of the decomposition, but also on the selection $\mathcal{S}$, which we denote $w_{\mathcal{S}}(U_i)$. We wish to solve:

Problem 2 *Given a weight function $w_{\mathcal{S}}$, find a subset $\hat{\mathcal{S}}$ of $\mathcal{C}$ of size s, called the* selection*, such that:*

$$\hat{\mathcal{S}} = \arg\max_{\mathcal{S} \in \binom{\mathcal{C}}{s}} w(\mathcal{S}), \text{ with } w(\mathcal{S}) = \sum_{U_i \subset \cup_{\mathcal{S}} C_j} w_{\mathcal{S}}(U_i). \tag{6.2}$$

Complexity issues. Our problems are intimately related to *max-k cover*. Given a set $\mathcal{U}$ of n points, and a collection $\mathcal{C}$ of subsets of $\mathcal{U}$, max-k cover is the problem of selecting k subset from $\mathcal{C}$ such that the union contains as many points from $\mathcal{U}$ as possible [97, 89]. (There is some confusion in the literature, as this problem is called set cover in [91]. In fact, the partial set cover problem consists of picking the minimum number of sets in $\mathcal{C}$ so as to contain at least k elements from $\mathcal{U}$.) If the weight function w assigns a unit weight to all cells, then Problem 1 reduces to max-k cover. Since this is a **NP**-Complete problem, we cannot expect to have an exact algorithm for our problem that works in time polynomial in both $|\mathcal{C}|$ and s.

On the other hand, for a fixed s, the search space of all possible combinations of conformers is $\mathcal{O}(|\mathcal{C}|^s)$. Hence, for a fixed s the problem is in **P**. However, even for a modest s, the brute force method is too costly to be used in practice. Section 6.1.4 presents an approximate strategy whose time complexity does not grow exponentially with s.

6.1.3 Instantiations to Conformer Selection

Problem 1 from volumetric decomposition. Consider the base set $\mathcal{A}$ whose cells are those of the $3D$ arrangement. In Eq. (6.1), let w be some general function defined on the cells of the volumetric decomposition, for example the standard Euclidean volume. For conformer selection, optimizing the volume of a selection is a direct way to aim for a good spatial diversity, since overlaps between conformers are minimized.

Problem 2 from surface decomposition. Consider the base set $\mathcal{P} = \{P_i\}$ whose cells are those of the $2D$ arrangements. Special cells of this arrangement are those which are exposed, i.e. contribute to the boundary of the union of balls. Focusing on these patches yields an instantiation of Problem 2, the dependence upon the selection $\mathcal{S}$ consisting of discarding the patches which are not exposed with respect to the selection. That is, in Eq. (6.2), $w_{\mathcal{S}}(P_i)$ stands for some general function defined on the surface patches found on the boundary of the union of balls. For example $w_{\mathcal{S}}(P_i)$ = surface area of patch P_i iff

P_i is found on the boundary of the union, and 0 otherwise.
Practically, we shall be dealing with atomic and coarse models. By molecular surface, we refer to the Van der Waals surface for the former, and to the boundary of the union for the latter—coarse models are specified in section 6.2.1. The molecular surface area (MSA in the following) must not be confused with the area of the so called Connolly surface.

Interestingly, maximizing the boundary surface of the selection is an indirect way to ascertain some diversity, since the overlap between conformers is minimized. Notice, though, that as opposed to the volume, the boundary surface area is not a monotonic function of the number of conformers. That is, for two selections S_1 and S_2 with $S_1 \subset S_2$, one has $volume(S_2) \geq volume(S_1)$, a property that may not hold for the boundary surface area.

6.1.4 The Greedy Strategy

The Strategy and its Guarantees

To solve our optimization problems, an obvious approach is the greedy strategy. The greedy strategy performs s steps, selecting at each step an element C_j of $\mathcal{C}$, that has not yet been selected, and that maximizes the sum of the weights of the cells being added. In other words, at each step, the algorithm selects a C_j that maximizes the weight of the union of the C_j.

Unfortunately, the selection obtained this way may not realize the optimum solution. As an example consider Fig. 6.1: For selecting two conformers, the optimum choice has a weight of 14 whereas the greedy method gives us a collection with a weight of 12. To quantify this performance, one resorts to the approximation ratio, that is the worst-case ratio between the solution returned and the optimal one. For max-k cover, this ratio is known to be of $1 - 1/e$, and is tight [66, 149, 91, 89].

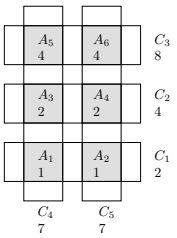


FIG. 6.1 – Selecting two conformers out of $C_1, \ldots, C_5$: the greedy strategy scores 12 (selecting C_3, C_2) while the optimum is 14 (selecting $C_4, C-5$). The shaded regions have the weights as indicated and the unshaded regions have null weights.

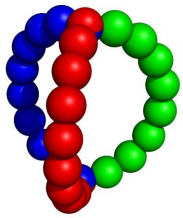


FIG. 6.2 – Three conformers of a fictitious protein loop: The surface of the union of the three conformers defines a double torus, i.e. a surface of genus two. Compare to Fig. 2.18(b).

Application to Conformer Selections

Volumetric decomposition, general weight w. Consider a volumetric decomposition as specified in section 6.1.1. The weighting scheme is called non-negative provided all weights are ≥ 0. The approximation ratio of the greedy strategy and its optimality are usually proved in the uniform weight case [66, 149, 91, 89]. The following theorems, proved in Appendix 6.5, generalize to non-negative weights:

Theorem 6 *Consider a volumetric decomposition with non-negative weights. For Problem 1, the greedy approach has an approximation ratio of* $1 - (1 - 1/s)^s > 1 - 1/e$.

Theorem 7 *The greedy approach cannot perform better than* $1 - (1 - 1/s)^s$.

Surface decomposition, boundary surface weight $w_{\mathcal{S}}$. For volumetric decompositions, the previous bound indicates that one is always above 63% $(1 - 1/e)$ from the optimum. Unfortunately, as shown in Appendix 6.6.1, such a result does not hold for problem 2:

Observation 9 *Consider a surface decomposition. For Problem 2, the greedy approach may have a worst-case approximation ratio as bad as* $1/s^2$.

Practical considerations. In the following, we shall focus on the problem of optimizing the surface area rather then the volume of a selection for two reasons. First, we are not aware of any robust implementation to report the volume of a union of balls, whereas, robust and optimized algorithms exist to handle surface arrangements (as the one developed in chapters 3 and 4).

6.2 Material and Methods

In this section, we introduce the models, concepts and algorithms used to make a geometric and topological assessment (section 6.3), and a docking assessment (section 6.4) of the selections.

6.2.1 Data Sets and Conformer Generation Methods

Protein models and their comparison. We use two protein models: the atomic model and the coarse model. Following [197], given an atomic model, a coarse-residue-based model is obtained by replacing each side-chain by one or two pseudo-atoms, depending on the amino-acid type—the location of the C_α carbon does not change. To distinguish two models of a complex, say 1BTH, we shall use the notation 1BTH-atomic and 1BTH-coarse for the all atom and coarse model respectively.

A classical statistic used for comparing two conformations of the same protein is the C_α-RMSD, that is the standard deviation of the distance between the atomic positions of the C_α carbons of the two proteins. (Below, we shall use the C_α-RMSD to specify algorithm `HClust`, and to compare the selection of `Greedy` and `HClust`.)

While the C_α-RMSD is a good measure to compare two conformations of the same (portions of a) protein, a finer statistic is called for to evaluate the interface of a putative complex proposed by a rigid docking algorithm. To meet this need, we shall use the interface RMSD, denoted I-RMSD. To define it, call the two partners of the complex the ligand and the receptor, and assume that the receptor of the co-crystallized complex has been aligned with that of the putative receptor. The I-RMSD is the C_α-RMSD restricted to selected atoms of the ligand: those identified in the native complex within a distance threshold of 7Å from the receptor [127, 22].

Protein loops. We study four flexible protein loops belonging to the protein-protein interface of four complexes, 1OAZ, 1CGI, 1BTH and 3HHR. For each complex, both the unbound and the bound i.e. co-crystallized structures of the partners are known, and the conformation of the studied loops differs between these two forms. Three of the complexes (1CGI, 1BTH and 3HHR) come from the non-redundant protein-protein docking benchmark [55]. Complexes 1BTH and 3HHR have been identified as difficult cases since no acceptable structure could be predicted in rigid body docking studies [55]. Complex 1OAZ has been added because of the known flexibility of its interface [113].

The four flexible loops differ by size and degree of variation between the bound and unbound forms, as characterized by the C_α-RMSD between the bound and unbound forms. In complex 1BTH, the 10 amino acid (aa) loop of the thrombin mutant bound to the pancreatic trypsin inhibitor undergoes a 5.7 Å deviation; in complex 1CGI, the structure of the 11 aa loop of α-Chymo-trypsinogen bound to pancreatic secretory trypsin inhibitor has not been resolved in the unbound form, showing a high degree of flexibility; in 1OAZ, the 12 aa loop of the Ige Fv Spe7 protein complexed with a recombinant thioredoxin only undergoes a 2.1 Å deviation, while in 3HHR, the 26 aa loop of the human growth hormone bound to the extracellular domain of its receptor presents a deviations of 5.5 Å.

Conformer generations methods. A number of methods exist to generate atomic loop geometries [195, 153, 73, 172, 90]. We selected `Direx` [172] and `Loopy` [195], which respectively generate dense and sparse (exploring more space) ensembles of conformers. (For completeness, an overview of these two methods is presented in Appendix 6.7.1.)

To quantify the diversity of the loop ensembles generated, we computed the MSA of the union of a collection of $n = 500$ conformers for the four models. (The residues involved in the MSA calculation are those from the loops together with the two residues bounding the loop, which are shared by all conformers.) To see that the `Loopy` data set is less redundant and explores more space, observe that the

ratio $MSA(\texttt{Loopy})/MSA(\texttt{Direx})$ spans the range $[1.79, 4.16]$ and $[1.86, 4.60]$ respectively. See Table 6.2 in Appendix for a full report.

Geometry versus energy. Conformer generation methods, when applied to flexible loops, disregard the geometry of the scaffold accommodating the loop. To avoid steric clashes within the loop and between the loop and its scaffold, having computed the potential energy of the system *loop+scaffold* by 100-steps of energy minimization with `GROMACS` [137]. We discard two types of conformers: first, those featuring a *large* short range Lennard-Jones term, which witnesses steric clashes between the loop and the scaffold, and second those with a *large* bonded energy, which witnesses clashes within the loop. Large energies are filtered out using a classical outlier filtering strategy [171] using box-and-whisker plot [46].

6.2.2 Greedy Selection: Implementation

The naive and priority-based versions. Denote I_i the selection of i conformers after i steps of the greedy strategy, and let R_i stand for the remaining candidates. Following Eq. (6.2), the naive way of computing I_i consists of incrementally linearly scanning all possible solutions, that is

$$I_i = \arg \max_{C_j \in R_{i-1}} w(I_{i-1} \cup \{C_j\}). \tag{6.3}$$

As proved in Appendix 6.6, the following complexity is worst-case optimal:

Theorem 8 *The naive version of Algorithm* `Greedy` *has complexity* $O(ns^3)$.

A more elaborate strategy than the naive (brute force) method maintains the increments associated to all candidates, so as to select the best one from a priority queue. To do so, one needs in particular the surface arrangements on all spheres, together with the inclusion information of spherical patches into the other conformers. To account for this information, which encodes the complexity of the surface arrangement, define

$$\tau = \sum_{C_i \in \mathcal{C}} \sum_{S_j \in C_i} \sum_{P_k} \mathbf{1}_{S_j \text{ covers patch } P_k \text{ or } P_k \text{ lies on } S_j}, \tag{6.4}$$

where $\mathbf{1}_X$ is the characteristic function of the Boolean variable X, $\mathcal{C}$ is the set of all conformers, S_j a sphere of a conformer C_j and P_k a patch on a sphere of a conformer in $\mathcal{C}$.

This *priority-based* version, presented in Appendix 6.6, satisfies:

Theorem 9 *The priority-based version of Algorithm* `Greedy` *has amortized complexity* $O(\tau + s \log n)$.

Implementations. The naive version is implemented using the `Delaunay_3` and `Alpha_shape_3` packages of the Computational Geometry Algorithms Library [44]. The priority-based version is implemented using the surface arrangements package described in chapters 3 and 4, which is the only one, to the best of our knowledge, able to compute effectively the exact arrangement of circles on a sphere. In both cases, robustness issues are critical due to the density of conformers manipulated.

6.2.3 Conformer Selection Methods

We compare algorithm `Greedy` against one contender, Algorithm `HClust`, which is a hierarchical agglomerative clustering [103] method based on the *average linkage*, used for protein-protein docking [22]. (We also tested the single linkage and complete linkage strategies, which performed equally w.r.t. the MSA—data not shown.) Given a dissimilarity measure between two clusters (i.e. groups of conformers), the algorithm generates a binary tree encoding a sequence of nested partitions of the n conformers. Notice the coarser partition features one cluster containing the n conformers, while the finer partition features n cluster of a single conformer. As dissimilarity measure, we use the C_α-RMSD between pairs of conformers. Cutting this binary tree at an appropriate level provides the number of desired conformers, since we select one representative within each cluster. The representative selection was carried out through a two-stage process, namely (i) a fictitious *average* loop is computed: for k conformers each consisting of p balls centered at $c_{i,j}$, with $i = 1, \ldots, k$ and $j = 1, \ldots, p$, the fictitious loop consists of p balls centered at $\overline{c_j} = (\sum_{i=1,\ldots,k} c_{i,j})/k$; (ii) the representative is taken as the conformer from the cluster having the least C_α-RMSD with this fictitious loop.

6.3 Diverse Ensembles: Geometric and Topological Assessment

In this section, we discuss geometric and topological quantities to characterize the diversity of an ensemble, and compare those produced by the `Greedy` and `HClust` algorithms on four protein models.

6.3.1 Statistics of Interest: Geometry vs. Topology

Comparing MSA. We first use the MSA of the union of conformers selected by either `Greedy` or `HClust` as a diversity criterion of the selection. To see how, for a given selection method M (G: greedy; H: hierarchical), let $\mathcal{N}_M = \{I_1, \ldots, I_n\}$ be a *collection of selections* of increasing size, i.e. selection I_i contains i conformers. The greedy strategy provides a *nested* collection of selections, since the selection I_{i+1} of size $i+1$ is the selection I_i of size i to which an additional conformer has been prepended. The nestedness does not hold for algorithm `HClust`, though. As explained in section 6.2.3, one indeed gains one conformer by splitting one cluster K (corresponding to a node n_K in the binary tree) into two clusters K_1 and K_2 (the sons of node n_K in the binary tree). But the representative conformer C_i of cluster K may not be that of the cluster (K_1 or K_2) the conformer C_i belongs to.

To compare two collections of selections, both for the atomic and the coarse models, we report two sets of values. Let R_M be the maximum MSA obtained over all selections in $\mathcal{N}_M$, that is $R_M = \max_{I_i \in \mathcal{N}_M} MSA(I_i)$. First, we focus on the maxima of MSA reached, that is on the ratio R_G/R_H. Second, denote n_{H_x} the smallest number of conformers required by algorithm H to get a MSA (say A) equal to $x\%$ of its maximum. Then, denote n_G the least number of conformers required by the greedy strategy to get a MSA greater or equal to A. We report n_{H_x}/n_G and n_{G_x}/n_G, for $x = 100\%$ and $x = 95\%$.

Comparing the topology. Apart from the MSA, an interesting information about the selection is the topology of the union of the balls of the conformers selected. The boundary of the union of conformers defines a compact orientable surface, possibly non-connected—as the union of conformers may isolate one or several hole(s). By the theorem of classification of connected compact orientable surfaces [108], each such connected component is a sphere with a number $g \geq 0$ of handles attached: for example, the sphere, one-torus, two-torus respectively correspond to $g = 0, g = 1, g = 2$. To characterize these situations, one resorts to Betti numbers, which are respectively $\beta_0 = 1, \beta_1 = 2g, \beta_2 = 1$. Alternatively, one can compute the Euler characteristic of the surface, that is $\chi = \beta_0 - \beta_1 + \beta_2 = 2 - 2g$, with g the genus of the surface. Fig. 6.2 presents and example selection of $g+1$ conformers anchored at the loops extremities, and defining a genus g surface ($g = 2$ here). We shall compare the variation of β_1 for $n_{G_{100\%}}$ conformers selected by algorithm `Greedy` and `HClust`.

Comparing the C_α-RMSD. The measures just described are somewhat tailored to our selection algorithm, since `Greedy` aims at maximizing the MSA. To provide a fair comparison, we thus also report on a measure based upon the C_α-RMSD used by `HClust`. More precisely, to make an assessment on the diversity of a given selection, we investigate the range spanned by the C_α-RMSD of loops from this selection with respect to the native co-crystallized loop. Notice that since the C_α carbons are common to an atomic model and its coarse representation (see beginning of section 6.2.1), algorithm `HClust` reports the same selection for the atomic and coarse models, while algorithm `Greedy` reports two different such selections.

6.3.2 Results

Comparing MSA. In the following, we refer to Tables 6.3 and 6.4 in Appendix 6.7.2. Speaking of the max values R_G and R_H, one observes that `Greedy` yields an increase in the range 9-13% for (`Direx`, atomic), 11-15% for (`Direx`, coarse), 14-54% for (`Loopy`, atomic) and 25-56% for (`Loopy`, coarse). More interesting is the speed at which the methods peak, as can be seen from the ratios n_{H_x}/n_G and n_{G_x}/n_G, for $x = 100\%$. The number of conformers required by algorithm `Greedy` to match the maximum of algorithm `HClust` incurs a dramatic k-fold reduction, where k spans the following ranges (decimals omitted): 9-154 (`Direx`, atomic), 1-160 (`Direx`, coarse), 4-79 (`Loopy`, atomic), 10-79 (`Loopy`, coarse). On the other hand, as can be seen from the plot Fig. 6.3 (a typical one), the asymptote is reached rather fast for all algorithms. Focusing on 95% of the max MSA obtained, the ratios $n_{H_{95\%}}/n_G$ now span the following

ranges: 1-6 (`Direx`, atomic), 1-3 (`Direx`, coarse), 3-28 (`Loopy`, atomic), 3-11 (`Loopy`, coarse). These values call for two conclusions.

First, consider the variation of the ratio $(n_{H_{100\%}}/n_G)/(n_{H_{95\%}}/n_G)$ for the `Direx` and `Loopy` data sets. This ratio is clearly much higher for `Direx` than `Loopy`, which has the following explanation: for a dense data set such as `Direx`, algorithm `HClust` selects pretty fast good representatives accounting for most of the MSA (95% here); but further selections fail at significantly increasing the MSA, as seen from much higher ratios $n_{H_{100\%}}/n_G$. On the other hand, algorithm `Greedy` consistently selects the conformers optimizing the increase of MSA. Second, focus on the statistic $n_{H_{95\%}}/n_G$ for the `Direx` and `Loopy` data sets. This ratio is much higher for the latter data set, which shows that algorithm `Greedy` is also better at selecting large increments of MSA within data sets of conformers exploring more space.

Variations of Betti numbers. For a qualitative explanation of these facts[56], consider the variation of the first Betti number β_1 for the two algorithms. As seen from Tables 6.5 and 6.6 in Appendix 6.7.2, the selection obtained with algorithm `Greedy`, when compared to that obtained with `HClust`, typically features an average value of β_1 which is about 12 times higher for `Direx` and 5 times higher for `Loopy`.

The variation of β_1 is illustrated on Fig. 6.4, which is also a prototypical plot. Indeed, all such curves feature a sharp peak, followed by a plateau, and algorithm `Greedy` outperforms its contenders in both regimes. The sharp rise at the beginning of the selection process corresponds to the choice of *independent* conformers i.e. conformers that do not overlap excepted at their extremities. Such conformers minimize the overlap between balls—in agreement with the criterion targeted by algorithm `Greedy`. Once the maximum has been reached, the conformers selected bridge gaps, whence a decrease in β_1. The sharp decrease stops as soon as the union of the selection is *essentially* a topological ball. The union still features small handles. Such handles get created and destroyed upon addition of new conformers, whence the minute fluctuations about the horizontal asymptote of the graphs displaying the variation of β_1.

The variation of β_1 (see plots in Appendix 6.7.3) also sheds an interesting light on the relative flexibility of the four loops. As for the MSA variation, the curve of 3HHR clearly shows that the $n = 500$ conformers are not enough in the `Loopy` data set. We also speculate that a comparison between the maximum value of β_1 obtained and the ensuing plateau encode interesting features on hinges found in the structure. To confirm these statements, though, one would need to geometrically qualify the geometry of the handles defining a basis of the homology groups.

Comparing the C_α-RMSD. Let δ_G and δ_H be the range of C_α-RMSD with respect to the native bound loop spanned by the conformers from the selection achieved by `Greedy` and `HClust` respectively. As reported in Appendix on Tables 6.7 and 6.8, for models 1BTH, 1CGI and 1OAZ, the ratio $(\delta_G - \delta_H)/\delta_H$ spans the range [-0.07, 0.63] when `Greedy` is run on atomic models, and [-0.13, 0.76] when `Greedy` is run on coarse models. Thus, apart from occasional minor losses, `Greedy` outperforms `HClust` regardless of the data set and of the representation level, even though the C_α-RMSD is not the criterion targeted. For 3HHR for the `Loopy` data set, while the two algorithms perform almost equally for the `Direx` data set, `Greedy` is clearly outperformed for the `Loopy` one. This owes to the length of the loops and its flexibility, and shows the independence of the MSA and C_α-RMSD criteria on this kind of system.

6.4 Diverse Ensembles: Docking Assessment

In this section, report docking results for `1BTH`,`1CGI` and `1OAZ`, based on selections of coarse conformers provided by algorithms `HClust` or `Greedy`. Following the discussion in section 6.3, we focus on conformer pools generated by `Loopy`, which are more diverse, and we omit complex 3HHR, since generating a representative pool of conformers for its long flexible loops of 26 amino is a problem in itself.

[56] The analysis is qualitative for the following reason: a handle accounts for one unit in the β_1 number, whatever its size. That is a large handle coming from a whole loop (as on Fig. 6.2) has the same weight as a small one coming from the creation of a local cycle between atoms of say a side-chain and the backbone. While computing the Betti numbers is by now standard—we use the α-shapes-based algorithm of [72]. The calculation of a geometrically pleasant basis of the homology groups is still an active area of research [54].

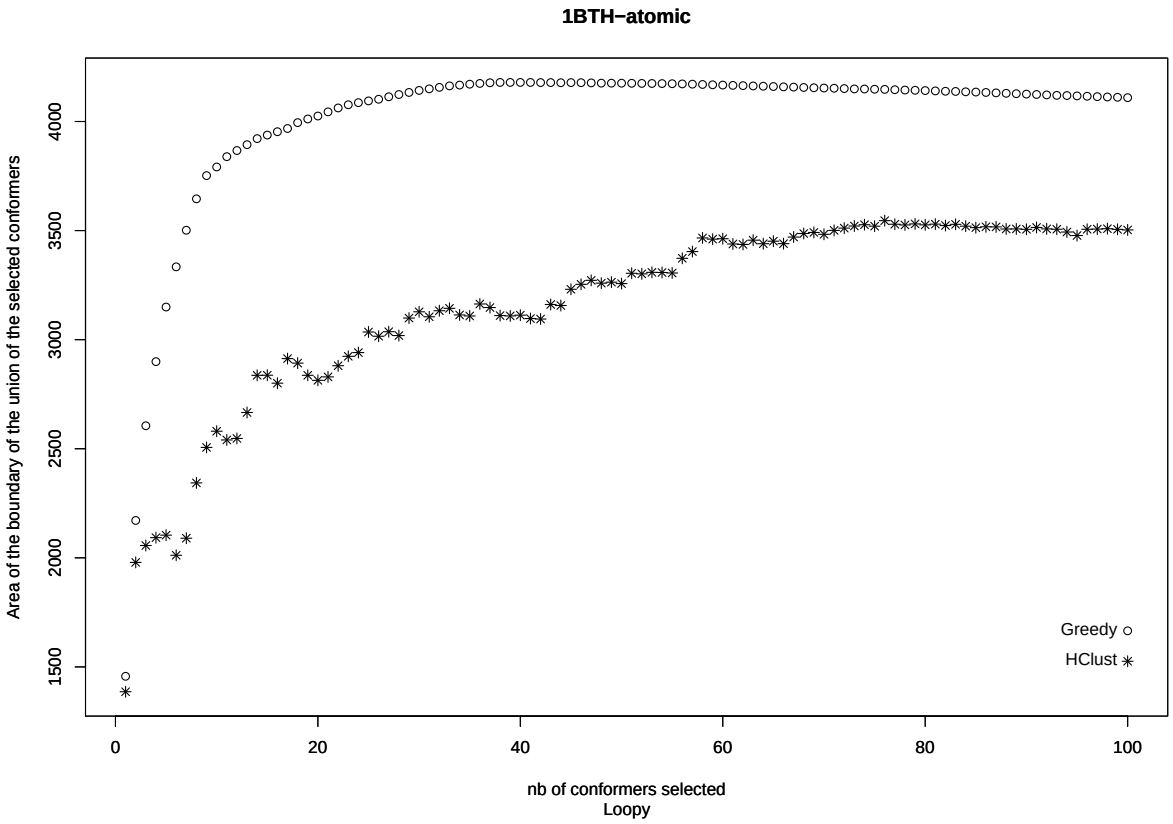


FIG. 6.3 – `Loopy` data set. Variation of MSA with the selection size: conformer selections of `Greedy` always expose more surface than those of `HClust`, a diversity criterion.

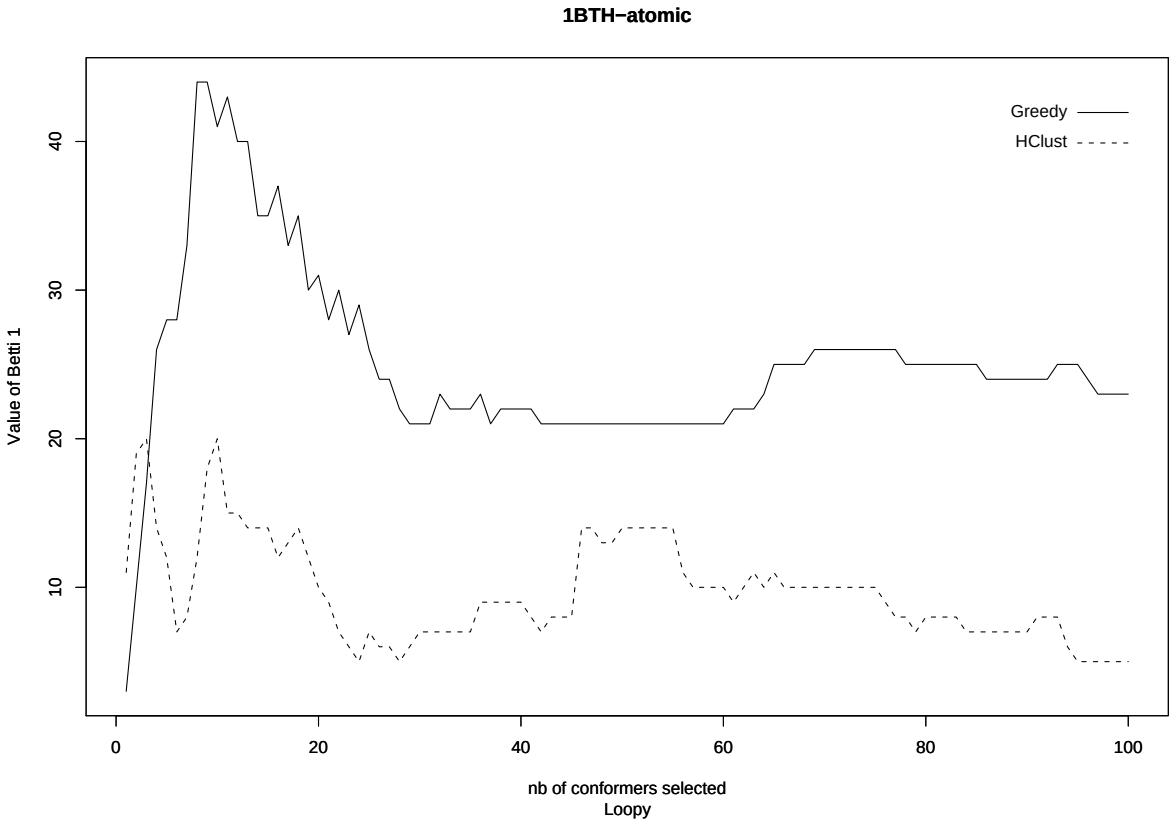


FIG. 6.4 – `Loopy` data set. Variation of the first Betti number β_1 with the selection size: more handles are observed for conformer selections of `Greedy` than for those of `HClust`.

Loopy	%	Direx	%
1BTH-atomic	90.73	1BTH-atomic	99.33
1CGI-atomic	95.73	1CGI-atomic	100
1OAZ-atomic	82.85	1OAZ-atomic	99.77
3HHR-atomic	66.46	3HHR-atomic	99.74
1BTH-coarse	87.72	1BTH-coarse	98.88
1CGI-coarse	91.23	1CGI-coarse	99.7
1OAZ-coarse	83.12	1OAZ-coarse	99.44
3HHR-coarse	55.03	3HHR-coarse	99.99

TAB. 6.1 – Percentage of the maximum MSA achieved by `Greedy` realized by the selection of only ten loops. Ten conformers seem not to be sufficient to cover available space for the loop of 3HHR sampled with `Loopy`

6.4.1 Docking Protocols

In this section, we want to show that the diverse selection of `Greedy` can enhance complex prediction of a flexible-loop docking algorithm.

Specifying the ligand and the receptor. We ran docking simulations on three complexes to validate the conformer selection strategy based upon MSA maximization. Each complex was decomposed into one rigid protein called the ligand (L), and one flexible called the receptor. The receptor itself decomposes into a rigid template (T) and a flexible loop (F). While performing flexible protein docking with conformer ensembles, the strategy consists of using a conformer ensemble for the flexible loop F, this ensemble being selected from a larger pool. Thus, specifying a docking protocol requires specifying the triple T/L/F.

To see how, recall that a binary complex used for docking validation features two molecules which have been crystallized under two forms: on their own, i.e. the unbound forms, and in complex i.e. the bound forms. Thus, to specify the rigid parts (T and L), we provide a tag indicating the origin of the partner, namely U for Unbound and B for Bound. To specify the ensemble associated to F, we provide three pieces of informations: (i) the bound/unbound tag which indicates the loop geometry used to generate the pool of conformers (ii) the algorithm used to select the conformers from this pool (`HClust` or `Greedy` here), and (iii) the selection size. For example, F=B-`Greedy`-10 refers to 10 conformers selected by algorithm `Greedy`, out of a pool of conformers generated from the Bound structure of the receptor. As a second example, F=B-1 means that a single loop has been used, the Bound one.
To summarize, we report on the following six docking protocols: three using the Bound form of the receptor, namely B/B/B-1, B/B/B-`HClust`-10, B/B/B-`Greedy`-10; and three using the Unbound form of the receptor, namely U/B/B-1, U/B/B-`HClust`-10, U/B/B-`Greedy`-10.

Two comments on these protocols are in order. First, notice that the incentive for using the Bound conformation of the flexible region to generate the conformers is the following: for very flexible systems, such as 1CGI mentioned in section 6.2.1, the reconstruction of the unbound conformation of the flexible loop from the crystallographic data is not possible. (If the conformation of the loop changes across the crystallographic units, the signal is not strong enough for the reconstruction to be carried out.) Second, the particular protocols B/B/B-1 and U/B/B-1 can be seen as sanity checks, since in using only the native loop conformer, one expects the docking process to yield satisfactorily complexes.

About the pool size and the number of conformers. For each flexible loop, a pool of $n = 500$ conformers was generated using `Loopy` [195], from which $s = 10$ were selected using the `Greedy` and `HClust` algorithms. Following [22], the choice of $s = 10$ comes from a trade-off between the requirement to have a representative selection, and the computational resources. As seen from Table 6.1, we observe that for all systems but `3HHR`, the MSA of the union of the first 10 conformers selected by `Greedy` realizes more than 80% of the maximum MSA observed along the iterative greedy selection up to n conformers. The same table shows that a mere 10 conformers is not enough to represent the flexible loop of `3HHR`.

6.4.2 Initial Conditions for a Protocol

For a given protocol, we ran N_t docking tests using algorithm `ATTRACT` [197], which is based on the coarse protein representation recalled in section 6.2.1. This algorithm has been adapted to handle multiple copies of a flexible loop in [22]. In this scheme, using Boltzmann's principle, each copy is assigned a fitness score (between 0 and 1) based upon its interaction energy with the receptor. Each docking test corresponds to a specific position and orientation of the ligand with respect to the receptor. Given these initial conditions, ATTRACT performs a sequence of minimizations so as to explore the six degrees of freedom of the ligand. At each stage, the energy of each conformation of the complex is computed. Upon termination, the loop selected is that having the highest fitness score. An assessment of the quality of the proposed complex is then based upon two figures: (i) the interaction potential energy E of the complex (ii) the I-RMSD of the atoms of the ligand.

For the N_t tests associated to a given protocol, the plot of the pairs (E,I-RMSD) defines the energy landscape of the docking experiment. Thus, a conformer ensemble is satisfactory if the landscape features at least one conformer yielding a large number of points (E,I-RMSD) next to the bottom left corner of the energy landscape. Practically, we represent an energy landscape using buckets. For a given bucket B_i and conformer C_j, let $s_{i,j}$ be the number of times conformer C_j yields a complex whose energy and I-RMSD fall in bucket B_i. (Notice that $\sum_{i,j} s_{i,j} = N_t$.) Finally, for a given bucket B_i, denote l_i the index of the conformer that yields the largest value of $s_{i,j}$, and let $r_i = \sum_{j=1,\ldots,n;j\neq l_i} s_{i,j}$. In bucket B_i we display the score s_{i,l_i}, together with r_i when $r_i \neq 0$. The color used is that associated to conformer l_i, with one color per conformer.

6.4.3 Results

For each selection method, a total of $N_t \sim 35,000$ docking tests were run using the same $s = 10$ selected conformers. To analyse the results, we plot the portion of the energy landscape corresponding to a I-RMSD ≤ 15Å with a negative energy. An example of such a plot is presented on Fig. 6.5, and we refer the reader to Appendix 6.7.4 for the remaining plots. In analyzing landscape, since the docking process is coarse, we just aim to identify conformers with good potential for atomic docking process. We thus skip a detailed atomic discussion of the results, a notoriously difficult task [139]. For six docking protocols examined (three systems, Bound and Unbound receptors for each), we argue that the results decompose as follows: three favorable to `Greedy`, two ties, one favorable to `HClust`.

Docking improved using `Greedy`. For complex `1BTH`, the docking protocol B/B/B-`Greedy`-10 leads to 161 predictions with I-RMSD $\in (1;2]$ and energy below -21 units. Only 3 such predictions are found using docking protocol B/B/B-`HClust`-10. See Fig. 6.5. The same kind of result can be observed when using the unbound form of the receptor of `1CGI`. Indeed, U/B/B-`Greedy`-10 leads to 160 predictions with I-RMSD $\in (3;4]$ and energy below -15 units, while U/B/B-`HClust`-10 yields 18 such predictions. For complex `1OAZ`, neither U/B/B-`Greedy`-10 nor U/B/B-`HClust`-10 leads to a high number of predictions below 5Å I-RMSD and below -15 energy units: 5 with one loop, and 13 with 5 different loops respectively. But considering predictions with I-RMSD in interval $(5,7]$ and energy below -15 units, U/B/B-`Greedy`-10 leads to 195 predictions with the same loop while U/B/B-`HClust`-10 leads to one such prediction.

Tie between `Greedy` and `HClust`. The results of the docking involving the unbound form of the receptor of the complex `1BTH` are more ambiguous. U/B/B-`HClust`-10 leads to two predictions with the same loop with I-RMSD below 5 Å and below -15 energy units, while U/B/B-`Greedy`-10 leads to no such prediction. When considering predictions with I-RMSD in interval $(5,7]$ and energy below -15 energy units, both U/B/B-`Greedy`-10 and U/B/B-`HClust`-10 lead to more than 300 such predictions. The results of the docking involving the bound form of the receptor of the complex `1OAZ` needs further scrutiny to detect whether some improvement is achieved by B/B/B-`Greedy`-10 compared to U/B/B-`HClust`-10. Indeed, no highly populated region with low energy and low I-RMSD clearly emerges.

No improvement while using `Greedy`. The results of the docking involving the bound form of the receptor of the complex `1CGI` are more favorable to the docking protocol B/B/B-`HClust`-10. Nevertheless, it must be noticed that even if B/B/B-`HClust`-10 leads to a larger number of good predictions, the energy of these predictions is much higher than those obtained with the native loop in the protocol B/B/B-1.

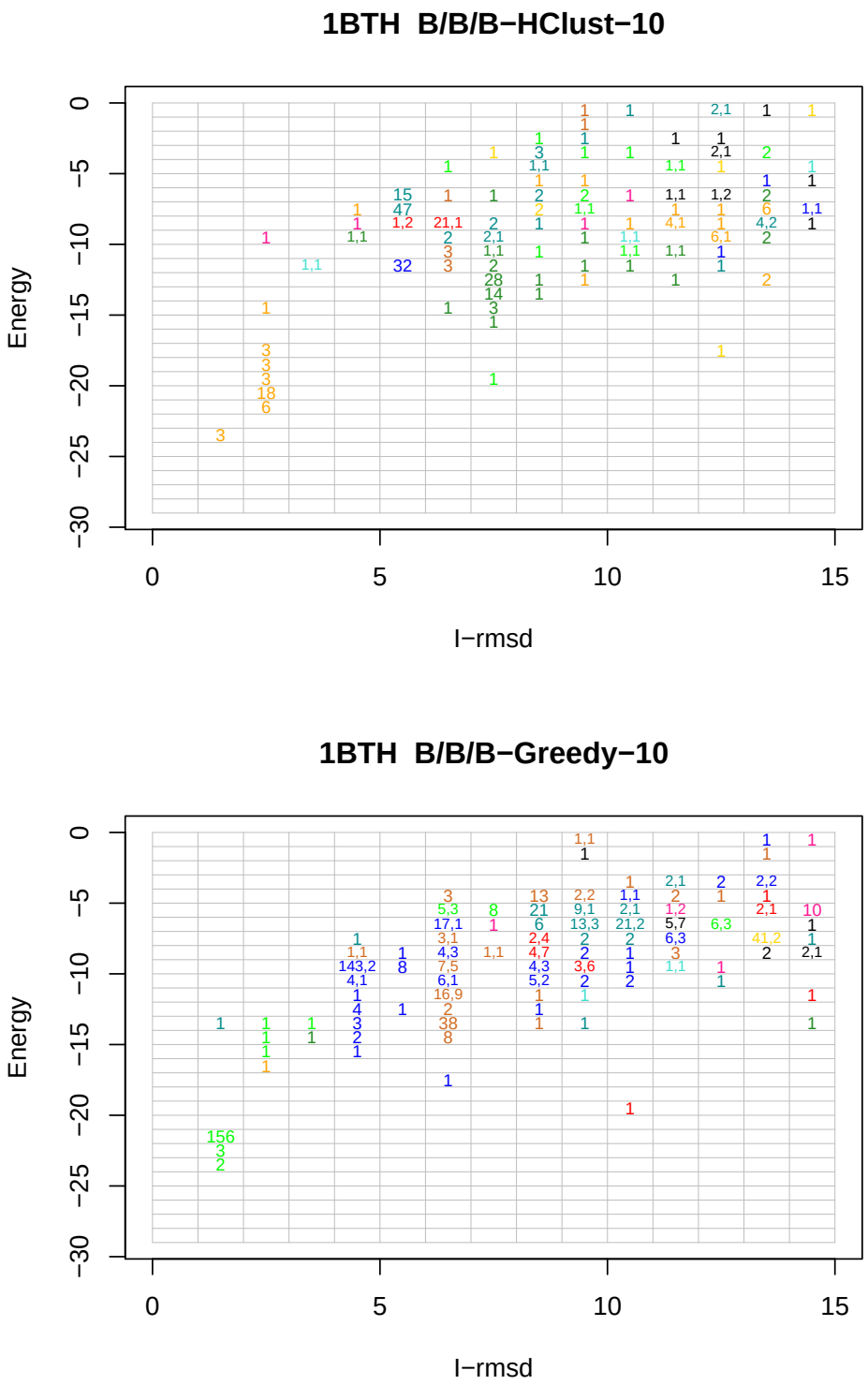


FIG. 6.5 – 1BTH: binning the docking tests using the Bound form of the receptor. The color associated to a bucket corresponds to the loop yielding the highest score in this bucket. A highly populated region next to the bottom left corner of the plot indicates that a satisfactory conformer was present. See text for details.

6.4.4 Running Times

Some comments are in order regarding the computational cost of the docking and selection algorithms. The naive and priority-based selection algorithms were run on a PC computer equipped with a Xeon processor (quadcore) at 2.33GHz, and 16GB of RAM. Surprisingly, we observed a factor of one order of magnitude in favor of the naive implementation, and in particularly when s is much smaller than n. Although asymptotically optimal, the problem of the priority-based algorithm lies in the computation of the arrangement: for n conformers, the size of the arrangement on a given ball may be (and actually is on some examples) as high as n^2. Using the naive implementation, the selection of ten loops requires about half an hour using the all atoms representation, and five minutes using the coarse grain representation.

As reported in [22], the docking algorithm using ten conformers requires 31 hours on a 2.2 GHz Athlon PC.

6.5 Appendix: Volumetric Decompositions

6.5.1 Greedy: Approximation Factor and Optimality

Approximation Factor

We shall use the following notation. The conformer selected at the k^{th} step is denoted C_k, and the weight of the optimum set of conformers OPT. Also, let us denote by $w^*(C_k)$ as the sum of the weights of the new elements in C_k that have not been covered in $C_j, 1 \leq j < k$ (i.e. the weight increment at step k). We need the following lemma in order to prove theorem 6.

Lemma 3 *For $1 \leq k \leq s$, the following holds:*

$$w^*(C_k) + \frac{1}{s}\sum_{j=1}^{k-1} w^*(C_j) \geq \frac{OPT}{s}. \tag{6.5}$$

Proof. At the k^{th} step, we select C_k that maximizes the weight of the new cells U_i being covered. The weight of the cells that are covered by the optimum solution but not yet covered by the $(k-1)$ is at least

$$OPT - \sum_{j=1}^{k-1} w^*(C_j) \tag{6.6}$$

Since w is non-negative, the union-bound property states that for any collection of conformers $C_1, \ldots, C_p$, one has $w(C_1 \cup \cdots \cup C_p) \leq \sum_{i=1,\ldots,p} w(C_i)$. Since all the cells involved in Eq. (6.6) are covered by the optimum set of conformers, by the union-bound property, there must exist one conformer,not yet selected, that covers these new cells with total weight at least

$$\frac{1}{s}\left(OPT - \sum_{j=1}^{k-1} w^*(C_j)\right). \tag{6.7}$$

Since C_k maximizes the weight of the new cells being covered, we must have

$$w^*(C_k) \geq \frac{1}{s}\left(OPT - \sum_{j=1}^{k-1} w^*(C_j)\right). \tag{6.8}$$

Rearranging completes the claim.
□

Remark. The non-negativity assumption is critical in the proof of Lemma 3. As a counter-example, consider the sets $C_1 = \{e1, e2\}, C_2 = \{e2, e3\}$ with $w(e1) = w(e3) = 1$ and $w(e2) = -1$. The union-bound fails for $w(C_1 \cup C_2)$.

Using Lemma 3, the proof of Thm. 6 goes as follows:
Proof.(Thm. 6) Multiplying the inequality obtained in the previous lemma by $\left(\frac{s-1}{s}\right)$ and adding to the inequality for step two, we get

$$w^*(C_1) + w^*(C_2) \geq \left(1 + \left(\frac{s-1}{s}\right)\right)\frac{OPT}{s}$$

We multiply this equation again by $\left(\frac{s-1}{s}\right)$ and add to the equation for step three, and so on. We get the following,

$$\sum_{j=1}^{k} w^*(C_j) \geq \left(1 - \left(\frac{s-1}{s}\right)^k\right) OPT$$

For $k = s$, we get,

$$\frac{\sum_{j=1}^{s} w^*(C_j)}{OPT} \geq \left(1 - \left(\frac{s-1}{s}\right)^s\right)$$

The left hand side is the ratio of the weight of the subset of $\mathcal{C}$ chosen by the greedy approach and the optimum solution i.e. that approximation factor and hence we have the above theorem. The fact that the above ratio is greater than $1 - \frac{1}{e}$ for all s is a trivial exercise. □

Optimality

To prove Thm. 7, we construct tight examples for the greedy approach.

Proof.(Thm. 7) Fix a given s. We shall construct an example where the greedy approach can achieve an approximation ratio arbitrarily close to $1 - (1 - \frac{1}{s})^s$.

Let

$$\mathcal{A} = \{A_i\}_{i=1,\ldots,(s^2+s)}$$

$$\forall i,j \text{ s.t. } 0 \leq i < s, 1 \leq j \leq s,\ w(A_{i.s+j}) = \frac{1}{s^2}\left(\frac{s-1}{s}\right)^i$$

$$\forall j \text{ s.t. } 1 < j \leq s,\ w(A_{s^2+j}) = \frac{1}{s}\left(\frac{s-1}{s}\right)^s - \epsilon$$

The conformers are defined as follows

$$\mathcal{C} = \{C_i\}_{i=1,\ldots,2s}$$

$$\forall i \text{ s.t. } 1 \leq i \leq s,\ C_i = \bigcup_{j=(i-1).s+1}^{i.s} A_j$$

$$\forall i \text{ s.t. } \leq i \leq s,\ C_{s+i} = \bigcup_{j \equiv i \pmod{s}} A_j$$

Simple calculations lead us the following total weights for the conformers

$$\forall 1 \leq i \leq s,\ w(C_i) = \frac{1}{s}\left(\frac{s-1}{s}\right)^{i-1}$$

$$\forall 1 \leq i \leq s,\ w(C_{s+i}) = \frac{1}{s} - \epsilon$$

The optimum choice of $\mathcal{S}$ with $|\mathcal{S}| = s$ is clearly $\{C_i\}_{i=s+1,\ldots,2s}$ with total weight $1 - s\epsilon$, whereas the greedy method would choose $\{C_i\}_{i=1,\ldots,s}$, with a maximum weight of $1-(1-\frac{1}{s})^s$, giving an approximation factor is arbitrarily close to $1 - (1 - \frac{1}{s})^s$. □

6.6 Appendix: Surface Decompositions

6.6.1 Approximating Factor for Problem 2

The following counter-example sets the approximation ratio for the greedy algorithm for the boundary surface case.

Proof.(Observation 9) Consider a large ball B, and place s small non-intersecting balls $(B_1, \ldots, B_s)$ with their centers on the surface of B. The surface of each B_i is now divided into 2 patches. To the patch which lies inside B, we assign a weight of s. To each surface patch of B covered by some B_i, we assign a weight of $1 + \epsilon$. All other surface patches are assigned a weight of 0.

The greedy strategy would first pick B because it has the largest exposed weight of $s(1 + \epsilon)$. Now picking any $s-1$ of the B_i's would leave us with an exposed weight of only $s(1+\epsilon)-(s-1)(1+\epsilon) = 1+\epsilon$. On the opposite, a selection of the s smalls balls would have given us total exposed surface weight of s^2. This approximation factor arbitrarily close to $1/s^2$. □

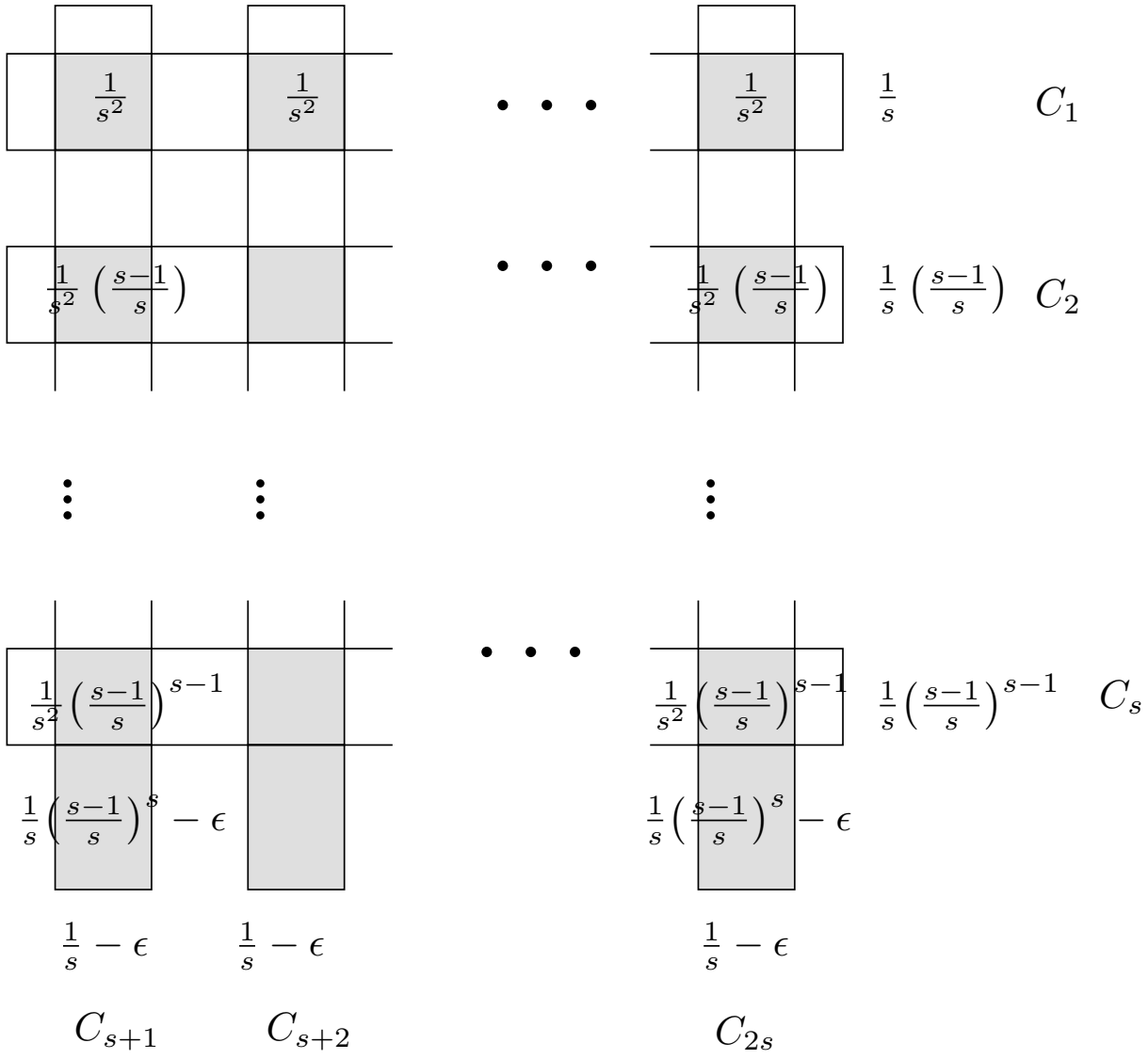


FIG. 6.6 – A tight example for the greedy strategy

6.6.2 Naive Algorithm for Surface Arrangement

Proof.(Thm. 8) To compute $w(G_{i-1} \cup \{C_j\})$, one needs the boundary of the corresponding balls. For a collection of i balls, this is done in worst-case optimal time of $O(i^2)$, by first computing the regular triangulation of the balls, and then by retrieving the boundary of the union from the α-complex with $\alpha = 0$ [7]. The overall complexity is thus bounded by $\sum_{i=1}^{s}(n-i+1)O(i^2)$, whence the claim. □

6.6.3 Priority-based Algorithm for Surface Arrangement

Notation. If X refers to a collection of conformers, $\cup X$ refers to the domain covered by these conformers, and $\partial \cup X$ refers to the boundary of the union of conformers in X. We shall abuse this notation, as we shall also use $\partial \cup X$ to refer to the finite number of spherical patches bounding the boundary of the union. Notice though, that the inclusion of a region r in the geometric boundary will be denoted $r \subset \cup X$, while the membership to the finite set describing this boundary will be denoted $r \in \partial \cup X$.
Computing the surface decompositions. Using the algorithm described in chapters 3 and 4), we compute the arrangement on each sphere, induced by the intersection circles with other spheres. The output consists of:
– $D(S_i) = \{P_k\}$: patches on sphere S_i,
– $H(P_k)$: collection of spheres covering patch P_k,
from which we easily derive:
– $K(S_i)$: collection of patches covered by sphere S_i,
– $B(C_i)$: patches contributing to the boundary of conformer C_i.
Algorithm. We now present Algorithm 5, which is illustrated on Fig. 6.7.

Let G_{i-1} be the collection of conformers selected up to stage $i-1$, and denote C_{s_i} the ith conformer selected. Also denote R_i the candidate conformers remaining once the ith conformer has been selected. In order to select C_{s_i}, we maintain a priority queue Q such that the key associated to a conformer C_l is $k(C_l) = w(G_{i-1} \cup \{C_l\}) - w(G_{i-1})$.

Apart from the heap itself, we shall use the following data structures:
– GB: greedy selection boundary, i.e. patches found on $\partial \cup G_{i-1}$,
– $H_Q(P_k)$: candidate conformers covering patch P_k.

As the arrangement calculation provides us with a list $H(P_k)$ of balls covering a given patch P_k, the list of conformers $H_Q(P_k)$ covering P_k is easily set up.

We shall also assume a patch found on the boundary of a candidate conformer has a status with respect to to G_{i-1}: $status(P_k) = covered$ iff $P_k \subset \cup G_{i-1}$, and *exposed* otherwise. Upon selection of conformer C_{s_i}, two types of patches have to be taken care of:
▷ **Case 1:** patches covered by C_{s_i}, which are found either on $\partial \cup G_{i-1}$ (Case 1a), or patches found on the boundary of conformers from R_i (Case 1b).

Consider sub-case 1a, i.e. a P_k patch found on $\partial \cup G_{i-1}$ which is covered by C_{s_i}. If this patch is also covered by another conformer C_l in R_i, the weight of this conformer has to be updated as $k(C_l) \leftarrow k(C_l)+w(P_k)$. Indeed, conformers C_{s_i} and C_l were competing in the queue, and both had been subtracted $w(P_k)$ to compare the relative increments $k(C_{s_i})$ and $k(C_l)$. Now that C_{s_i} has been selected, and since patch P_k has already been accounted for in the weight of conformer C_{s_i}, the weight of conformer C_l has to be corrected as indicated.

Consider now sub-case 1b, i.e. a patch P_k found on the boundary of a candidate conformer. This patch being now covered by C_{s_i}, it will not contribute to an increment of the boundary of the union, so that conformer C_l has to be updated as $k(C_l) \leftarrow k(C_l) - w(P_k)$.
▷ **Case 2:** patches found on the boundary $\partial \cup G_i$ contributed by conformer C_{s_i}. In selecting the $i+1$th conformer, such patches may get covered by candidate conformers. The weight of each such conformer C_l thus has to be updated as $k(C_l) \leftarrow k(C_l) - w(P_k)$. Note in passing that the fact that several candidates may cover such a patch is responsible for the afore-described sub-case 1a.

To prove Thm. 9, we shall assume that the priority queue is implemented using a Fibonacci heap, while dictionaries are handles using hash tables. Under these assumptions:
Proof.(Thm. 9) We first note that each hash-set operation and `UpdateKey` operation individually takes $O(1)$ amortized time, whereas the `RemoveMin` operation takes $O(\log n)$ time –using Fibonacci heaps.

The outermost loop and hence the `RemoveMin` operation is repeated s times. We now look at the calls of Update_H_lists. This function is called at most once for each conformer. The first loop in the function runs at most once for each primitive. For each primitive, the two inner most lines are executed as many times as there are patches that are covered by the primitive. Summing over all possible primitives and conformers, the number of times the inner most lines are executed is clearly bounded by τ. Now, consider the loop that repeats for all patches P_k that are covered by the primitive S_j. The statements inside the if loop are executed if the patch was on the boundary of the union of previously selected conformers. If this is the case, the patch no longer remains on the boundary after the execution of this part. So the lines inside the *if part* are executed at most once for each patch. In the *if part*, there is a loop that repeats for every candidate conformer that covers the patch. Summing over all possible patches, these lines are executed at most τ times. The *else part* takes constant time in each run, and is repeated for every candidate conformer that contains the patch. Thus, the else part is also repeated at most τ times. The second loop for the boundary patches repeats at most once for each patch. The inner loop there, again repeats for each candidate that contains the patch, hence the number of executions of the innermost statement is again bounded by τ.

Thus, overall the execution of the algorithm is bounded by $O(\tau + s \log n)$. □

Algorithm 5 Greedy algorithm for surface decomposition.

$W_t \leftarrow 0$ /*Total weight returned*/
$G_0 \leftarrow \emptyset$/*Greedy Selection*/
$GB \leftarrow \emptyset$/*Greedy Selection Boundary*/
for $i = 1$ to s **do**
  `RemoveMin`: Pop C_{s_i} from queue
  $G_i = G_{i-1} \cup \{C_{s_i}\}$
  $Update\_H\_lists(C_{s_i})$
  for all primitives S_j of C_{s_i} **do**
    /*Case 1: patches covered by S_j*/
    for all patches $P_k \in K(S_j)$ /*covered by S_j*/ **do**
      /*Case 1a: patches on G_{i-1}*/
      if $P_k \in GB$ /*$P_k \in \partial \cup G_{i-1}$*/ **then**
        $GB \leftarrow GB \backslash \{P_k\}$
        for all $C_l \in H_Q(P_k)$ /*candidates covering P_k*/ **do**
          `UpdateKey`: $k(C_l) \leftarrow k(C_l) + w(P_k)$
        end for
        /*Case 1b: patches of conformers in R_i*/
      else if $status(P_k) = exposed$ /*$P_k \not\subset \cup G_{i-1}$*/ **then**
        Let C_l be the conformer patch P_k is on the boundary of
        `UpdateKey`: $k(C_l) \leftarrow k(C_l) - w(P_k)$
      end if
      $status(P_k) \leftarrow$ *covered*
    end for
    /*Case 2: patches on the boundary of C_{s_i}*/
    for all $P_k \in B(C_{s_i})$ /* boundary of C_{s_i}*/ **do**
      if $status(P_k) = exposed$ /*$P_k \not\subset \cup G_{i-1}$*/ **then**
        $GB \leftarrow GB \cup \{P_k\}$
        for all $C_l \in H_Q(P_k)$ /*candidates covering P_k*/ **do**
          `UpdateKey`: $k(C_l) \leftarrow k(C_l) - w(P_k)$
        end for
      end if
    end for
  end for
end for

Algorithm 6 Algorithm *Update_H_lists*(C_i)

```
for all primitives S_j of C_i do
  for all patches P_k ∈ K(S_j) /*covered by S_j*/ do
    if C_i ∈ H_Q(P_k) then
      remove C_i from H_Q(P_k)
    end if
  end for
end for
```

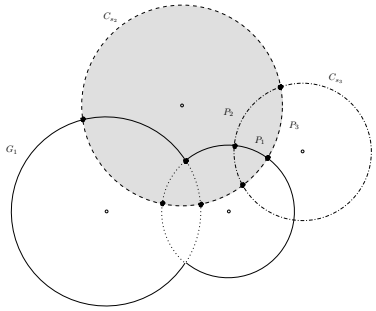


FIG. 6.7 – Greedy algorithm for surface weights. Patches triggering updates of keys upon selection of C_{s_2} are P_1, P_2, P_3.

6.7 Appendix: Material and Methods

6.7.1 `Direx` and `Loopy`

We selected two algorithms to generate loop conformers, which respectively yield dense and sparse ensembles of conformers. Algorithm `Direx` [172], based on algorithm CONCOORD [69], handles a whole protein and processes all the atoms in the same way. The method performs perturbations of the atomic positions while preserving constraints on internal coordinates (bond lengths and dihedral angles). The generation of n conformers is iterative, since the kth conformer is taken as starting point for the generation of the $k+1$th one. Applying this algorithm to a loop from a PDB structure yields a collection of conformers spanning a relatively small region around the original loop in the PDB file.

Algorithm `Loopy` [195] is a genetic algorithm that evolves a population of loops, the $k+1$th generation mixing a subset (survivors) of the kth generation together with new individuals derived from this subset. The main features of the algorithm are two-fold. First, the algorithm focuses on the backbone, onto which side-chains are added using a rotamer library. Second, the selection of the survivors uses a *colony* energy. This energy features a potential energy term, together with an entropy term encoding the spread of the neighborhood of a given conformation. This latter term accounts for the usual enthalpy-entropy competition, since a high internal energy conformation might be promoted thanks to a large entropy. This strategy naturally yields rather diverse sets of conformations.

PDBCODE	MSA `Direx`	MSA `Loopy`	Nb balls	Nb res.	RMSD
1BTH-atomic	1906.01	3423.34	108	12	5.7Å
1BTH-coarse	1639.72	3142.11	29	12	
1CGI-atomic	1867.17	3516.97	103	13	unres.
1CGI-coarse	1583.7	3032.78	28	13	
1OAZ-atomic	2535.6	4788.63	142	14	2.1Å
1OAZ-coarse	2262.51	4211.13	37	14	
3HHR-atomic	3835.2	15976.8	223	28	5.5Å
3HHR-coarse	3549.63	16345.7	67	28	

TAB. 6.2 – `Direx` versus `Loopy`: MSA for $n = 500$ conformers. The number of residues takes into account the two residues bounding all the conformers—these are common to all conformers.

6.7.2 Geometric and Topological Assessment: Tables

PDB	$\frac{R_G}{R_H}$	$\frac{n_{H_{100\%}}}{n_G}$	$\frac{n_{H_{95\%}}}{n_G}$
1BTH-atomic	1.11	112.67	6.0
1CGI-atomic	1.12	154.33	1.67
1OAZ-atomic	1.13	1.67	1.33
3HHR-atomic	1.09	9.67	2.67
1BTH-coarse	1.14	67.2	3.6
1CGI-coarse	1.12	160.0	2.33
1OAZ-coarse	1.15	1.67	1.33
3HHR-coarse	1.11	10.67	2.67

TAB. 6.3 – `Direx` data set. Comparison of the selection methods. See section 6.3.2 for notation.

PDB	$\frac{R_G}{R_H}$	$\frac{n_{H_{100\%}}}{n_G}$	$\frac{n_{H_{95\%}}}{n_G}$
1BTH-atomic	1.18	9.5	7.0
1CGI-atomic	1.14	79.5	28.17
1OAZ-atomic	1.21	42.64	10.82
3HHR-atomic	1.54	4.3	3.8
1BTH-coarse	1.25	10.86	8.29
1CGI-coarse	1.34	79.5	11.0
1OAZ-coarse	1.27	44.67	10.89
3HHR-coarse	1.56	13.31	3.0

TAB. 6.4 – `Loopy` data set. Comparison of the selection methods. See section 6.3.2 for notation.

PDB	$n_{G_{100\%}}$	$m(\beta_1)$	$M(\beta_1)$	$\mu(\beta_1)$	$med(\beta_1)$	$m(\beta_1)$	$M(\beta_1)$	$\mu(\beta_1)$	$med(\beta_1)$
1BTH-atomic	28	7	16	9.38	8	0	7	0.44	0
1CGI-atomic	10	2	12	8.06	8	0	8	0.45	0
1OAZ-atomic	13	5	21	7.07	6	0	19	0.44	0
3HHR-atomic	6	6	30	13.95	11.5	0	31	1.05	0
1BTH-coarse	17	0	12	9.63	11	0	11	0.96	1
1CGI-coarse	22	0	17	6.49	7	0	8	1.14	1
1OAZ-coarse	14	1	18	9.79	10	0	14	1.53	2
3HHR-coarse	11	4	36	16.16	15	1	23	2.22	1

TAB. 6.5 – `Direx` data set. Comparing the evolution of the first Betti number up to $n_{G_{100\%}}$ conformers selected; Left: `Greedy`; Right: `HClust`. m, M, μ and med respectively stand for $\min, \max, mean, median$.

PDB	$n_{G_{100\%}}$	$m(\beta_1)$	$M(\beta_1)$	$\mu(\beta_1)$	$med(\beta_1)$	$m(\beta_1)$	$M(\beta_1)$	$\mu(\beta_1)$	$med(\beta_1)$
1BTH-atomic	39	3	44	25.08	24	1	20	3.78	2
1CGI-atomic	16	4	38	23.68	22	1	27	2.6	2
1OAZ-atomic	37	4	43	25.82	24	4	18	7.58	5
3HHR-atomic	36	17	342	283.12	306	26	163	86.56	74
1BTH-coarse	32	0	51	36.26	35	2	25	6.73	6
1CGI-coarse	21	0	59	30.15	26	0	30	5.67	5
1OAZ-coarse	28	0	77	61.67	63	7	35	19.99	19
3HHR-coarse	46	1	492	382.59	452	6	219	149.02	141

TAB. 6.6 – `Loopy` data set. Comparing the evolution of the first Betti number up to $n_{G_{100\%}}$ conformers selected; Left: `Greedy`; Right: `HClust`. m, M, μ and med respectively stand for $\min, \max, mean, median$.

File	Loop generated by	Selected by	Selection size	Min.	Max.	δ	$\frac{\delta_G - \delta_H}{\delta_H}$
1BTH	`Loopy`	Greedy	10	2.767	10.15	7.383	0.392
1BTH	`Loopy`	HClust	10	2.693	7.996	5.303	
1BTH	`Loopy`	Greedy	15	2.767	10.15	7.383	0.315
1BTH	`Loopy`	HClust	15	2.693	8.307	5.614	
1BTH	`Loopy`	Greedy	30	2.581	10.15	7.569	-0.052
1BTH	`Loopy`	HClust	30	2.162	10.15	7.988	
1BTH	`Direx`	Greedy	10	0.5627	3.501	2.9383	0.184
1BTH	`Direx`	HClust	10	0.9152	3.396	2.4808	
1BTH	`Direx`	Greedy	15	0.5627	3.501	2.9383	0.184
1BTH	`Direx`	HClust	15	0.9152	3.396	2.4808	
1BTH	`Direx`	Greedy	30	0.5627	3.679	3.1163	0.203
1BTH	`Direx`	HClust	30	0.8562	3.446	2.5898	
1CGI	`Loopy`	Greedy	10	3.614	10.4	6.786	0.27
1CGI	`Loopy`	HClust	10	3.413	8.755	5.342	
1CGI	`Loopy`	Greedy	15	3.614	10.91	7.296	0.326
1CGI	`Loopy`	HClust	15	3.413	8.916	5.503	
1CGI	`Loopy`	Greedy	30	3.614	10.95	7.336	-0.078
1CGI	`Loopy`	HClust	30	2.441	10.4	7.959	
1CGI	`Direx`	Greedy	10	0.7777	4.442	3.6643	0.637
1CGI	`Direx`	HClust	10	1.364	3.602	2.238	
1CGI	`Direx`	Greedy	15	0.7777	4.442	3.6643	0.336
1CGI	`Direx`	HClust	15	1.364	4.106	2.742	
1CGI	`Direx`	Greedy	30	0.7777	4.442	3.6643	0.336
1CGI	`Direx`	HClust	30	1.364	4.106	2.742	
1OAZ	`Loopy`	Greedy	10	4.237	17.32	13.083	0.58
1OAZ	`Loopy`	HClust	10	3.327	11.61	8.283	
1OAZ	`Loopy`	Greedy	15	4.237	17.35	13.113	0.392
1OAZ	`Loopy`	HClust	15	3.079	12.5	9.421	
1OAZ	`Loopy`	Greedy	30	2.716	17.35	14.634	0.68
1OAZ	`Loopy`	HClust	30	3.079	11.79	8.711	
1OAZ	`Direx`	Greedy	10	0.8034	6.967	6.1636	0.353
1OAZ	`Direx`	HClust	10	1.933	6.49	4.557	
1OAZ	`Direx`	Greedy	15	0.8034	6.967	6.1636	0.219
1OAZ	`Direx`	HClust	15	1.111	6.169	5.058	
1OAZ	`Direx`	Greedy	30	0.8034	6.967	6.1636	0.221
1OAZ	`Direx`	HClust	30	1.111	6.159	5.048	
3HHR	`Loopy`	Greedy	10	8.517	21.16	12.643	-0.288
3HHR	`Loopy`	HClust	10	5.521	23.27	17.749	
3HHR	`Loopy`	Greedy	15	8.517	23.27	14.753	-0.169
3HHR	`Loopy`	HClust	15	5.521	23.27	17.749	
3HHR	`Loopy`	Greedy	30	8.517	23.27	14.753	-0.169
3HHR	`Loopy`	HClust	30	5.521	23.27	17.749	
3HHR	`Direx`	Greedy	10	0.4793	5.389	4.9097	0.092
3HHR	`Direx`	HClust	10	0.8399	5.335	4.4951	
3HHR	`Direx`	Greedy	15	0.4793	5.485	5.0057	0.114
3HHR	`Direx`	HClust	15	0.8399	5.335	4.4951	
3HHR	`Direx`	Greedy	30	0.4793	5.485	5.0057	0.111
3HHR	`Direx`	HClust	30	0.8399	5.346	4.5061	

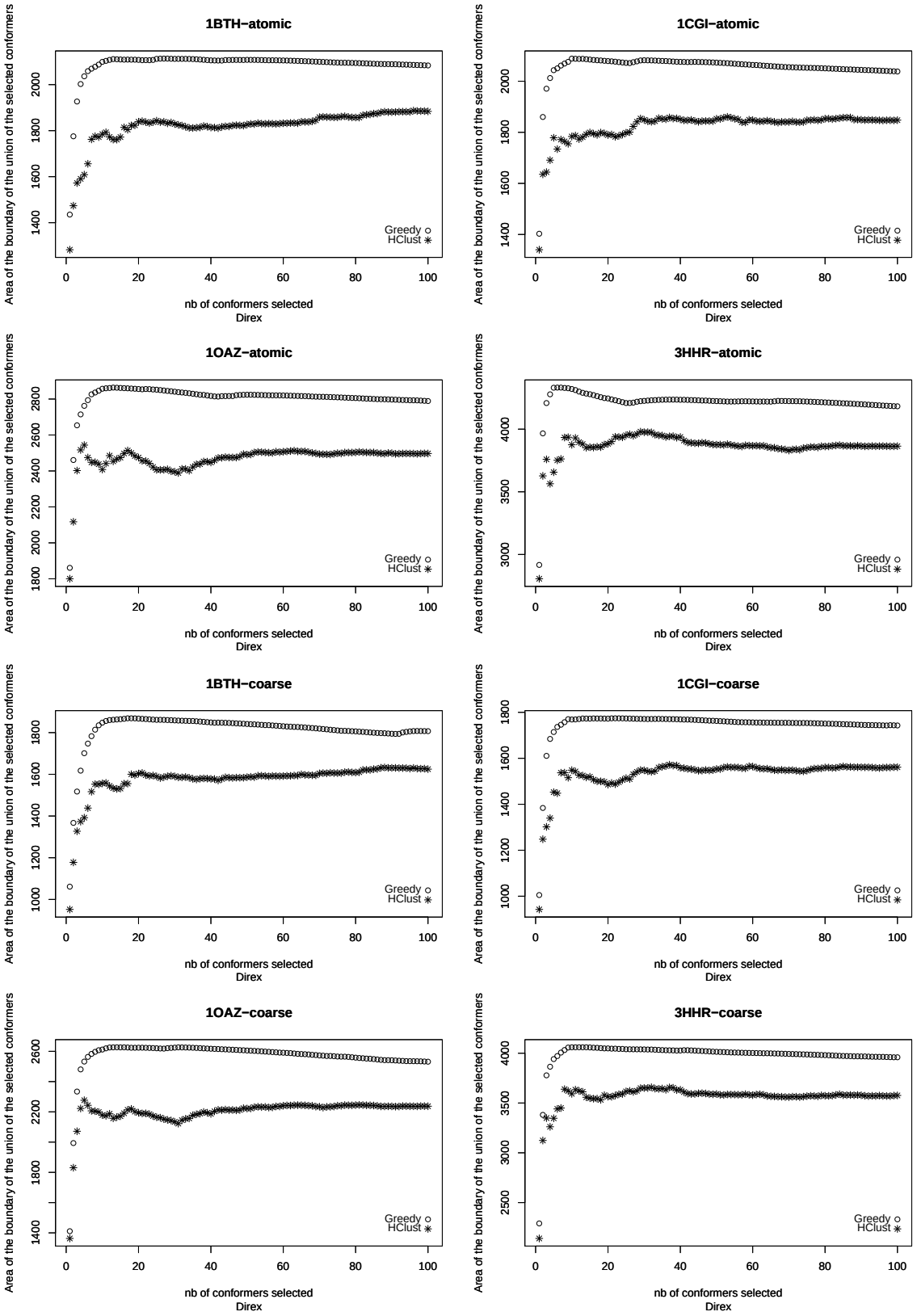
TAB. 6.7 – Selection by `HClust` versus selection by `Greedy` for coarse models: C_α-RMSD of loops selected with respect to the native bound loop; Min. and Max. stand for the minimum and the maximum of these values. $\delta = Max - Min$, the subscript G or H standing for the algorithm used, `Greedy` or `HClust`.

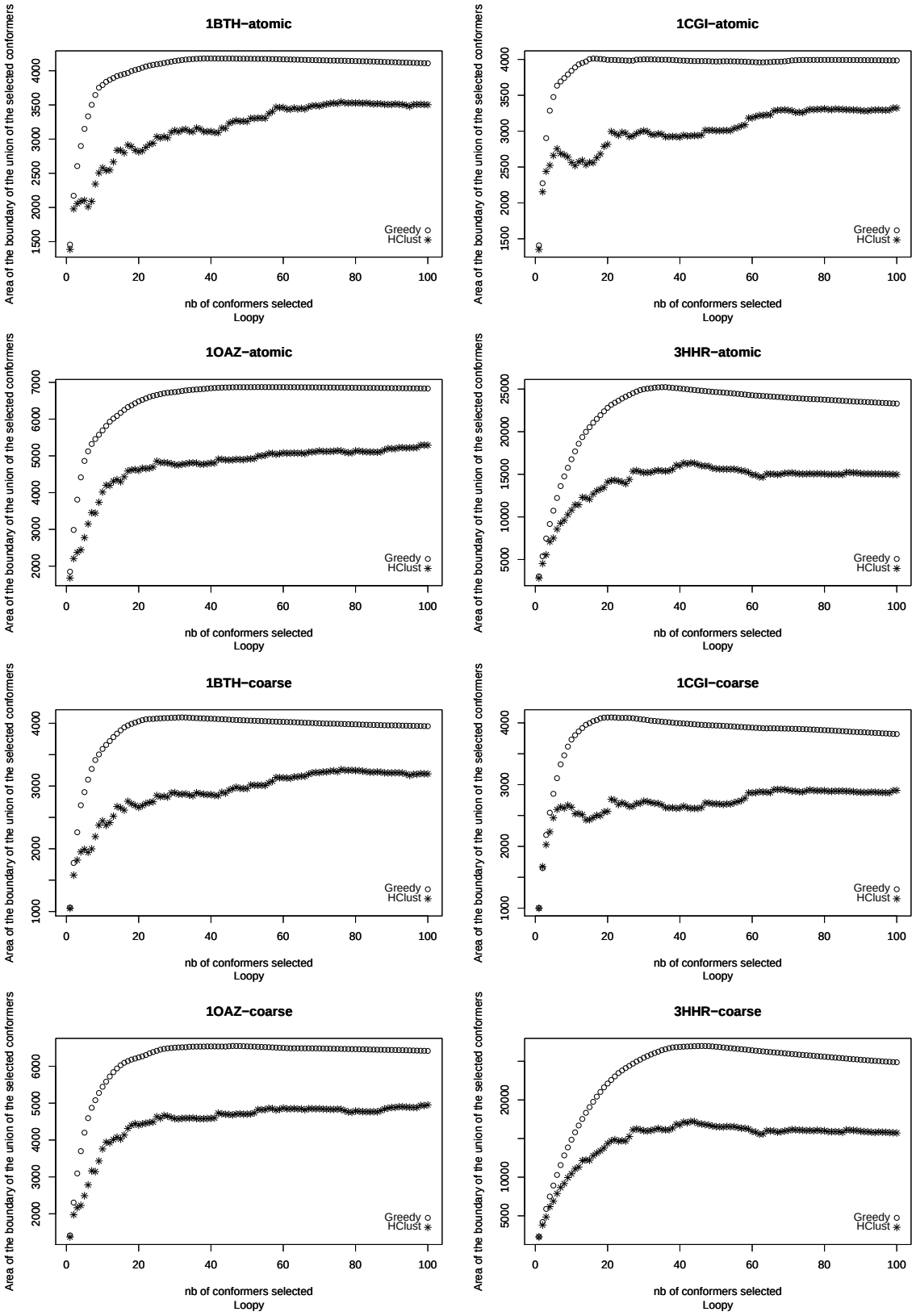
File	Loop generated by	Selected by	Selection size	Min.	Max.	δ	$\frac{\delta_G - \delta_H}{\delta_H}$
1BTH	Loopy	Greedy	10	2.693	10.15	7.457	0.406
1BTH	Loopy	HClust	10	2.693	7.996	5.303	
1BTH	Loopy	Greedy	15	2.581	10.15	7.569	0.348
1BTH	Loopy	HClust	15	2.693	8.307	5.614	
1BTH	Loopy	Greedy	30	2.581	10.15	7.569	-0.052
1BTH	Loopy	HClust	30	2.162	10.15	7.988	
1BTH	Direx	Greedy	10	0.9006	3.528	2.6274	0.059
1BTH	Direx	HClust	10	0.9152	3.396	2.4808	
1BTH	Direx	Greedy	15	0.9006	3.679	2.7784	0.12
1BTH	Direx	HClust	15	0.9152	3.396	2.4808	
1BTH	Direx	Greedy	30	0.9006	3.679	2.7784	0.073
1BTH	Direx	HClust	30	0.8562	3.446	2.5898	
1CGI	Loopy	Greedy	10	5.934	10.55	4.616	-0.136
1CGI	Loopy	HClust	10	3.413	8.755	5.342	
1CGI	Loopy	Greedy	15	3.614	10.91	7.296	0.326
1CGI	Loopy	HClust	15	3.413	8.916	5.503	
1CGI	Loopy	Greedy	30	3.614	10.95	7.336	-0.078
1CGI	Loopy	HClust	30	2.441	10.4	7.959	
1CGI	Direx	Greedy	10	1.45	4.442	2.992	0.337
1CGI	Direx	HClust	10	1.364	3.602	2.238	
1CGI	Direx	Greedy	15	1.45	4.442	2.992	0.091
1CGI	Direx	HClust	15	1.364	4.106	2.742	
1CGI	Direx	Greedy	30	1.437	4.442	3.005	0.096
1CGI	Direx	HClust	30	1.364	4.106	2.742	
1OAZ	Loopy	Greedy	10	2.716	17.35	14.634	0.767
1OAZ	Loopy	HClust	10	3.327	11.61	8.283	
1OAZ	Loopy	Greedy	15	2.716	17.35	14.634	0.553
1OAZ	Loopy	HClust	15	3.079	12.5	9.421	
1OAZ	Loopy	Greedy	30	2.716	17.35	14.634	0.68
1OAZ	Loopy	HClust	30	3.079	11.79	8.711	
1OAZ	Direx	Greedy	10	0.8034	6.71	5.9066	0.296
1OAZ	Direx	HClust	10	1.933	6.49	4.557	
1OAZ	Direx	Greedy	15	0.8034	6.71	5.9066	0.168
1OAZ	Direx	HClust	15	1.111	6.169	5.058	
1OAZ	Direx	Greedy	30	0.8034	6.71	5.9066	0.17
1OAZ	Direx	HClust	30	1.111	6.159	5.048	
3HHR	Loopy	Greedy	10	11.11	23.27	12.16	-0.315
3HHR	Loopy	HClust	10	5.521	23.27	17.749	
3HHR	Loopy	Greedy	15	11.11	23.27	12.16	-0.315
3HHR	Loopy	HClust	15	5.521	23.27	17.749	
3HHR	Loopy	Greedy	30	8.611	23.27	14.659	-0.174
3HHR	Loopy	HClust	30	5.521	23.27	17.749	
3HHR	Direx	Greedy	10	1.37	5.344	3.974	-0.116
3HHR	Direx	HClust	10	0.8399	5.335	4.4951	
3HHR	Direx	Greedy	15	1.37	5.344	3.974	-0.116
3HHR	Direx	HClust	15	0.8399	5.335	4.4951	
3HHR	Direx	Greedy	30	1.37	5.344	3.974	-0.118
3HHR	Direx	HClust	30	0.8399	5.346	4.5061	

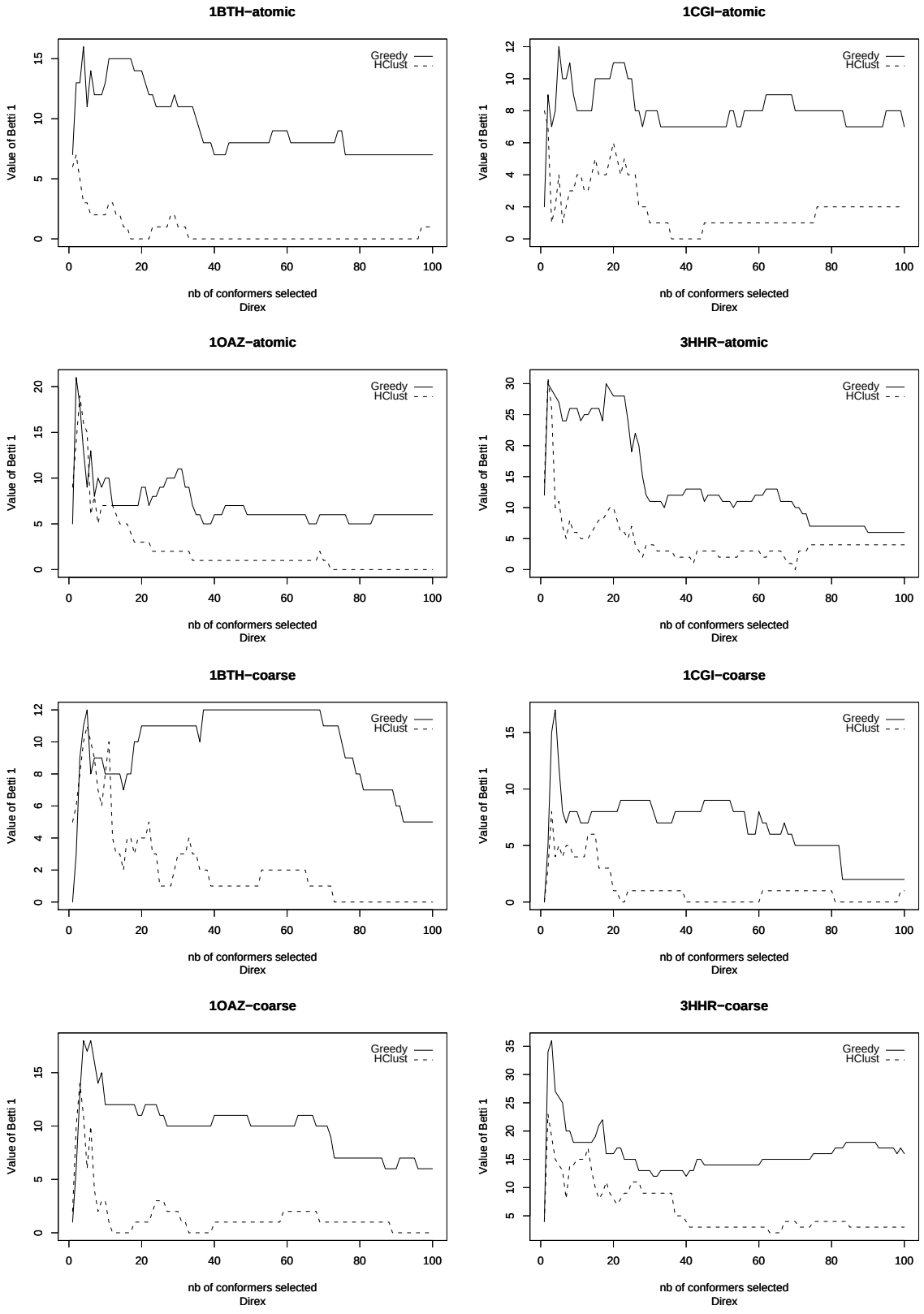
TAB. 6.8 – Selection by `HClust` versus selection by `Greedy` for atomic models: C_α-RMSD of loops selected with respect to the native bound loop; Min. and Max. stand for the minimum and the maximum of these values. $\delta = Max - Min$, the subscript G or H standing for the algorithm used, `Greedy` or `HClust`.

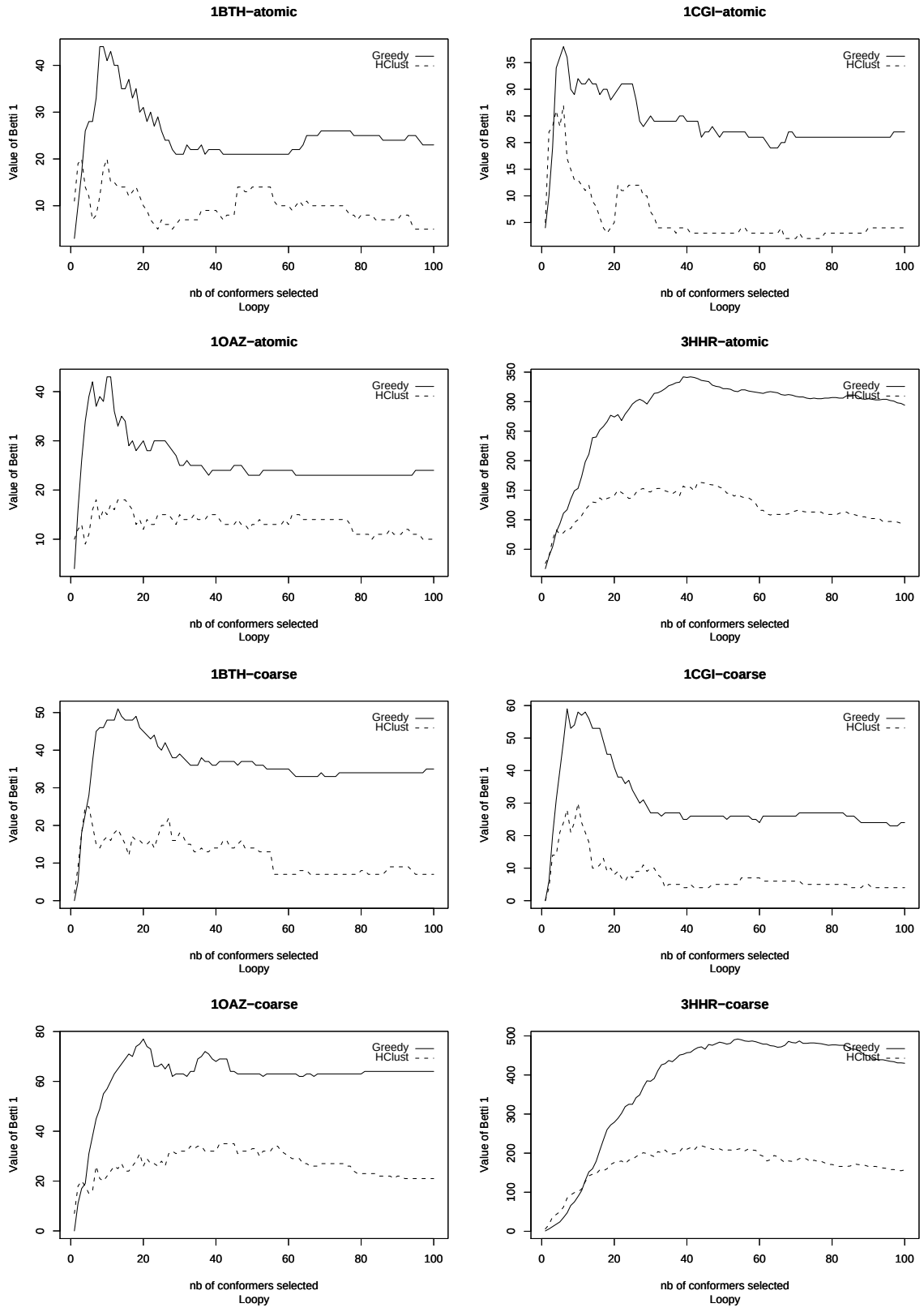
6.7.3 Geometric and Topological Assessment: Graphs

This section presents the plots of the variation of the MSA and of the Betti number β_1, as a function of the selection size. For each statistic, each model features four plots: {`Direx`, `Loopy`} $\times$ {atomic resolution, coarse grain}. Results are discussed in section 6.3.2.


1BTH−atomic
1CGI−atomic
1OAZ−atomic
3HHR−atomic
1BTH−coarse
1CGI−coarse
1OAZ−coarse
3HHR−coarse
Area of the boundary of the union of the selected conformers
nb of conformers selected
Direx
Greedy
HClust


1BTH−atomic
1CGI−atomic
1OAZ−atomic
3HHR−atomic
1BTH−coarse
1CGI−coarse
1OAZ−coarse
3HHR−coarse
Area of the boundary of the union of the selected conformers
nb of conformers selected
Loopy
Greedy
HClust


1BTH−atomic
1CGI−atomic
1OAZ−atomic
3HHR−atomic
1BTH−coarse
1CGI−coarse
1OAZ−coarse
3HHR−coarse
Greedy
HClust
Value of Betti 1
nb of conformers selected
Direx


1BTH−atomic
1CGI−atomic
1OAZ−atomic
3HHR−atomic
1BTH−coarse
1CGI−coarse
1OAZ−coarse
3HHR−coarse
Greedy
HClust
Value of Betti 1
nb of conformers selected
Loopy

6.7.4 Graphs: Docking Assessment

For a given system, the 6 plots are organized as follows: the first two correspond to the sanity check B/B/B-1 and U/B/B-1; the next (respectively last) two, namely B/B/B-`HClust`-10 and B/B/B-`Greedy`-10 (respectively U/B/B-`HClust`-10, U/B/B-`Greedy`-10) allow the comparison of `Greedy` and `HClust` for the Bound (respectively Unbound) version of the receptor. Recall that flexible regions on `1BTH`, `1CGI` and `1OAZ` feature 10, 11 and 12 amino acids respectively on their receptor. These graphs are discussed in section 6.4.3.

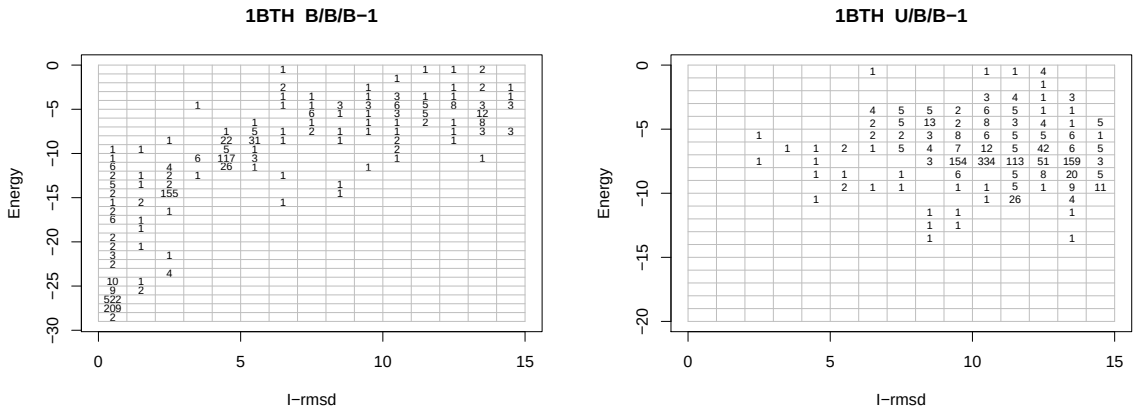


FIG. 6.8 – 1BTH: binning the docking tests using the Bound and Unbound forms of the receptor . See text for details.

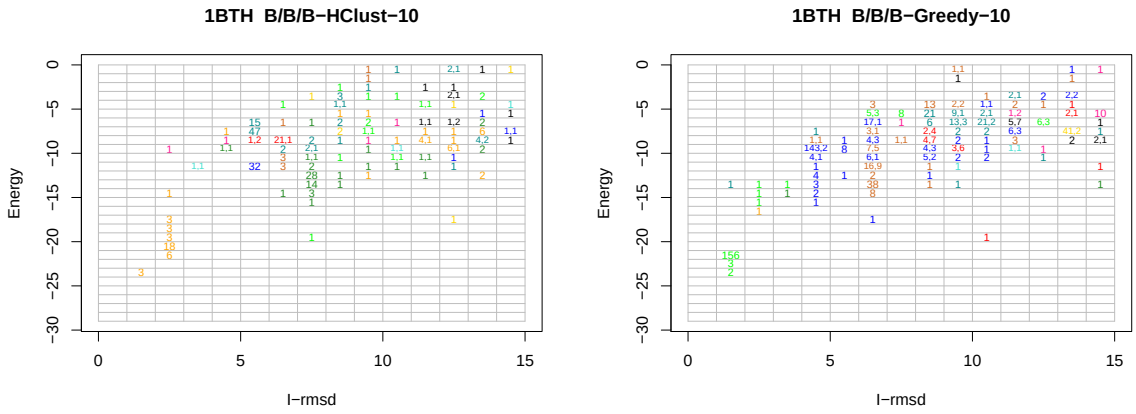


FIG. 6.9 – 1BTH: binning the docking tests using the Bound form of the receptor. See text for details.

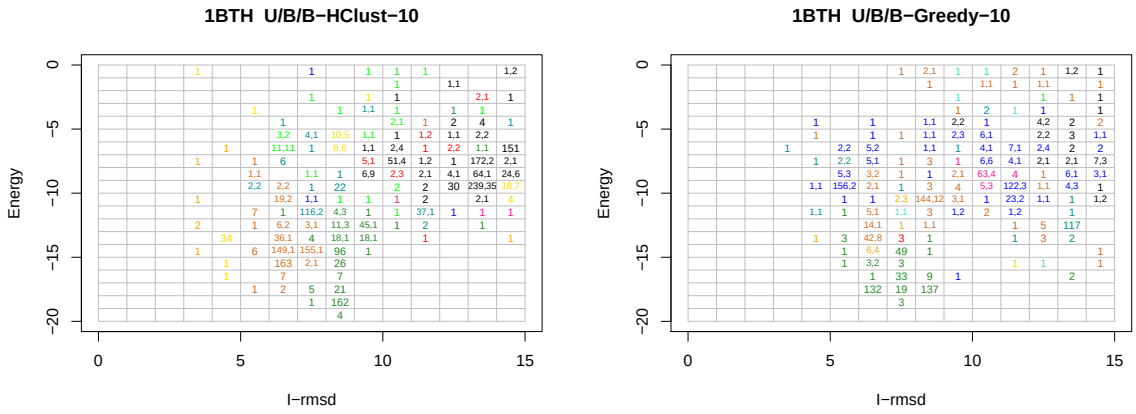


FIG. 6.10 – 1BTH: binning the docking tests using the Unbound form of the receptor. See text for details.

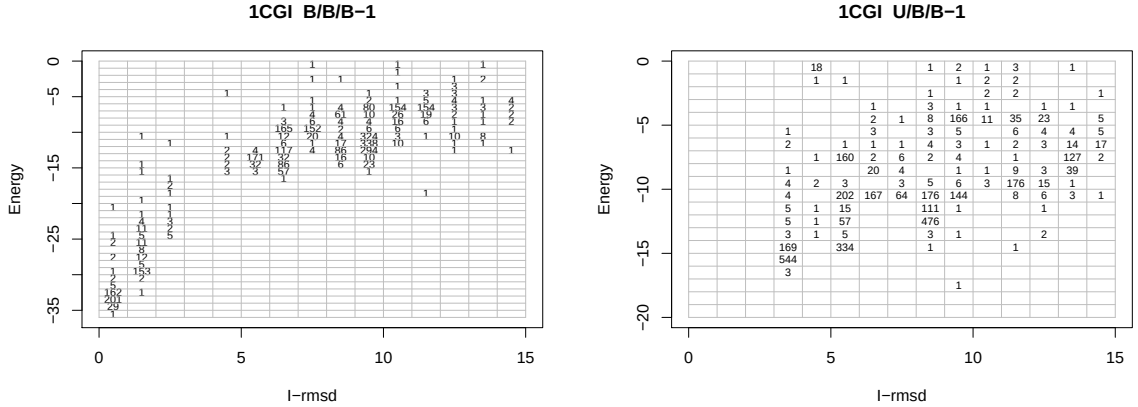


FIG. 6.11 – 1CGI: binning the docking tests using the Bound and Unbound forms of the receptor . See text for details.

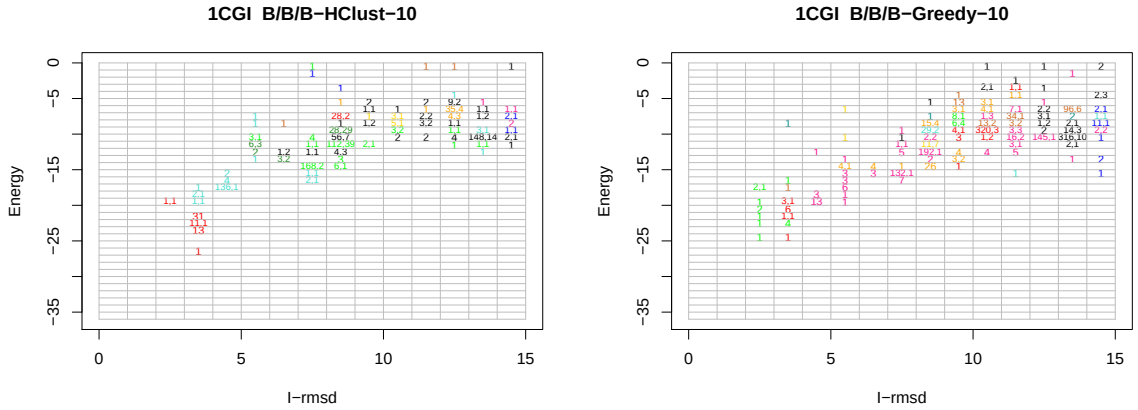


FIG. 6.12 – 1CGI: binning the docking tests using the Bound form of the receptor. See text for details.

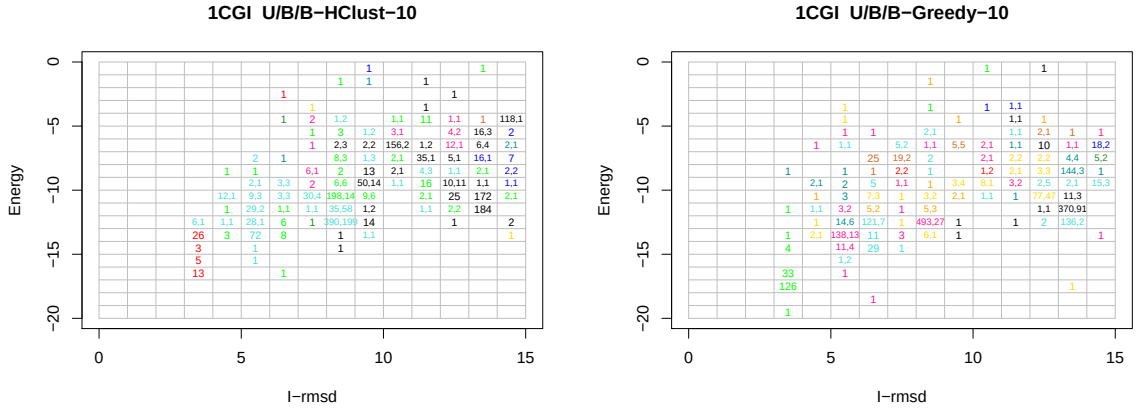


FIG. 6.13 – 1CGI: binning the docking tests using the Unbound form of the receptor. See text for details.

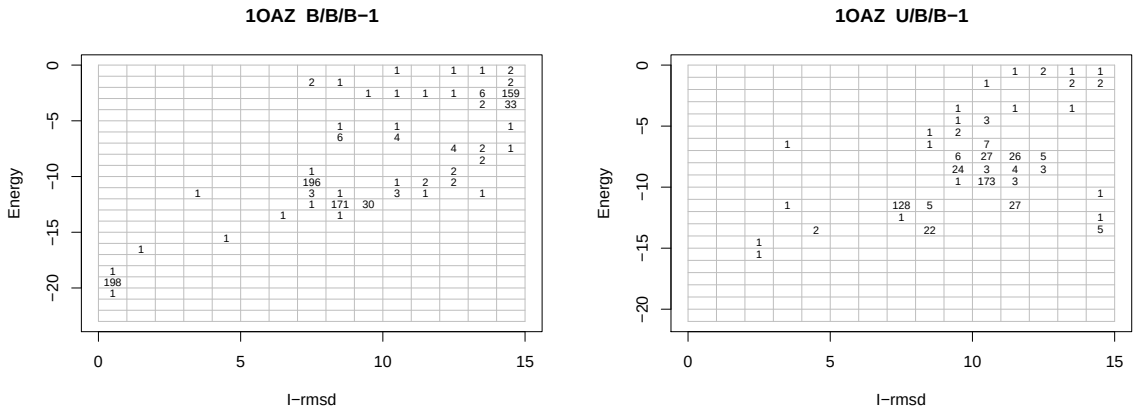


FIG. 6.14 – 1OAZ: binning the docking tests using the Bound and Unbound forms of the receptor . See text for details.

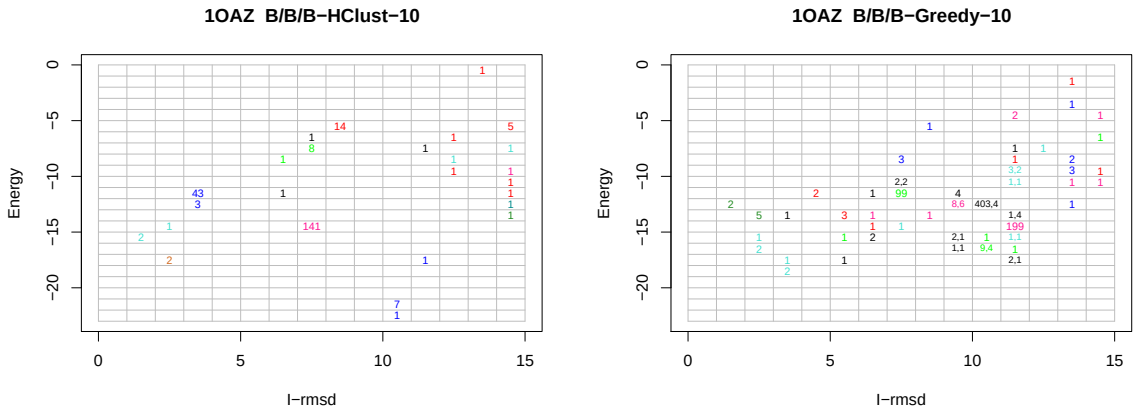


FIG. 6.15 – 1OAZ: binning the docking tests using the Bound form of the receptor. See text for details.

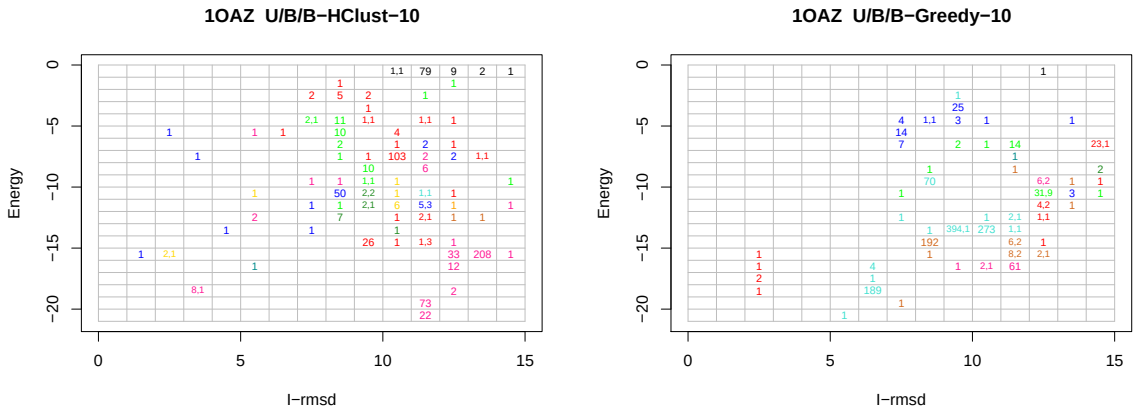


FIG. 6.16 – 1OAZ: binning the docking tests using the Unbound form of the receptor. See text for details.

Chapitre 7

Implementation and Software Design

Chapter Overview

This chapter sketches the implementation of the algorithms presented in the previous chapters. Section 7.1, dedicated to the implementations of the 3D Spherical Kernel (SK) and the Algebraic Kernel (AK) of chapter 7, starts introducing some classical C++ generic programming tricks. Section 7.2 outlines the interface of the arrangement of circles on a sphere algorithm (which is presented in chapters 3 and 4), and presents one way to use it for computing all surface arrangements induced by a set a spheres. In both sections, an outline of the engineering decisions made is provided.

7.1 3D Spherical Kernel Implementation

In this section, we depict our implementation of the kernel concept developed in chapter 5. The implementation of the 3D Spherical Kernel and of the Algebraic Kernelare about $9,000$ and $2,500$ lines of C++ respectively. For a given class and to limit space requirements, the emphasis is on data members. To make the presentation self-contained, we first present C++ prerequisites.

7.1.1 C++ Prerequisites

Generalities. In C++, the keyword `typedef` allows to create a shortcut for a given type name. Inside a template class, types provided by the template parameter are accessed using the `typename` keyword. In the same way, inside a template class, while instantiating a nested template type which depends on a template parameter of the class, the keyword `template` is used to disambiguate the syntax. Finally, as a general remark, all classes of the 3D Spherical Kernel are instantiated with the 3D Spherical Kernel itself, so as to access all features provided by the 3D Spherical Kernel.

Reference counting. Reference counting is a classical strategy used to speed up copies of complicated objects. The strategy is simple: the object allocated is accompanied by an integer counting the number of references to that object. The value of this integer is increased when a *constructed copy* of the object is created and decreased when such a copy is deleted. When the value of the integer vanishes, the object is de-allocated. This mechanism is implemented in CGAL using the template class `CGAL::Handle_for<T>`. Reference counting is an option for CGAL kernels, though a mechanism provided by the template class `SK::Handle<T>`. This template class provides the `SK::Handle<T>::type` type, which is either the template parameter `T` itself (not using reference counting), or the `CGAL::Handle_for<T>` type (using reference counting). To accommodate both options, the `CGAL::get` function always returns the object of type `T`.

Algebraic numbers. The 3D Spherical Kernel relies on two number types: a field number type (see section 5.2.1), from the linear kernel (`Linear_kernel::FT`), and an algebraic number of degree two type from the algebraic kernel (`AK::Root_of_2`). This former type (`double` for example) may come with a built-in

square root function. If not, the square root is provided by the template class `CGAL::Root_of_2<FT>`, whose underlying representation is $\alpha + \beta\sqrt{\gamma}$ where α, β and γ are of the template type `FT`. To accommodate both options, the `AK::Root_of_2` is retrieved from the template class `CGAL::Root_of_traits<FT>`, namely as `CGAL::Root_of_traits<Linear_kernel::FT>::Root_of_2`.

7.1.2 3D Spherical Kernel

Outline The member variables of each object type are gathered into a tuple to ease reference counting: one pointer is manipulated independently of the number of member variables. For each object, we distinguish two cases: the general setting, and the particular case where the object lies on a reference sphere which may also accommodates other objects. This latter cases indeed requires additional predicates and constructions, for example for the comparison of cylindrical coordinates of points located on that sphere. Implementation-wise, C++ inheritance is used, so as to make the members of the base class accessible in the derived class.

▷ A circle on a sphere is defined as the intersection of a sphere and a plane. The corresponding class thus stores these two primitives. The class corresponding to a circle on a reference sphere inherits from the class for circles on a sphere, so that the reference sphere is retrieved from the data member of the base class.

```
template <class SK>
class Circle_3{
  typedef std::pair<typename SK::Sphere_3,typename SK::Plane_3> Rep;
  typename SK::template Handle<Rep>::type base;
};

template <class SK>
class Circle_on_reference_sphere_3
    :public Circle_3<SK>{
  public:
  const typename SK::Sphere_3& reference_sphere() const
    {return CGAL::get(this->base).first;}
};
```

▷ The class used to represent the θ coordinate of a point on a reference sphere stores one algebraic number of degree two for the exact value of $\tan\theta$ or $\cot\theta$, together with an index specifying whether tan or cot is used —see section 5.6.2. The index is of type `SK::Hq_indices`, an enumerated type.

```
template <class SK>
class Theta_rep{
  typedef std::pair<typename SK::Hq_indices,typename SK::AK::Root_of_2> Rep;
  typename SK::template Handle<Rep>::type base;
};
```

▷ The class representing a point on a sphere is fully specified by the type of its coordinates, this type being provided by the algebraic kernel. The class corresponding to points on a reference sphere inherits from the class for points on a sphere, and has two additional members: the reference sphere, and the exact representation of the θ coordinate of the point.

```
template <class SK>
class Circular_arc_point_3{
  typedef typename AK::Root_for_spheres_2_3 Rep;
  typename SK::template Handle<Rep>::type base;
};

template <class SK>
class Circular_arc_point_on_reference_sphere_3
    :public Circular_arc_point_3<SK>{
  typedef std::pair<typename SK::Sphere_3,typename SK::Theta_rep> Expanded_rep;
  typename SK::template Handle<Expanded_rep>::type expanded_base;
  public:
  const typename SK::Sphere_3& reference_sphere() const
    {return CGAL::get(expanded_base).first;}
  const typename SK::Theta_rep& theta_rep() const
    {return CGAL::get(expanded_base).second;}
};
```

▷ The circular arc class stores the circle supporting the arc and the two endpoints. The class for circular arc on a reference sphere inherits from this class, but the types of the circle and of the endpoints differ. In the second and third template parameters of the base class, which are not the default ones. We recall that a circular arc is unambiguously defined as the set of points lying on the circle, counterclockwise from its source to its target in the positive plane containing the circle—a plane is positive if its equation is of the form $ax + by + cz + d = 0$ with $(a, b, c) > (0, 0, 0)$ according to the lexicographic order.

```
template <class SK,class Circle=typename SK::Circle_3,
          class Endpoint=typename SK::Circular_arc_point_3>
class Circular_arc_3{
  typedef CGAL::Triple<Circle,Endpoint,Endpoint> Rep;
  typename SK::template Handle<Rep>::type base;
};

template <class SK>
class Circular_arc_on_reference_sphere_3
    :public Circular_arc_3<SK,typename SK::Circle_on_reference_sphere_3,
      typename SK::Circular_arc_point_on_reference_sphere_3>{
  typename SK::template Handle<typename SK::Sphere_3>::type ref_sphere;
  public:
  const typename SK::Sphere_3& reference_sphere() const
    {return CGAL::get(ref_sphere);}
};
```

▷ The class for segments is defined by a triple: a line and two endpoints.

```
template <class SK>
class Line_arc_3{
  typedef CGAL::Triple<typename SK::Line_3,typename SK::Circular_arc_point_3,
                       typename SK::Circular_arc_point_3> Rep;
  typename SK::template Handle<Rep>::type base;
};
```

7.1.3 Algebraic Kernel

▷ The class representing equations of planes of the form $ax + by + cz + d = 0$, stores the 4 coefficients in an array. The template parameter determines the number type used for the coefficients.

```
template <class FT>
class Polynomial_1_3{
  FT rep[4]; // stores a, b, c, d
};
```

▷ The class representing equations of spheres of the form $(x - a)^2 + (y - b)^2 + (z - c)^2 - R^2 = 0$, stores the 4 parameters in an array. The template parameter determines the number type used for the parameters.

```
template <class FT>
class Polynomial_for_spheres_2_3{
  FT rep[4]; // stores a, b, c, R^2
};
```

▷ The algebraic class representing a point on a sphere stores one algebraic number of degree two per Cartesian coordinate. The template parameter determines the number type on which are built the algebraic numbers.

```
template <class FT>
class Root_for_spheres_2_3 {
  typename Root_of_traits<FT>::Root_of_2 x_;
  typename Root_of_traits<FT>::Root_of_2 y_;
  typename Root_of_traits<FT>::Root_of_2 z_;
};
```

7.1.4 An Example

The following example piece of code checks whether three spheres intersect in two points.

```
#include <CGAL/Cartesian.h>
#include <CGAL/Algebraic_kernel_for_spheres_2_3.h>
#include <CGAL/Spherical_kernel_3.h>
#include <CGAL/MP_Float.h>
#include <CGAL/Quotient.h>

typedef CGAL::Quotient< CGAL::MP_Float>               NT;
typedef CGAL::Cartesian<NT>                           Linear_k;
typedef CGAL::Algebraic_kernel_for_spheres_2_3<NT>    Algebraic_k;
typedef CGAL::Spherical_kernel_3<Linear_k,Algebraic_k> SK;
```

```
int main(){
  //construction of 3 spheres from their centers and squared radii
  SK::Sphere_3 s1(SK::Point_3(0,0,0),2);
  SK::Sphere_3 s2(SK::Point_3(0,1,0),1);
  SK::Sphere_3 s3(SK::Point_3(1,0,0),3);

  SK::Intersect_3 inter;
  SK::Compare_xyz_3 cmp;
  std::vector< CGAL::Object > intersections;
  inter(s1,s2,s3,std::back_inserter(intersections));

  std::pair<SK::Circular_arc_point_3,unsigned> p1,p2;
  //unsigned integer indicates multiplicity of intersection point
  if (intersections.size() >1){
    //as intersection can return several types (points with multiplicity,
    //circle,...), CGAL::Object and CGAL::assign are used to recover
    //the expected type
    if (CGAL::assign(p1,intersections[0]) && CGAL::assign(p2,intersections[1])){
      std::cout << "Two different intersection points" << std::endl;
      //intersection points are sorted lexicographically
      CGAL_assertion(cmp(p1.first,p2.first)==CGAL::SMALLER);
    }
    else
      std::cout << "Error" << std::endl;
  }
  return 0;
}
```

7.2 Implementation of the Arrangement of Circles on a Sphere

Outline The code presented here can be decomposed into three parts. The first part is a container storing objects *within* a 3D grid allowing iterations on objects within a given cell, and iterations on cells adjacent to a given one. The key in the design of this grid is to delegate non-generic operations to the traits class. The only constraint is on the connectivity of the cells (that of a cubic lattice). For example, if the user provides the correct traits class, he can define cells of different size. We decided to use a map to store the cells to avoid manipulation and storage of empty cells, with relatively fast access to any cell. The second part is the implementation of the BO algorithm on a sphere described in chapters 3 and 4. There are three levels in this implementation. The first level is the *advanced* level (class `Sweep_arc_sphere`). It provides low level functions for the sweep algorithm. In particular, one can for example stop the sweep at any theta value. The second level is the *user* level (class `Arrangement_on_a_sphere`). It provides high level functions; the complete sweep is done by one function and other functions are provided to recover information from the output. The third level (class `Argt_DS`) is the storage level. All the data structures storing the output of the sweep algorithm are gathered in this class (covering tree, HDS, *dots*). This is a convenient way to share the output between several classes. The last part is devoted to the computation of all the arrangements of circles on the surface of a set of spheres. It makes use of all the previously described classes. As summarized in section 5.5, given S_0 from a set of spheres, the grid class reports all the spheres from the set that intersect S_0. The 3D Spherical Kernel and the Algebraic Kernel provided as template parameters, supply the predicates, constructions and underlying number type to the class computing the arrangement of intersection circles on S_0. The area of each face is then computed using an implementation of the local Gauss-Bonnet theorem. The spheres covering each face of the arrangement are recovered using the covering tree. In particular, we show how to set up a filter at the level of the

arrangement to speed up computation, preserving the exactness and the robustness.

The implementation presented here makes extensive use of the Boost C++ library [32].

Area of a spherical cap Consider a sphere S_0 (of radius r_0) and a face F defined in an arrangement of circles on that sphere. In the following we give a formula for computing the area A, of the simply connected face F. Using notation introduced in section 4.2.1, the area of a face featuring several CCB (i.e. the face is defined by several closed cycles of halfedges) is obtained as the area of the region defined by the principal CCB, from which the area of each region defined by the remaining CCB after reversing their orientation, is subtracted.

Let h_i, $i = 0, \ldots, n-1$ be the n oriented circular arcs of the CCB defining F. For $i \in \{0, \ldots, n-1\}$, let v_i be the target vertex of h_i and θ_i be the external angle between h_i and $h_{i+1 \bmod n}$. Finally let s be a parametrization by arc length of the CCB defining F (with same orientation).

Using the local Gauss-Bonnet theorem [75], one can write:

$$\sum_{i=1}^{n} \int_{v_{i-1}}^{v_{i \bmod n}} k_g(s)ds + \frac{A}{r_0^2} + \sum_{i=0}^{n-1} \theta_i = 2\pi$$

where $k_g(s)$ is the geodesic curvature of the oriented circular arcs of the CCB defining F. Notice that this quantity is constant on a given oriented circular arc, and its value is 0 if the underlying circle is a great circle. As we computed the coordinates of each vertex v_i, we can compute the length of each circular arc and the value of the external angles using the center of the circles and trigonometric functions. The absolute value of the geodesic curvature on a given circle C_i is $\frac{d_i}{r_0 r_{0i}}$ [61], where d_i is the distance between the centers of S_0 and C_i. For a non-great circle, using the definition of the geodesic curvature based upon the covariant derivative [75, 40], we can see that the geodesic curvature on an oriented circle is positive if the orientation induces the smallest part of S_0 bounded by the circle, and negative otherwise.

The draft version of the implementation features 100 lines of code. The interface of the function implementing the aforementioned formula to compute the area of a face of the arrangement is the following:

```
//compute the area of a face of the arrangement
template <class T_HDS>
double compute_area_spherical_face(typename T_HDS::Face_handle Face_handle){...}
```

Grid class The class `Grid_partition` implements a partition of a 3D cuboid bonding box by smaller cubes. Each such cube stores a set of objects which type is specified by a traits class given as template parameter. This template parameter also provides a method to initialize the parameter of the grid, together with a method to locate an object within the grid. The complete draft version features about 400 lines of code.

```
#include <boost/tuple/tuple.hpp>
#include <boost/tuple/tuple_comparison.hpp>

using boost::tuple;
using boost::make_tuple;

//Required traits class for Grid_partition.
template<class Obj>
struct Traits_for_grid{
  typedef Obj Object;
  typedef CGAL::Cartesian<double>::Point_3 Point_3;
```

```
  //Initialize the tuple XYZlim and the lower corner of the grid.
  template <class input_iterator>
  static void init_grid(const input_iterator& it_beg,const input_iterator& it_end,
                        tuple<int,int,int>& XYZlim,double& rmax,Point_3& lower_corner)
  {...}

  //Indicate to which cube belong an object according to XYZlim and lower_corner.
  static tuple<int,int,int> locate_cube(const Obj& V,const tuple<int,int,int>& XYZlim,
                                        const double& edge_length,
                                        const Point_3& lower_corner){...}

  //Convert an input_iterator to an Object.
  static Object& get_object(const input_iterator& it){...};

};

template <class Traits>
class Grid_partition{
public:
//Internal class definitions.
  class iterator{...}//iterator over cubes
  class objects_iterator{...}//iterator over all the objects stores in all the cubes
  class neighbors_iterator{...}//iterator over the neighbors of a cube


  struct unit_cube{ //definition of a cube in the grid
  //Data members.
    std::list<typename Traits::Object> data;
    typedef typename std::list<typename Traits::Object>::iterator iterator;
  //Functions.
    unit_cube(typename Traits::Object& V){data.push_front(V);}
    void insert(typename Traits::Object& V){data.push_front(V);}
    unsigned size(){return data.size();}
    iterator begin(){return data.begin();}
    iterator end(){return data.end();}
  };

private:
//Internal definitions.
  typedef tuple<int,int,int> internal_coordinate;
  typedef std::map<internal_coordinate,unit_cube* > Grid;


public:
//Data members.
  tuple<int,int,int> XYZlim;//number of cubes in each direction
  Grid grid;//contains cubes from 0 to XYZlim-<1,1,1> for each direction
  double edge_length;//size of edges of cubes
  typename Traits::Point_3 lower_corner;//lower corner of the grid in the frame


public:
  template <class input_iterator>
```

```
  Grid_partition(const input_iterator& it_beg,const input_iterator& it_end){
   Traits::init_grid(it\_beg,it\_end,XYZlim,edge\_length,lower\_corner);
  }

  ~Grid_partition(){
    for(typename Grid::iterator it=grid.begin();it!=grid.end();++it)
      delete (*it).second;
  }

  void insert_in_a_cube(typename Traits::Object& V){
    internal_coordinate T=Traits::locate_cube(V,XYZlim,edge_length,lower_corner);
    typename Grid::iterator it=grid.find(T);
    if (it==grid.end())
      grid[T]=new unit_cube(V);
    else
      (*it).second->insert(V);
  }

  void fill(const input_iterator& it_beg,const input_iterator& it_end){
    for(input_iterator it=it_beg;it!=it_end;++it){
      typename Traits::Object v=Traits::get_object(it);
      insert_in_a_cube(v);
    }
  }

  unit_cube* get_cube(const internal_coordinate& t){
    typename Grid_partition::iterator it=grid.find(t);
    return (it==grid.end())?(NULL):((*it).second);
  };

//Access data trough iterators.
  iterator begin(){...}
  iterator end(){...}
  neighbors_iterator neighbors_begin(const internal_coordinate& int_coord){...}
  neighbors_iterator neighbors_end(){...}
  objects_iterator objects_begin(){...}
  objects_iterator objects_end(){...}
}
```

Spherical arrangement class The class `Arrangement_on_a_sphere` is templated by a Spherical Kernel and a HDS. The member function `Compute` provides a way to compute an arrangement of circles on a sphere and to store the result in a object of type `Argt_DS`. A member function is also provided to construct for each face the list of spheres covering it. The complete draft version features about 10,000 lines of code.

```
//The class performing the sweep algorithm on a sphere.
template<class SK,class T_HDS>
class Sweep_arc_sphere<SK,T_HDS>{...}

template<class SK,class T_HDS>
struct Argt_DS{
```

```
//Definition from the internal arrangement class.
  //The circle type.
  typedef typename SK::Circle_on_reference_sphere_3 Circle_on_reference_sphere_3;
  //Map a sphere to a list of spheres.
  typedef typename Sweep_arc_sphere<SK,T_HDS>::sphere_to_lsphere sphere_to_lsphere;
  //Map a sphere to another sphere.
  typedef typename Sweep_arc_sphere<SK,T_HDS>::sphere_to_sphere sphere_to_sphere;
  //Data structure containing the covering lists.
  typedef typename Sweep_arc_sphere<SK,T_HDS>::hedge_to_face hedge_to_face;
  typedef typename SK::Sphere_3 Sphere_3;
//Data resulting from the arrangement.
  T_HDS hds; //the HDS containing the arrangement
  std::list<const Circle_on_reference_sphere_3> List_S;// the list of circles on S_0
  std::map<const Circle_on_reference_sphere_3*,Sphere_3> map_type;
  hedge_to_face covering_ds;//data structure encoding the covering tree
                          //of input spheres
  sphere_to_sphere map_opposite;//map a primary sphere to
                              //its opposite sphere---if exists
  sphere_to_lsphere map_friends;//map a primary or opposite sphere
                              //to its list of friends
  Sweep_arc_sphere<SK,T_HDS>* SAS_ptr;//a pointer on the structure used
                                    //to compute the arrangement
  void build_covering_lists(){...}//explicitly construct the covering lists
                                //with primary and opposite spheres
};

template<class SK,class T_HDS>
struct Arrangement_on_a_sphere{
  typedef std::list<typename SK::Sphere_3*> lsphere;
  typedef typename SK::Point_3 Point_3;
  typedef typename SK::FT FT;

//Fill into an output iterator spheres covering a given face.
  template<class output_iterator>
  static void get_face_covering(const typename T_HDS::Halfedge *HH,
                              const Argt_DS<SK,T_HDS>& ADS,
                              const output_iterator output){
  //Recover list of spheres covering the face induced by halfedge pointed by HH.
    typename hedge_to_face::iterator cov_ds_it=ADS.covering_ds.find(HH);
    for (typename lsphere::iterator it=cov_ds_it->sphere_list->begin();
         it!=cov_ds_it->sphere_list->end();++it){
      *output++=*it;
      typename Argt_DS<SK,T_HDS>::sphere_to_lsphere::iterator it_fr=
        ADS.map_friends.find(*it);
      if (it_fr!=ADS.map_friends.end()){
        lsphere* LS=(*it_fr).second;
        for(typename lsphere::iterator it2=LS->begin();it2!=LS->end();++it2)
          *output++=*it2;
      }
    }
  }

//Construct a list of circle on a sphere from a set of intersecting spheres.
  template<class sphere_container,class T_Argt_DS>
```

```
  static void fill_sphere_list(sphere_container& spct,Point_3& c_0,T_Argt_DS& ADS,
                               const FT& r0){
    for (typename sphere_container::iterator it=spct.begin();it!=spct.end();++it)
    {
      //Functions getx, gety, getz, getr2 return either the corresponding center
      //coordinate, or the squared radius of the sphere.
      Point_3 c_i(getx(*it),gety(*it),getz(*it));
      //All circles lie on a sphere centered at the origin to simplify predicates.
      ADS.List_S.push_front(Sweep_arc_sphere<SK,T_HDS>::
        make_circle_on_reference_sphere_3(r0,getr2(*it),
          Point_3(c_i.x()-c_0.x(),c_i.y()-c_0.y(),c_i.z()-c_0.z())));
      ADS.map_type[&(*ADS.List_S.begin())]=*it;
    }
  }


  template<class sphere_container, class T_Argt_DS>
  static void Compute(sphere_container& spct,const FT& x_0,const FT& y_0,
                      const FT& z_0,const FT& r_0, T_Argt_DS& ADS ){
    Point_3 c_0(x_0,y_0,z_0);
  //Each sphere in spct is associated to a circle on S_0 in ADS.List_S.
    fill_sphere_list(spct,c_0,ADS,r_0);
  //Initialize the sweep process.
    ADS.SAS_ptr=new Sweep_arc_sphere<SK,T_HDS>(r_0,CGAL::ORIGIN,ADS.List_S,
                                               &ADS.hds,&ADS.covering_ds,
                                               &ADS.map_opposite,&ADS.map_friends);
  //Handle events while queue is not empty.
    while(ADS.SAS_ptr->not_end())
      ADS.SAS_ptr->Handle_Event();
  //Merge the faces cut at the begin of the sweep process.
    ADS.SAS_ptr->merge_Virtual_Faces();
  //Clean the internal data structure used for the sweep.
    delete ADS.SAS_ptr;
    ADS.SAS_ptr=NULL;
  }
};
```

An example of an application We give an example of how to combine the two previously defined classes to compute on each sphere of a given set, the arrangement of circles induced by intersecting spheres. This is the same kind of implementation we have for computing the arrangements on each atomic sphere of a molecular model. The complete draft version features about 2,500 lines of code.

```
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Exact_predicates_exact_constructions_kernel.h>
#include <CGAL/Cartesian.h>
#include <CGAL/Interval_nt.h>
#include <CGAL/Spherical_kernel_3.h>
#include <CGAL/HalfedgeDS_default.h>

//Custom classes to define HDS with holes
class my_items{...}
```

```
class hds_traits_with_holes{...}

class Argts_computation{
  //Linear kernels.
  typedef CGAL::Exact_predicates_inexact_constructions_kernel input_LK;
  typedef CGAL::Cartesian<CGAL::Interval_nt<false> >      fast_interval_LK;
  typedef CGAL::Exact_predicates_exact_constructions_kernel exact_LK;
  //Algebraic kernels.
  typedef CGAL::Algebraic_kernel_for_spheres_2_3<typename fast_interval_LK::FT>
    fast_interval_AK;
  typedef CGAL::Algebraic_kernel_for_spheres_2_3<typename LK::FT>
    exact_AK;
  //Spherical kernels.
  typedef CGAL::Spherical_kernel_3<fast_interval_LK,fast_interval_AK>
    fast_interval_SK;
  typedef CGAL::Spherical_kernel_3<exact_LK,exact_AK>
    exact_SK;
  //HDS declaration.
  typedef CGAL_HALFEDGEDS_DEFAULT <hds_traits_with_holes,my_items > HDS;
  //The grid containing the input spheres.
  typedef Grid_partition<Traits_for_grid<input_LK::Sphere_3 > > my_Grid;
  //Compute the arrangement on sphere pointed by it, circles defined by spheres
  //in neighbors, for each face print the number of spheres covering it
  //together with its area.
  template<class SK>
  void compute_one_arrangement(const std::list<my_Grid::Object>& neighbors,
                               my_Grid::objects_iterator it);
  //Report all the spheres intersecting v into N using grid G.
  void report_intersecting_spheres_with_grid(const my_Grid::Object& v,my_Grid& G,
                                             std::list<my_Grid::Object>& N)
  public:
  //Compute all the arrangements of circles on all spheres
  //using the filtering strategy at the arrangement level.
  void compute_all_arrangements();
}


template<class SK>
void Argts_computation::compute_one_arrangement
  (const std::list<my_Grid::Object>& neighbors,my_Grid::objects_iterator it)
{
  //Functions getx, gety, getz, getr2 return either the corresponding center
  //coordinate, or the squared radius of the sphere.
  Argt_DS<SK,HDS> ADS;
  Arrangement_on_a_sphere<SK,HDS>::Compute(neighbors,getx(*it),gety(*it),
                                           getz(*it),getr2(*it),ADS);
  //Compute area and print it together with number of spheres covering each face.
  typename SK::FT r_0=getr2(*it);
  ADS.build_covering_lists();

  if (ADS.hds.faces_begin()==ADS.hds.faces_end()){
    double area_tot=4*M_PI*CGAL::to_double(r_0*r_0);
    std::cout << "0 ; " << area_tot<< std::endl;
  }
```

```
  else
    for (HDS::Face_iterator itf=ADS.hds.faces_begin();
         itf!=ADS.hds.faces_end();++itf)
    {
      double area_face=compute_area_spherical_face<HDS>(itf);
      std::list<typename SK::Sphere_3*> covering_list;
      std::insert_iterator<std::list<typename SK::Sphere_3*> >
        it_inc(covering_list,covering_list.begin());
      Arrangement_on_a_sphere<SK,HDS>::get_face_covering(&(*itf->halfedge()),
                                                         ADS,it_inc);
      std::cout << covering_list.size() << " ; " << area_face << std::endl;
    }
}

void Argts_computation::report_intersecting_spheres_with_grid
  (const my_Grid::Object& v,my_Grid& G,std::list<my_Grid::Object>& N)
{
  //Visit neighbor cubes.
  for (typename my_Grid::neighbor_iterator it=G.neighbors_begin(G.locate_cube(v));
       it!=G.neighbors_end();++it)
    for (typename my_Grid::unit_cube::iterator itc=(*it).begin();
         itc!=(*it).end();++itc)
      if (input_LK::do_intersect(v,*itc))
        N.push_front((*itc));
  //Neighbors in the same cube.
  for (typename my_Grid::unit_cube::iterator itc=G.get_cube(v)->begin();
       itc!=G.get_cube(v)->end();++itc)
    if(*itc!=v)
      if (input_LK::do_intersect(v,*itc))
        N.push_front((*itc));
}

void Argts_computation::compute_all_arrangements()
{
  std::list<my_Grid::Object> neighbors;
  for (my_Grid::objects_iterator it=G.first_object();it!=G.last_object();++it){
    neighbors.clear();
    report_intersecting_spheres_with_grid(*it,G, neighbors);
    //Compute arrangement, using a filtering technique at the arrangement level:
    //predicates are first evaluated using double intervals, and in case of failure,
    // the whole arrangement is computed exactly.
    try{
      CGAL::Interval_nt_advanced::Protector P;//handle rounding mode problems
      compute_one_arrangement<fast_interval_SK>(neighbors,it);
    }
    catch(CGAL::Interval_nt_advanced::unsafe_comparison){
      compute_one_arrangement<exact_SK>(neighbors,it);
    }
  }
}
```

Chapitre 8

Conclusion and Outlook

Results in computational geometry We have presented a generalization of the Bentley-Ottmann algorithm to the spherical setting. This generalization accommodates the calculation of the exact arrangement of circles on a sphere (chapter 3), as well as the construction of the corresponding half-edge data structure on the fly (section 4.2). Moreover, assuming that each circle comes from the intersection between the central sphere and a neighboring sphere, each coming with an accompanying ball, we explain how to efficiently compute the covering lists of faces of the arrangement (section 4.3). We notice that while exactness of the arrangement is not a goal per se, it is one way to achieve robustness, at a modest computational overhead (section 5.5).

High quality geometric code relies on four virtues: robustness, efficiency, modularity, re-usability. Although spheres are amongst the most elementary geometric objects, no library of essential primitives was available to deal with them. We answered this need (chapter 5), by developing the CGAL 3D Spherical Kernel concept, i.e. requirements for the basic types and operations required to deal with spheres, planes, circles, circular arcs and points in 3D. A clear distinction is made between the algebraic and the geometric aspects on the one hand, and on the concepts of a kernel and its implementation on the other hand. The concept is accompanied by an implementation.

Outlook in computational geometry Additional work is needed to comply with the CGAL standards for a public release of our arrangement code and of components of the 3D Spherical Kernel dedicated to operations on a reference sphere. We also plan to finish the implementation of the primitives and constructions needed (based on the 3D Spherical Kernel) for the future new generic CGAL arrangement package [25]. A traits class based on our work would allow to handle circular arcs of all types of circles using this algorithm. A comparison with our algorithm may be interesting then. Moreover, investigations to find a mechanism to implement the covering list within this package (using the C++ concept of *visitor*), would open additional possibilities. In particular, the MPII group in Saarbrücken should soon release a traits class for the generic arrangement algorithm, to compute the arrangement of curves on a given quadric, induced by the intersection with a set of quadrics. In the case of ellipsoids, this should allow to use this package to obtain better molecular models with adapted properties. The computation of the area of each face of this arrangement (a difficult problem) should also ideally be carried out. Finally, the development of a generic data structure to combine a set of 2D surface arrangements into a 3D volumetric decomposition is the ultimate step when dealing with such representations. An efficient and robust algorithm to compute the volume of each 3D cell (either bounded by spheres, or by ellipsoids) will also have to be developed.

For the volume of a union of balls, we, together with Quentin Mérigot, recently completed a proof-of-concept implementation following the inclusion-exclusion strategy, and using formulae in [18] to compute the volume of intersection of up to four spheres. Unfortunately, our implementation is slow and not robust. The speed problem is on the one hand due to the fact that we use an inclusion-exclusion formula, and on the other hand to some predicates whose robustness incurs a significant overhead. About the robustness problem, given a set of balls (up to four), the idea to compute the volume of the intersection of these balls is to identify the configuration of the balls (using predicates) so as to apply the correct formula.

Moreover some parts of our formulae need more investigation as they feature quotients of quantities that can be close to zero. In the future, we may also try to discard the inclusion-exclusion formula, to resort to one computation per ball [76, 141], the ultimate goal being to be able to compute the volume of each cell in the 3D arrangement of a set of balls.

The current running times of our algorithm for conformer selection are comparable to those required by the docking algorithms exploiting the conformer selections. The problem of coming up with an algorithm, on the one hand with a better approximation of the optimal solution, and on the other hand a good output sensitive running time deserves further investigation.

Results in computational structural biology The flexibility of systems undergoing large motions cannot yet be explored using classical molecular dynamics simulations. The manipulation of discrete ensemble of pre-generated conformers is a strategy we can resort to in that case. This strategy is valid for conformers of any size, namely for side chains, protein loops or domains. Because the generation of such ensembles does not take into account the whole environment of the conformer (in the whole protein or complex), the energy functionals used to compute the energy of a conformer cannot, in general, be directly related to the thermodynamic equilibrium between the conformations. This observation calls for the development of methods providing a rather uniform sampling of the conformational space of the fragment considered, so as to retain conformers avoiding obvious steric clashes. But such algorithms face the central difficulty of characterizing the conformational space coverage, to maximize the diversity of the conformers. In this context, we make three contributions.

First, we presented geometric optimization methods geared towards the characterization and the selection of conformational diversity. Given a collection of conformers, the methods aim to return a selection maximizing a functional of the volume occupied by the conformers, or of the molecular surface exposed by the conformers. Greedy strategies are used to solve these problems, and theoretical bounds are proved.

Second, for the particular problem of the optimization of the MSA, we make a geometric assessment of the conformational diversity of the conformers selected, based upon experiments carried out on three flexible protein loops. We show that the number of conformers needed by our greedy strategy to match the MSA of standard selection methods is smaller then for these selection methods by *one and two orders of magnitude*, depending on the particular system and the model (atomic or coarse). Moreover, tracking the variation of the MSA together with topological informations of the selection (the Betti numbers) yields insights on the quality of the coverage of the conformational space associated to a collection of conformers.

Third, using coarse representations of three of these protein models, we compare the results of a multi-copy docking algorithm, for two sets of copies: one selected by our greedy strategy—`Greedy`, and one generated by a standard hierarchical clustering algorithm—`HClust`. For six docking protocols (three systems, Bound and Unbound receptors for each), the results decompose as follows: three favorable to `Greedy`, two ties, one favorable to `HClust`.

Outlook in computational structural biology Our developments on conformer selection have a number of direct applications. First, our characterization of the conformational diversity based upon geometric and topological measures, together with the greedy strategy, should prove useful to improve the conformational space coverage of conformer generation methods. For example, algorithms `Loopy` and `Direx` could bootstrap from our selections to improve their conformational diversity. Second, the positive results obtained for coarse docking call for further development. In particular, bootstrapping from the selections of coarse conformers generated by `Greedy` to generate high-quality atomic models should improve the predictions for challenging flexible protein-protein complexes. Interestingly, this work also raises the following open theoretical question: For a particular problem (conformer generation, docking), what is the particular functional to be optimized (volume-based, surface-based) ? Volume-based and surface-based functionals are obvious candidates, especially since the surface exposed by a collection of balls is the *geometric locus* where interaction occurs. But these might be seen as *first approximations* to quantify the conformational diversity. That is, because covering a $3D$ volume with a collection of conformers does not admit a unique solution, it might actually be necessary to incorporate into the

functional some measure of the multiplicity of the cells of the volume or surface arrangements, to guarantee that each portion of space is covered the same number of times.

A number of modeling situations should benefit from the information encoded in the arrangement of circles on a sphere with covering lists. In particular, our implementation generalizes all previously developed algorithms to compute VdW surface area or SAS area, since in addition to computing both quantities, it can also compute most of (if not all) the interaction coefficients previously defined with the model of union of balls.

Contact surfaces covered by more than one atomic sphere represent more than 80% of the surface of expanded Van der Waals spheres in protein-protein complexes. A complete study involving these parameters should improve our understanding and prediction for binding regions (pockets for proteins-drug complexes), and may lead to the development of scoring functions.

The good performance of knowledge-based potentials in previous structure prediction studies is incontestable. The quality of the coordinates of new structures being determined experimentally increases, together with their number. The amount of information available must not be neglected as it encodes the final state of the folding process in nature. The careful study of atomic environments encoded with arrangements should reveal new parameters to derive efficient knowledge-based potentials.

Most of the side chain placement algorithms work using a library of rotameric conformations. The study of classes of equivalence of surface arrangement (for atoms or residues), could lead to the definition of a structural alphabet to build protein structures.

Chapitre 9

Conclusions et Perspectives

Résultats en géométrie algorithmique Nous avons présenté une adaptation de l'algorithme de Bentley-Ottmann au cas sphérique. Cette adaptation permet le calcul exact de l'arrangement de cercles sur une sphère, ainsi que la construction d'une structure dite en demi-arète à la volée. De plus, en supposant que chaque cercle provient de l'intersection de la sphère centrale et d'une sphère voisine, nous avons montré comment construire efficacement pour chaque face de l'arrangement, la liste des sphères dont les boules couvrent cette face. Par la même occasion, on a observé que même si l'exactitude n'est pas un but en soit, c'est une des façons de garantir la robustesse de l'algorithme, au prix d'une faible augmentation du temps de calcul.

Les implémentations d'algorithmes géométriques de bonne qualité reposent sur quatre principes : robustesse, efficacité, modularité et réutilisablilité. Bien que les sphères soient parmi les objets géométriques les plus élémentaires, aucune bibliothèque fournissant les primitives essentielles n'était disponible pour manipuler ces objets. Nous avons comblé ce manque en développant le concept CGAL du noyau sphérique, c'est à dire un ensemble de pré requis pour les types de bases et les opérations dédiées aux sphères, plans, arcs de cercles et points en 3D. Une séparation est faîte entre d'un côté l'algèbre et les aspects géométriques, et de l'autre le concept de noyau. Une implémentation de ce concept est aussi fournie.

Perspectives en géométrique algorithmique Notre code pour calculer des arrangements de cercles sur une sphère, ainsi que les composants du noyau sphérique dédiés aux opérations sur une sphère de référence ont besoin d'être améliorés pour répondre aux critères de CGAL pour une diffusion publique. Nous devons aussi terminer l'implémentation de la classe de traits (à partir du noyau sphérique) pour le futur paquet générique CGAL d'arrangements sur une surface [25]. Ces primitives basées sur nos travaux devraient permettre de traiter tous les types de cercles sur une sphère utilisant cet algorithme. Une comparaison avec notre algorithme devrait être intéressante. De plus, trouver un mécanisme pour implémenter les listes de couverture dans leur algorithme (en utilisant le concept C++ de *visiteur*), devrait offrir d'autres possibilités. En particulier, un groupe du MPII de Saarbrücken devrait bientôt publier des primitives pour le paquet CGAL d'arrangement générique afin de pouvoir calculer l'arrangement d'un ensemble de courbes sur une quadrique, induite par l'intersection avec un ensemble de quadriques. Dans le cas des ellipsoïdes, ceci permettrait d'avoir une méthode qui offrirait la possibilité de manipuler des modèles moléculaires plus adaptés aux propriétés des atomes. Le calcul de l'aire de chaque face de cet arrangement (un problème difficile) devrait aussi idéalement être traité. Pour finir, le développement d'une structure de données générique permettant de combiner un ensemble d'arrangement sur des surfaces (en 2D) pour obtenir une décomposition du volume (en 3D) d'une molécule, est l'étape ultime pour les modèles utilisant une représentation par un ensemble de boules. Un algorithme efficace et robuste pour calculer le volume de chaque cellule 3D (soit bordée par des sphères ou des ellipsoïdes) devra aussi faire l'objet de recherches.

A propos du volume d'une union de boules, avec Quentin Mérigot, nous avons récemment réalisé une implémentation suivant une formule d'inclusion-exclusion, et en utilisant les formules de [18] pour calculer le volume de l'intersection d'au plus quatre sphères. Malheureusement, notre implémentation est lente et peu robuste. Le problème de vitesse est dû à l'utilisation de la formule d'inclusion-exclusion,

mais aussi aux prédicats dont la robustesse se traduit par un surcoût au niveau des temps de calcul. En ce qui concerne les problèmes de robustesse, étant donné un ensemble d'au plus quatre boules, l'idée de la méthode utilisée est d'identifier dans quelle configuration sont les boules (à l'aide de prédicats) afin d'appliquer la formule correspondante. Certaines parties de ces formules ont besoin d'être retravaillées car elles impliquent des quotients de quantités qui peuvent être proches de zéro. On pourra aussi essayer de se passer de la formule d'inclusion-exclusion, pour se limiter à un calcul par boule [76, 141], tout en gardant l'idée de pouvoir calculer le volume de chaque cellule dans l'arrangement 3D d'une union de boules.

Pour ce qui est du problème de la sélection des conformers, les temps de calcul actuel sont du même ordre de grandeur que ceux de l'algorithme d'amarrage qui utilise une sélection de conformers. Parvenir à fournir un algorithme avec à la fois une meilleure approximation de la solution optimale, et de meilleurs temps de calcul sensibles à la sortie est aussi un de nos objectifs.

Résultats en biologie structurale computationelle Les simulations classiques de dynamique moléculaire ne permettent pas encore d'explorer la flexibilité des systèmes qui subissent des mouvements de grandes amplitudes. La manipulation d'ensembles finis de conformers pré-calculés est une solution pertinente dans ce cas. Cette stratégie est valable pour des fragments de toute taille (chaînes latérale, boucles ou domaines de protéines). Comme la génération de tels ensembles ne prend pas en compte l'environnement complet de ces fragments (dans la protéine ou le complexe), les fonctions d'énergie utilisées pour qualifier un conformer ne peuvent pas, en général, être reliées à l'équilibre thermodynamique entre les conformations. Cette observation montre le besoin de s'appuyer sur d'autre méthodes qui fournissent un échantillonage quasi uniforme de l'espace de conformations des fragments considérés. Mais de tels algorithmes sont confrontés au problème de la caractérisation de l'espace couvert, afin de maximiser la diversité des conformers. Dans ce contexte, nous apportons trois contributions :

Premièrement, nous présentons des méthodes d'optimisation orientées vers la caractérisation et la sélection d'un ensemble de conformations varié. Étant donné une collection de conformers, le but de la méthode est de fournir une sélection qui maximise une fonction du volume occupé par les conformers, ou la surface moléculaire exposée par les conformers. Des stratégies gloutonnes sont utilisées pour résoudre ces problèmes, et des bornes théoriques sont prouvées. Deuxièmement, pour le problème de l'optimisation de la MSA, nous présentons un moyen géométrique pour expliquer la diversité des conformers sélectionnés, basé sur des mesures faites sur quatre boucles flexibles d'une protéine. Nous montrons que notre stratégie gloutonne est capable de surpasser une méthode de sélection classique pour la valeur de la MSA de la sélection. De plus, l'observation de la variation de la MSA avec des informations topologiques sur la sélection (les nombres de Betti) donne une idée de la qualité de la couverture de l'espace des conformations par cette sélection. Troisièmement, en utilisant une représentation gros grains de trois de ces protéines, nous comparons les résultats d'un algorithme d'amarrage à plusieurs copies, et ce pour deux ensembles de copies : celui sélectionné par notre algorithme glouton (`Greedy`), et celui sélectionné par un algorithme de classification hiérarchique classique (`HClust`). Pour six des protocoles d'amarrage (trois systèmes, avec la forme libre ou lié du récepteur), les résultats sont les suivants : trois sont favorables à `Greedy`, deux sont équivalents et un est favorable à `HClust`.

Perspectives en biologie structurale computationelle Nos algorithmes de sélection de conformations ont plusieurs applications directes. Premièrement, la caractérisation de la diversité des conformations basée sur des mesures géométriques et topologiques, couplée avec une stratégie gloutonne devraient se montrer utile pour améliorer la couverture de l'espace des conformations des méthodes de génération. Par exemple, les algorithmes `Loopy` et `Direx` pourraient s'appuyer sur nos sélections pour améliorer la diversité de des conformations proposées. Deuxièmement, les résultats positifs obtenus pour l'amarrage en gros grain montre le besoin d'aller plus loin. En particulier, l'utilisation de nos sélections de conformations en gros grain générées par `Greedy`, pour générer des modèles tout-atome de haute qualité, devraient améliorer les prédictions de complexes protéine-protéine. Ce travail lève des interrogations sur des aspects théoriques. Pour un problème particulier (génération de conformations, amarrage), la question du choix de la fonction à optimiser (basé sur le volume, l'aire,...) se pose. Utiliser le volume et l'aire sont des candidats naturels, en particulier parce que la surface exposée par une collection de boules est le

lieu géométrique où l'interaction a lieu. Mais ces critères peuvent être vues comme une première étape pour qualifier la diversité des conformations. Comme couvrir un volume en 3D avec une collection de conformations n'admet pas une solution unique, il peut s'avérer nécessaire d'intégrer dans la fonction à optimiser une façon de tenir compte de la multiplicité des cellules du volume ou de l'arrangement en surface, afin de garantir que chaque portion de l'espace est couverte le même nombre de fois.

Si l'on considère l'arrangement de cercles sur une sphère avec les listes de couverture, plusieurs problèmes de modélisation peuvent bénéficier des informations encodées. En particulier, notre implémentation généralise toutes les précédentes développées pour calculer l'aire de la surface de VdW ou de la SAS, puisqu'en plus de calculer ces quantités, notre méthode permet aussi de calculer la plupart (si ce n'est tous) des coefficients d'interaction définis précédemment pour les modèles utilisant une union de boules.

Pour ce qui est de la formation des complexes protéine-protéine, les surfaces de contact couvertes par plus d'une sphère atomique représentent plus de 80% des surfaces des sphères étendues de VdW. Une étude complète à l'aide de ces paramètres devrait améliorer notre compréhension et nos capacités de prédiction des zones d'interaction (les poches pour les médicaments,...), et contribuer au développement de fonctions de score.

Pour ce qui est de la prédiction de structure, les bonnes performances des fonctions empiriques dans des études précédentes est indéniable. La qualité des coordonnées des nouvelles structures déterminées expérimentalement augmente, ainsi que le nombre de telles structures. Cette quantité d'information disponible ne doit pas être négligée, car elle encode l'état final du processus de repliement dans la nature. L'étude approfondie des environnements atomiques à l'aide des arrangements peut révéler des paramètres pour définir de nouvelles fonctions empiriques.

La plupart des algorithmes de placement de chaînes latérales fonctionnent à partir de bibliothèques de conformations rotameriques. L'étude des classes d'équivalence des arrangements surfaciques (au niveau des résidus ou des atomes) permettrait peut-être de définir un alphabet structural pour prédire la structure des protéines.

Software

The molecular illustrations of this manuscript were produced using the software PyMOL [71]. The 2D drawings of this manuscript were produced using the IPE extensible drawing editor [173]. Ramachandran plots were produced using a plug-in of VMD [112]. The chemical formulae for amino acids were produced using BKChem [30]. The latex style sheet (`thloria`) used for this manuscript was written by Denis Roegel (`http://www.loria.fr/~roegel/TeX/TL.html`).

Bibliographie

[1] R. Abagyan and M. Totrov. Contact area difference (CAD): a robust measure to evaluate accuracy of protein models. *Journal of Molecular Biology*, 268(3):678–685, 1997.

[2] P. Agarwal. *Intersection and Decomposition Algorithms for Planar Arrangements.* Cambridge University Press, 1991.

[3] P. Agarwal, L. Guibas, A. Nguyen, D. Russel, and L. Zhang. Collision detection for deforming necklaces. *Computational Geometry: Theory and Applications*, 28:137–163, 2004.

[4] P. Agarwal and M. Sharir. Arrangements and their applications. *Handbook of Computational Geometry*, pages 49–119, 2000.

[5] H. Ahn, M. de Berg, P. Bose, S. Cheng, D. Halperin, J. Matoušek, and O. Schwarzkopf. Separating an object from its cast. *Computer-Aided Design*, 34(8):547–559, 2002.

[6] O. Aichholzer, F. Aurenhammer, T. Hackl, B. Jüttler, M. Oberneder, and Z. Sir. Computational and structural advantages of circular boundary representation. In *Proc. 10th International Workshop on Algorithms and Data Structures*, volume 4619 of *Lecture Notes Comput. Sci.*, pages 374–385, 2007.

[7] N. Akkiraju and H. Edelsbrunner. Triangulating the surface of a molecule. *Discrete Applied Mathematics*, 71(1-3):5–22, 1996.

[8] S. G. Akl and K. A. Lyons. *Parallel Computational Geometry.* Prentice Hall, 1993.

[9] N. Alon, H. Last, R. Pinchasi, and M. Sharir. On the Complexity of Arrangements of Circles in the Plane. *Discrete and Computational Geometry*, 26(4):465–492, 2001.

[10] N. Amato, M. Goodrich, and E. Ramos. Computing the arrangement of curve segments: divide-and-conquer algorithms via sampling. *Proc. 11th Annual Symposium on Discrete Algorithms*, pages 705–706, 2000.

[11] N. Amenta, S. Choi, and R. K. Kolluri. The power crust, unions of balls, and the medial axis transform. *Computational Geometry : Theory and Applications*, 19(2):127–153, 2001.

[12] M. V. A. Andrade and J. Stolfi. Exact algorithms for circles on the sphere. *International Journal of Computational Geometry and Applications*, 11:267–290, 2001.

[13] A. Andreeva, D. Howorth, J. Chandonia, S. Brenner, T. Hubbard, C. Chothia, and A. Murzin. Data growth and its impact on the SCOP database: new developments. *Nucleic Acids Research*, 36(Database issue):D419, 2008.

[14] D. H. Arrangements. *Handbook of Discrete and Computational Geometry Second Edition, J. O'Rourke, JE Goodman editors.* Chapman & Hall/CRC, 2004.

[15] D. Attali and H. Edelsbrunner. Inclusion-Exclusion Formulas from Independent Complexes. *Discrete and Computational Geometry*, 37(1):59–77, 2007.

[16] F. Aurenhammer. Power Diagrams: Properties, Algorithms and Applications. *SIAM Journal on Computing*, 16:78, 1987.

[17] F. Aurenhammer. Voronoï Diagrams - A Survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys*, 23(3), 1991.

[18] D. Avis, B. K. Bhattacharya, and H. Imai. Computing the volume of the union of spheres. *The Visual computer*, 3(6):323–328, 1988.

[19] I. Balaban. An optimal algorithm for finding segments intersections. *Proc. 11th Annual Symposium on Computational Geometry*, pages 211–219, 1995.

[20] D. Bandyopadhyay and J. Snoeyink. Almost-Delaunay simplices: nearest neighbor relations for imprecise points. *Proc. 5th Annual Symposium on Discrete Algorithms*, pages 410–419, 2004.

[21] D. Bandyopadhyay, A. Tropsha, and J. Snoeyink. Analyzing protein structure using almost-Delaunay tetrahedra. Technical Report TR03-043, University of North Carolina, Departement of computer science, 2003.

[22] K. Bastard, C. Prévost, and M. Zacharias. Accounting for loop flexibility during protein-protein docking. *Proteins*, 62(4):956–969, 2006.

[23] J. Bentley and T. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, 28(9):643–647, 1979.

[24] E. Berberich, A. Eigenwillig, M. Hemmer, S. Hert, K. Mehlhorn, and E. Schömer. Computational basis for conic arcs and boolean operations on conic polygons. In *Proc. 10th Annual European Symposium on Algorithms*, volume 2461 of *Lecture Notes Computer Science*, pages 321–327, 2002.

[25] E. Berberich, E. Fogel, D. Halperin, K. Mehlhorn, and R. Wein. Sweeping and maintaining two-dimensional arrangements on surfaces: A first step. In *Proc. 15th Annual European Symposium on Algorithms*, volume 4698 of *Lecture Notes Computer Science*, pages 645–656, 2007.

[26] E. Berberich, M. Hemmer, L. Kettner, E. Schömer, and N. Wolpert. An exact, complete and efficient implementation for computing planar maps of quadric intersection curves. *Proc. 21st Annual Symposium on Computational Geometry*, pages 99–106, 2005.

[27] H. Berman, K. Henrick, and H. Nakamura. Announcing the worldwide Protein Data Bank. *Nature Structural Biology*, 10(12):980–980, 2003.

[28] H. Berman, K. Henrick, H. Nakamura, and J. Markley. The worldwide Protein Data Bank (wwPDB): ensuring a single, uniform archive of PDB data. *Nucleic Acids Research*, 35(Database issue):D301, 2007.

[29] J. Bernauer, J. Aze, J. Janin, and A. Poupon. A new protein-protein docking scoring function based on interface residue properties. *Bioinformatics*, 23(5):555, 2007.

[30] Bkchem. `http://bkchem.zirael.org`.

[31] J.-D. Boissonnat and M. Yvinec. *Algorithmic Geometry*. Cambridge University Press, UK, 1998. Translated by Hervé Brönnimann.

[32] Boost, C++ libraries.
`http://www.boost.org`.

[33] C. Branden and J. Tooze. *Introduction to Protein Structure*. Garland Pub., 1999.

[34] K. Q. Brown. Comments on "Algorithms for reporting and counting geometric intersections". *IEEE Transactions on Computers*, 30(2):147–148, 1981.

[35] C. Burnikel, S. Funke, K. Mehlhorn, S. Schirra, and S. Schmitt. A separation bound for real algebraic expressions. In *Proc. 9th Annual European Symposium on Algorithms*, volume 2161 of *Lecture Notes Computer Science*, pages 254–265, 2001.

[36] P. M. M. de Castro, F. Cazals, S. Loriot, and M. Teillaud. Design of the CGAL spherical kernel and application to arrangements of circles on a sphere. Research Report 6298, INRIA, 2007. `https://hal.inria.fr/inria-00173124`.

[37] P. M. M. de Castro, S. Pion, and M. Teillaud. Exact and efficient computations on circles in CGAL. In *Abstracts 23rd European Workshop on Computational Geometry*, pages 219–222. Technische Universität Graz, Austria, 2007. Full version available as INRIA Research Report 6091, Exact and efficient computations on circles in CGAL and applications to VLSI design, `https://hal.inria.fr/inria-00123259`.

[38] P. M. M. de Castro and M. Teillaud. 3D Spherical Kernel. In *CGAL User and Reference Manual*. 3.4 edition.

[39] A. Canutescu, A. A. Shelenkov, and R. Dunbrack. A graph theory algorithm for protein side-chain prediction. *Protein Science*, 12:2001–2014, 2003.

[40] F. Cazals, F. Chazal, and T. Lewiner. Molecular shape analysis based upon the morse-smale complex and the connolly function. *Proc. 19th Annual Symposium on Computational Geometry*, pages 351–360, 2003.

[41] F. Cazals and S. Loriot. Computing the exact arrangement of circles on a sphere, with applications in structural biology. Research Report 6049, INRIA, 2006. `https://hal.inria.fr/inria-00118781`.

[42] F. Cazals and S. Loriot. Computing the exact arrangement of circles on a sphere, with applications in structural biology. In *Proc. 23th Annual Symposium Computational Geometry—Video/Multimedia track*, 2007.

[43] F. Cazals, F. Proust, R. Bahadur, and J. Janin. Revisiting the Voronoï description of protein-protein interfaces. *Protein Science*, 15(9):2082, 2006.

[44] CGAL, Computational Geometry Algorithms Library. `http://www.cgal.org`.

[45] P. Chakrabarti and J. Janin. Dissecting protein-protein recognition sites. *Proteins: Structure, Function and Genetics*, 47(3):334–343, 2002.

[46] J. Chambers, W. Cleveland, B. Kleiner, and P. Tukey. *Graphical Methods for Data Analysis*. Belmont, 1983.

[47] F. Chazal, D. Cohen-Steiner, and A. Lieutier. A sampling theory for compact sets in Euclidean space. *Proc. 22nd Annual Symposium on Computational Geometry*, pages 319–326, 2006.

[48] F. Chazal, D. Cohen-Steiner, and Q. Mérigot. Stability of boundary measure. Research Report 6219, INRIA, 2007. `http://hal.inria.fr/inria-00154798`.

[49] B. Chazelle. Reporting and counting segment intersections. *Journal of Computer and System Sciences*, 32(2):156–182, 1986.

[50] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *Journal of the ACM (JACM)*, 39(1):1–54, 1992.

[51] B. Chazelle et al. Application challenges to computational geometry: CG impact task force report. Technical Report TR-521-96, Princeton Univ., Apr. 1996.

[52] B. Chazelle et al. Application challenges to computational geometry: CG impact task force report. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, pages 407–463. American Mathematical Society, Providence, 1999.

[53] B. Chazelle, L. Guibas, and D. Lee. The power of geometric duality. *BIT Numerical Mathematics*, 25:76–90, 1985.

[54] C. Chen and D. Freedman. Quantifying homology classes ii: Localization and stability. *Preprint*, 2007. arXiv:0709.2512v2.

[55] R. Chen, J. Mintseris, J. Janin, and Z. Weng. A protein–protein docking benchmark. *Proteins*, 52(1):88–91, 2003.

[56] C. Chothia. Hydrophobic bonding and accessible surface area in proteins. *Nature*, 248(446):338–339, 1974.

[57] C. Chothia. Structural invariants in protein folding. *Nature*, 254(5498):304–308, 1975.

[58] C. Chothia. One thousand families for the molecular biologist. *Nature*, 357(6379):543–544, 1992.

[59] C. Chothia and J. Janin. Principles of protein-protein recognition. *Nature*, 256(5520):705–708, 1975.

[60] K. L. Clarkson and P. W. Shor. Applications of Random Sampling in Computational Geometry,II. *Discrete and Computational Geometry*, 4:387–421, 1989.

[61] M. Connolly. Analytical molecular surface calculation. *Journal of Applied Crystallography*, 16(5):548–558, 1983.

[62] M. Connolly. Computation of molecular volume. *Journal of the American Chemical Society*, 107(5):1118–1124, 1985.

[63] M. L. Connolly. Molecular surfaces: a review. *Network Science*, 14, 1996. `http://www.netsci.org/Science/Compchem/feature14.html`.

[64] L. Conte, C. Chothia, and J. Janin. The atomic structure of protein-protein recognition sites. *Journal of Molecular Biology*, 285(5):2177–2198, 1999.

[65] CORE Number Library. `http://cs.nyu.edu/exact/core_pages`.

[66] G. Cornuejols, M. Fisher, and G. Nemhauser. Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management Science*, 23(8):789–810, 1977.

[67] J. Cortés, T. Siméon, V. Ruiz de Angulo, D. Guieysse, M. Remaud-Siméon, and V. Tran. A path planning approach for computing large-amplitude motions of flexible molecules. *Bioinformatics*, 21:116–125, 2005.

[68] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[69] B. de Groot, D. van Aalten, R. Scheek, A. Amadei, G. Vriend, and H. Berendsen. Prediction of protein conformational freedom from distance constraints. *Proteins: Structure Function and Genetics*, 29(2):240–251, 1997.

[70] X. de la Cruz and M. Calvo. Use of surface area computations to describe atom–atom interactions. *Journal of Computer-Aided Molecular Design*, 15(6):521–532, 2001.

[71] W. L. DeLano. The PyMOL molecular graphics system. `http://www.pymol.org`.

[72] C. Delfinado and H. Edelsbrunner. An incremental algorithm for Betti numbers of simplicial complexes on the 3-sphere. *Computer Aided Geometric Design*, 12(7):771–784, 1995.

[73] A. Dhanik, P. Yao, N. Marz, R. Propper, C. Kou, G. Liu, H. van den Bedem, and J. Latombe. Efficient algorithms to explore conformation spaces of flexible protein loops. In *7th Workshop on Algorithms in Bioinformatics (WABI), Philadelphia*, pages 265–276, 2007.

[74] K. Dill. Theory for the folding and stability of globular proteins. *Biochemistry*, 24(6):1501–1509, 1985.

[75] M. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall Englewood Cliffs, 1976.

[76] L. Dodd and D. Theodorou. Analytical treatment of the volume and surface area of molecules formed by an arbitrary collection of unequal spheres intersected by planes. *Molecular Physics*, 72(6):1313–1345, 1991.

[77] R. S. Drysdale, G. Rote, and A. Sturm. Approximation of an open polygonal curve with a minimum number of circular arcs and biarcs. *Computational Geometry : Theory and Applications*, 41:31–47, 2008.

[78] H. Edelsbrunner. Weighted alpha shapes. Technical Report UIUCDCS-R-92-1760, Dept. Comput. Sci., Univ. Illinois, Urbana, IL, 1992.

[79] H. Edelsbrunner. The union of balls and its dual shape. *Discrete and Computational Geometry*, 13(1):415–440, 1995.

[80] H. Edelsbrunner and L. Guibas. Topologically sweeping an arrangement. *Journal of Computer and System Sciences*, 38(1):165–194, 1989.

[81] H. Edelsbrunner and L. Guibas. Corrigendum. *Journal of Computer and System Sciences*, 42(2):249–251, 1991.

[82] H. Edelsbrunner and E. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics (TOG)*, 9(1):66–104, 1990.

[83] D. Eisenberg and A. McLachlan. Solvation energy in protein folding and binding. *Nature*, 319(6050):199–203, 1986.

[84] I. Z. Emiris, A. Kakargias, S. Pion, M. Teillaud, and E. P. Tsigaridas. Towards an open curved kernel. In *Proc. 20th Annual Symposium on Computational Geometry*, pages 438–446, 2004.

[85] EXACUS, Efficient and Exact Algorithms for Curves and Surfaces. `http://www.mpi-inf.mpg.de/projects/EXACUS`.

[86] E. Eyal and D. Halperin. Dynamic maintenance of molecular surfaces under conformational changes. In *Proc. 21st Annual Symposium on Computational Geometry*, pages 45–54, 2005.

[87] E. Eyal, R. Najmanovich, B. J. Mc Conkey, M. Edelman, and V. Sobolev. Importance of solvent accessibility and contact surfaces in modeling side-chain conformations in proteins. *Journal of Computational Chemistry*, 25(5):712–724, 2004.

[88] A. Fabri and S. Pion. A generic lazy evaluation scheme for exact geometric computations. In *Proc. 2nd Library-Centric Software Design*, pages 75–84, 2006.

[89] U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[90] A. Fiser, R. Do, and A. Šali. Modeling of loops in protein structures. *Protein science*, 9(09):1753–1773, 2000.

[91] P. Fishburn and W. Gehrlein. Pick-and choose heuristics for partial set covering. *Discrete Applied Mathematics*, 22(2):119–132, 1989.

[92] E. Fogel, D. Halperin, L. Kettner, M. Teillaud, R. Wein, and N. Wolpert. Arrangements. In J.-D. Boissonnat and M. Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, pages 1–66. Springer-Verlag, Mathematics and Visualization, 2006.

[93] E. Fogel and M. Teillaud. Generic programming and the CGAL library. In J.-D. Boissonnat and M. Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, pages 313–320. Springer-Verlag, Mathematics and Visualization, 2006.

[94] F. Fogolari, L. Pieri, A. Dovier, L. Bortolussi, G. Giugliarelli, A. Corazza, G. Esposito, and P. Viglino. Scoring predictive models using a reduced representation of proteins: model and energy definition. *BMC Structural Biology*, 7(1):15, 2007.

[95] R. Fraczkiewicz and W. Braun. Exact and efficient analytical calculation of the accessible surface areas and their gradients for macromolecules. *Journal of Computational Chemistry*, 19(3):319–333, 1998.

[96] S. Funke and K. Mehlhorn. LOOK: A Lazy Object-Oriented Kernel for geometric computation. In *Proc. 16th Annual Symposium on Computational Geometry*, pages 156–165, 2000.

[97] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, 1979.

[98] D. M. Geometric Intersections, Mount. *Handbook of Discrete and Computational Geometry Second Edition, J. O'Rourke, JE Goodman editors*. Chapman & Hall/CRC, 2004.

[99] M. Gerstein and C. Chothia. Packing at the protein-water interface. *Proceedings of the National Academy of Sciences*, 93(19):10167–10172, 1996.

[100] M. Gerstein and F. M. Richards. Protein Geometry: Volumes, Areas, and Distances. *International Tables for Crystallography*, F:531–539, 2001.

[101] K. Gibson and H. Scheraga. Exact calculation of the volume and surface area of fused hard-sphere molecules with unequal atomic radii. *Molecular Physics*, 62(5):1247–1265, 1987.

[102] GMP, GNU Multiple Precision Arithmetic Library. `http://www.swox.com/gmp`.

[103] A. Gordon. *Classification*. Chapman & Hall/CRC, 1999.

[104] L. Greene, T. Lewis, S. Addou, A. Cuff, T. Dallman, M. Dibley, O. Redfern, F. Pearl, R. Nambudiry, A. Reid, et al. The CATH domain structure database: new protocols and classification levels give a more comprehensive resource for exploring evolution. *Nucleic Acids Research*, 35:D291, 2007.

[105] R. Grunberg, J. Leckner, and M. Nilges. Complementarity of structure ensembles in protein-protein binding. *Structure*, 12:2125–2136, 2004.

[106] D. Halperin and C. Shelton. A perturbation scheme for spherical arrangements with application to molecular modeling. *Computational Geometry: Theory and Applications*, 10(4):273–287, 1998.

[107] I. Halperin, B. Ma, H. Wolfson, and R. Nussinov. Principles of docking: An overview of search algorithms and a guide to scoring functions. *Proteins*, 47(4):409–43, 2002.

[108] M. Henle. *A Combinatorial Introduction to Topology*. Dover, 1994.

[109] K. Henrick and J. Thornton. PQS: a protein quaternary structure file server. *Trends in Biochemical Sciences*, 23(9):358–361, 1998.

[110] S. Hert, M. Hoffmann, L. Kettner, S. Pion, and M. Seel. An adaptable and extensible geometry kernel. *Computational Geometry : Theory and Applications*, 38:16–36, 2007.

[111] G. Hummer. Hydrophobic force field as a molecular alternative to surface-area models. *Journal of the American Chemical Society*, 121(26):6299–6305, 1999.

[112] W. Humphrey, A. Dalke, and K. Schulten. VMD: Visual molecular dynamics. *Journal of Molecular Graphics*, 14(1):33–38, 1996.

[113] L. James, P. Roversi, and D. Tawfik. Antibody multispecificity mediated by conformational diversity. *Science*, 2999:1362 1367, 2003.

[114] J. Janin and C. Chothia. The structure of protein-protein recognition sites. *Journal of Biological Chemistry*, 265(27):16027–16030, 1990.

[115] J. Janin and S. Wodak. The Third CAPRI Assessment Meeting Toronto, Canada, April 20–21, 2007. *Structure*, 15(7):755–759, 2007.

[116] J. C. Kendrew, G. Bodo, H. Dintzis, R. G. Parrish, H. Wyckoff, and P. D. C. Molecular structure of nucleic acids; a structure for deoxyribose nucleic acid. *Nature*, 181(4610):662–666, 1958.

[117] J. C. Kendrew, R. E. Dickerson, B. E. Strandberg, R. G. Hart, D. R. Davies, D. C. Phillips, and V. C. Shore. Structure of Myoglobin: A Three-Dimensional Fourier Synthesis at 2 Å. Resolution. *Nature*, 185:422–427, 1960.

[118] J. C. Kendrew, H. C. Watson, B. E. Strandberg, R. E. Dickerson, D. C. Phillips, and V. C. Shore. A Partial Determination by X-ray Methods, and its Correlation with Chemical Data. *Nature*, 190:666–670, 1961.

[119] L. Kettner. Halfedge data structures. In *CGAL User and Reference Manual*. 3.3 edition.

[120] L. Kettner, K. Mehlhorn, S. Pion, S. Schirra, and C. K. Yap. Classroom examples of robustness problems in geometric computations. *Computational Geometry : Theory and Applications*, 40:61–78, 2008.

[121] J. Keyser, T. Culver, D. Manocha, and S. Krishnan. MAPC: a library for efficient and exact manipulation of algebraic points and curves. In *Proc. 15th Annual Symposium on Computational Geometry*, pages 360–369, 1999.

[122] O. Kohlbacher and H. Lenhof. BALL-rapid software prototyping in computational molecular biology. *Bioinformatics*, 16(9):815–824, 2000.

[123] B. Krishnamoorthy and A. Tropsha. Development of a four-body statistical pseudo-potential to discriminate native from non-native protein conformations. *Bioinformatics*, 19(12):1540–1548, 2003.

[124] R. Laurie, T. Alasdair, and R. Jackson. Methods for the Prediction of Protein-Ligand Binding Sites for Structure-Based Drug Design and Virtual Ligand Screening. *Current Protein and Peptide Science*, 7(5):395–406, 2006.

[125] LEDA, Library for Efficient Data Types and Algorithms. `http://www.algorithmic-solutions.com/enleda.htm`.

[126] B. Lee and F. Richards. The interpretation of protein structures: estimation of static accessibility. *Journal of Molecular Biology*, 55(3):379–400, 1971.

[127] M. Lensink, R. Mendez, and S. Wodak. Docking and scoring protein complexes: CAPRI 3rd Edition. *Proteins*, 69(4):704–18, 2007.

[128] M. Levitt. A simplified representation of protein conformations for rapid simulation of protein folding. *Journal of Molecular Biology*, 104(1):59–107, 1976.

[129] M. Levitt. The birth of computational structural biology. *Nature Structural Biology*, 8:392–393, 2001.

[130] M. Levitt and C. Chothia. Structural patterns in globular proteins. *Nature*, 261(5561):552–558, 1976.

[131] M. Levitt, M. Hirshberg, R. Sharon, and V. Daggett. Potential energy function and parameters for simulations of the molecular dynamics of proteins and nucleic acids in solution. *Computer Physics Communications*, 91(1-3):215–231, 1995.

[132] C. Li, S. Pion, and C. K. Yap. Recent progress in exact geometric computation. *Journal of Logic and Algebraic Programming*, 64(1):85–111, July 2005. Special issue on the practical development of exact real number computation.

[133] C. Li and C. K. Yap. A new constructive root bound for algebraic expressions. In *Proc. 12th Symposium on Discrete Algorithms*, pages 496–505, 2001.

[134] X. Li and J. Liang. ComputationaL design of combinatorial peptide library for modulating protein-protein interactions. *Biocomputing 2005: Proceedings of the Pacific Symposium, Hawaii, USA 4-8 January 2005*, 28:39, 2005.

[135] X. Li and J. Liang. Geometric Cooperativity and Anticooperativity of Three-Body Interactions in Native Proteins. *Proteins: Structure, Function, and Bioinformatics*, 60:46–65, 2005.

[136] X. Li and J. Liang. Knowledge-Based Energy Functions for Computational Studies of Proteins. *Computational Methods for Protein Structure Prediction and Modeling*, pages 71–123, 2007.

[137] E. Lindahl, B. Hess, and D. van der Spoel. GROMACS 3.0: a package for molecular simulation and trajectory analysis. *Journal of Molecular Modeling*, 7(8):306–317, 2001.

[138] T. Liu and R. Samudrala. The effect of experimental resolution on the performance of knowledge-based discriminatory functions for protein structure selection. *Protein Engineering, Design and Selection*, 19(9):431–437, 2006.

[139] N. London and O. Schueler-Furman. Funnel Hunting in a Rough Terrain: Learning and Discriminating Native Energy Funnels. *Structure*, 16(2):269–279, 2008.

[140] A. Malevanets, F. Sirota-Leite, and S. Wodak. Mechanism and energy landscape of domain swapping in the B1 domain of protein G. *Journal of Molecular Biology*, 382(1):223–235, 2008.

[141] B. J. McConkey, V. Sobolev, and M. Edelman. Quantification of protein surfaces, volumes and atom-atom contacts using a constrained Voronoï procedure. *Bioinformatics*, 18(10):1365–1373, 2002.

[142] B. J. McConkey, V. Sobolev, and M. Edelman. Discrimination of native protein structures using atom-atom contact scoring. *Proceedings of the National Academy of Sciences*, 100(6):3215, 2003.

[143] K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, UK, 2000.

[144] J. Monod, J. Wyman, and J.-P. Changeux. On the nature of allosteric transitions: a plausible model. *Journal of Molecular Biology*, 12:88–118, 1965.

[145] J. Moult, K. Fidelis, A. Kryshtafovych, B. Rost, T. Hubbard, and A. Tramontano. Critical assessment of methods of protein structure prediction - Round VII. *Proteins: Structure Function and Genetics*, 69(S8):3–9, 2007.

[146] B. Mourrain, J.-P. Técourt, and M. Teillaud. On the computation of an arrangement of quadrics in 3d. *Computational Geometry: Theory and Applications*, 30:145–164, 2005. Special issue, 19th European Workshop on Computational Geometry.

[147] K. Mulmuley. A fast planar partition algorithm, II. *Journal of the ACM (JACM)*, 38(1):74–103, 1991.

[148] A. Murzin, S. Brenner, T. Hubbard, and C. Chothia. SCOP: A structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247(4):536–540, 1995.

[149] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, 14(1):265–294, 1978.

[150] S.-C. Ngan, L.-H. Hung, T. Liu, and R. Samudrala. *Scoring Functions for De Novo Protein Structure Prediction Revisited*, volume 413 of *Methods in Molecular Biology*. Springer, September 2007.

[151] S.-C. Ngan, M. Inouye, and R. Samudrala. A knowledge-based scoring function based on residue triplets for protein structure prediction. *Protein Engineering Design and Selection*, 19(5):187–193, 2006.

[152] J. Nievergelt and F. P. Preparata. Plane-sweep algorithms for intersecting geometric figures. *Commun. ACM*, 25(10):739–747, 1982.

[153] K. Noonan, D. O'Brien, and J. Snoeyink. Probik: Protein Backbone Motion by Inverse Kinematics. *The International Journal of Robotics Research*, 24(11):971, 2005.

[154] C. Orengo, A. Michie, S. Jones, D. Jones, M. Swindells, and J. Thornton. CATH–a hierarchic classification of protein domain structures. *Structure*, 5(8):1093–1108, 1997.

[155] L. Pauling and R. B. Corey. The Pleated Sheet, a New Layer Configuration of Polypeptide Chains. *Proceedings of the National Academy of Sciences*, 37(5):251–256, 1951.

[156] L. Pauling, R. B. Corey, and H. R. Branson. The Structure of Proteins. *Proceedings of the National Academy of Sciences*, 37(4):205–211, 1951.

[157] B. Pierce and Z. Weng. ZRANK: Reranking Protein Docking Predictions With an Optimized Energy Function. *PROTEINS: Structure, Function, and Bioinformatics*, 67:1078–1086, 2007.

[158] S. Pion and M. Teillaud. 2D circular kernel. In *CGAL User and Reference Manual*. 3.2 and 3.3 edition. `http://www.cgal.org/Manual/3.3/doc_html/cgal_manual/packages.html#Pkg:CircularKernel2`.

[159] A. Poupon. Voronoï and Voronoï-related tessellations in studies of protein structure and interaction. *Current Opinion in Structural Biology*, 14(2):233–241, 2004.

[160] R. Prasad Bahadur, P. Chakrabarti, F. Rodier, and J. Janin. A Dissection of Specific and Non-specific Protein–Protein Interfaces. *Journal of Molecular Biology*, 336(4):943–955, 2004.

[161] E. Rafalin, D. Souvaine, and I. Streinu. Topological Sweep in Degenerate Cases. *Lecture Notes In Computer Science; Vol. 2409*, pages 155–165, 2002.

[162] G. Ramachandran and V. Sasisekharan. Conformation of polypeptides and proteins. *Adv Protein Chem*, 23:283–438, 1968.

[163] V. Ranjan and A. Fournier. Volume models for volumetric data. *Computer*, 27(7):28–36, 1994.

[164] D. Robotics, Halperin, L. Kavraki, and J. Latombe. *Handbook of Discrete and Computational Geometry Second Edition, J. O'Rourke, JE Goodman editors*. Chapman & Hall/CRC, 2004.

[165] G. D. Rose. No assembly required. *The Sciences*, 36:26–31, 1996.

[166] S. Rusinkiewicz and M. Levoy. QSplat: a multiresolution point rendering system for large meshes. *Proc. 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 343–352, 2000.

[167] S. Sacquin-Mora1, A. Carbone, and R. Lavery. Identification of Protein Interaction Partners and Protein-Protein Interaction Sites. *Journal of Molecular Biology*, 382(5), 2008.

[168] R. Samudrala and J. Moult. An all-atom distance-dependent conditional probability discriminatory function for protein structure prediction. *Journal of Molecular Biology*, 275(5):895–916, 1998.

[169] F. Sanger. The arrangement of amino acids in proteins. *Advances in Protein Chemistry*, 7:1–67, 1952.

[170] M. Sanner, A. J. Olson, and J. Spehner. Reduced surface: an efficient way to compute molecular surfaces. *Biopolymers*, 38:305–320, 1996.

[171] G. Saporta and J. Bouroche. *L'Analyse des Données*. Presses Universitaires de France, Paris, 1985.

[172] G. Schröder, A. Brunger, and M. Levitt. Combining efficient conformational sampling with a deformable elastic network model facilitates structure refinement at low resolution. *Structure*, 15:1630–1641, 2007.

[173] O. Schwarzkopf. The extensible drawing editor Ipe. *Proc. 11th Annual Symposium on Computational Geometry*, pages 410–411, 1995.

[174] A. Schweikard, J. Adler, and J. Latombe. Motion planning in stereotaxic radiosurgery. *Robotics and Automation, IEEE Transactions on*, 9(6):764–774, 1993.

[175] M. Sela, F. White, and C. Anfinsen. Reductive Cleavage of Disulfide Bridges in Ribonuclease. *Science*, 125(3250):691–692, 1957.

[176] M. Shamos and D. Hoey. Geometric intersection problems. *Proc. 17th Annual IEEE Symposium on Foundations of Computer Science*, pages 208–215, 1976.

[177] M. Shen and A. Sali. Statistical potential for assessment and prediction of protein structures. *Protein Science*, 15(11):2507, 2006.

[178] M. Shirts and V. Pande. COMPUTING: Screen Savers of the World Unite! *Science*, 290(5498):1903–1904, 2000.

[179] J. Skolnick. In quest of an empirical potential for protein structure prediction. *Current Opinion in Structural Biology*, 16(2):166–171, 2006.

[180] V. Sobolev and M. Edelman. Modelling the quinone-B binding site of the photosystem I1 reaction center using notions of complementarity and contact-surface between atoms. *Proteins: Structure, Function and Genetics*, 21:214–225, 1995.

[181] S. Sousa, P. Fernandes, and M. Ramos. Protein-ligand docking: current status and future challenges. *Proteins*, 65(1):15–26, 2006.

[182] C. Summa and M. Levitt. Near-native structure refinement using in vacuo energy minimization. *Proceedings of the National Academy of Sciences*, 104(9):3177, 2007.

[183] C. Summa, M. Levitt, and W. DeGrado. An Atomic Environment Potential for use in Protein Structure Prediction. *Journal of Molecular Biology*, 352(4):986–1001, 2005.

[184] R. E. Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

[185] M. Totrov and R. Abagyan. The Contour-Buildup Algorithm to Calculate the Analytical Molecular Surface. *Journal of Structural Biology*, 116(1):138–143, 1996.

[186] O. Tsodikov, M. Jr., and Y. Sergeev. Novel computer program for fast exact calculation of accessible and molecular surface areas and average surface curvature. *Journal of Computational Chemistry*, 23(6):600–609, 2002.

[187] J. Vahrenhold. Line-segment intersection made in-place. *Computational Geometry: Theory and Applications*, 38(3):213–230, 2007.

[188] B. Vallone, A. Miele, P. Vecchini, E. Chiancone, and M. Brunori. Free energy of burying hydrophobic residues in the interface between protein subunits. *Proceedings of the National Academy of Sciences*, 95(11):6103–6107, 1998.

[189] A. Varshney and F. P. Brooks, Jr. Fast analytical computation of Richards's smooth molecular surface. In G. M. Nielson and D. Bergeron, editors, *IEEE Visualization '93 Proceedings*, pages 300–307, October 1993.

[190] D. Voet and J. Voet. *Biochemistry, 2nd edn., J.* Wiley & Sons Inc., New York, 1995.

[191] B. Wallner and A. Elofsson. Can correct protein models be identified? *Protein Science*, 12(5):1073–1086, 2003.

[192] J. Watson and F. Crick. Molecular structure of nucleic acids; a structure for deoxyribose nucleic acid. *Nature*, 171(4356):737–738, 1953.

[193] R. Wein, E. Fogel, B. Zukerman, and D. Halperin. 2D arrangements. In *CGAL User and Reference Manual*. 3.2 and 3.3 edition. `http://www.cgal.org/Manual/3.3/doc_html/cgal_manual/packages.html#Pkg:Arrangement2`.

[194] Worldwide Protein Data Bank. `http://www.wwpdb.org`.

[195] Z. Xiang, C. Soto, and B. Honig. Evaluating conformational free energies: The colony energy and its application to the problem of loop prediction. *Proceedings of the National Academy of Sciences*, 99(11):7432, 2002.

[196] C. K. Yap and T. Dubé. The exact computation paradigm. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, volume 4 of *Lecture Notes Series on Computing*, pages 452–492. World Scientific, Singapore, 2nd edition, 1995.

[197] M. Zacharias. Protein-protein docking with a reduced protein model accounting for side-chain flexibility. *Protein Science*, 12(6):1271–1282, 2003.

Résumé

Depuis les travaux précurseurs de Richard et al., les constructions géométriques occupent une place importante dans la description des macro-molécules et leurs assemblages. En particulier, certains complexes cellulaires liés au diagramme de Voronoï ont été utilisés pour décrire les propriétés de compacité des empilement atomiques, calculer des surfaces moléculaires, ou encore détecter des cavités à la surface des molécules. Cette thèse se positionne dans ce contexte, et après une brève introduction à la structure des protéines, détaille quatre contributions.

Premièrement, en utilisant le principe de balayage introduit par Bentley et Ottmann, cette thèse présente le premier algorithme effectif pour construire l'arrangement exact de cercles sur une sphère. De plus, en supposant que les cercles proviennent de l'intersection entre sphères, une stratégie pour calculer les listes couvrantes d'une face de l'arrangement (i.e. la liste des boules qui la recouvrent) est proposée. L'exactitude n'étant pas une fin en soi, mais plutôt une façon de rendre l'algorithmique robuste, nous montrons expérimentalement que le surcoût induit est modeste.

Deuxièmement, cette thèse développe les primitives algébriques et géométriques requises par l'algorithme de balayage afin de le rendre générique et robuste. Ces primitives sont intégrées dans une contexte plus général, à savoir le noyau CGAL pour les objets sphériques.

Troisièmement, la machinerie introduite est utilisée pour traiter un problème de biologie structurale computationelle : la sélection d'un sous-ensemble varié à partir d'un ensemble redondant de conformations de boucles. Nous proposons de résoudre ce problème de sélection en retenant les représentants qui maximisent l'aire ou le volume de la sélection. Ces questions peuvent être traitées géométriquement à l'aide d'arrangements de cercles sur une sphère. La validation est faîte sur deux fronts. D'un point de vue géométrique, nous montrons que notre approche génère des sélections dont l'aire de la surface moléculaire équivaut à celle de sélections obtenues par des stratégies classiques, mais qui sont de taille nettement inférieure. Du point de vue amarrage de protéines, nous montrons que nos sélections améliorent de manière significative les résultats obtenus à l'aide d'un algorithme manipulant des parties flexibles.

Pour finir, nous discutons les problèmes et choix d'implémentation, en les replaçant dans le contexte de la librairie CGAL.

Mots-clés: Arrangement de circles, objects courbes, noyaux géométriques, programmation générique, robustesse, modèles de Van der Waals models, sélection de conformers, amarrage flexible, surface moléculaire

Abstract

Since the early work of Richard et al., geometric constructions have been paramount for the description of macromolecules and macro-molecular assemblies. In particular, Voronoï and related constructions have been used to describe the packing properties of atoms, to compute molecular surfaces, to find cavities. This thesis falls in this realm, and after a brief introduction to protein structure, makes four contributions.

First, using the sweep line paradigm of Bentley and Ottmann, we present the first effective algorithm able to construct the exact arrangement of circles on a sphere. Moreover, assuming the circles stem from the intersection between spheres, we present a strategy to report the covering list of a face of the arrangement—that is the list of spheres covering it. Along the way, we ascertain the fact that exactness of the arrangement can be achieved with a small computational overhead.

Second, we develop the algebraic and geometric primitives required by the sweep algorithm, so as to make it generic and robust. These primitives are integrated in a broader context, namely the CGAL 3D Spherical Kernel.

Third, we use the aforementioned machinery to tackle a computational structural biology problem, namely the selection of diverse conformations from a large redundant set. We propose to solve this selection

problem by computing representatives maximizing the surface area or the volume of the selection. From a geometric standpoint, these questions can be handled resorting to arrangements of circles and spheres. The validation is carried out along two lines. On the geometric side, we show that our selections match the molecular surface area of selections output by standard strategies but using a smaller number of conformers by one and two orders of magnitude. On the docking side, we show that our selections can significantly improve the results obtained for a flexible-loop docking algorithm.

Finally, we discuss the implementation issues and the design choices, in the context of the best practices underlying the development of CGAL.

Keywords: Arrangement of circles, curved objects, geometric kernel, generic programming, robustness, Van der Waals models, conformer selection, flexible docking, molecular surface area