Thèse présentée pour obtenir le grade de

DOCTEUR DE L'ÉCOLE POLYTECHNIQUE

Spécialité : Informatique

par

Luca CASTELLI ALEARDI

Représentations compactes de structures de données géométriques

Soutenue le 12 décembre 2006 devant le jury composé de :

Cyril GAVOILLE	Université Bordeaux 1	Rapporteur
Jack SNOEYINK	University of North Carolina, Chapell Hill	Rapporteur
Ferrán HURTADO	Universitat Politècnica de Catalunya, Barcelone	
Jean-Marc STEYAERT	École Polytechnique, Palaiseau	
Olivier DEVILLERS	INRIA, Sophia-Antipolis	Directeur de thèse
Gilles SCHAEFFER	École Polytechnique, Palaiseau	Directeur de thèse

Table des matières

Table des figures

Liste des tableaux

Abstract

We consider the problem of designing compact and succint representations for geometric data structures. As opposed to raw compression issues, the focus is here on designing data structures that preserve the possibility of answering local incidence queries in constant time, while using little as memory resources as possible. One of the main contribution of this work is to present a general algorithmic framework for designing compact representations of structures as planar graphs and 3D meshes. As application we propose some solutions for compactly representing the connectivity (combinatorial information) of some classes of local planar graphs.

For planar triangulations with m faces, we propose a compact representation of the connectivity information that improves to 2.175 bits per triangle the asymptotic amount of space required and that supports navigation between adjacent triangles in constant time : this is optimal for the class of triangulations with a boundary of arbitrary sizes. For triangulations of a surface with bounded genus g, our representation is still valid and optimal, requiring $O(g \log m)$ extra bits. Such a representation can be adapted to allow local updates of the triangulation. More precisely, a triangulation with m triangles can be maintained under vertex insertions in $O(1)$ amortized time and under vertex deletions/edge flips in $O(\log^2 m)$ amortized time. We also propose the first optimal representations for 3-connected planar graphs and triangulations, which are the standard classes of graphs underlying meshes with spherical topology. Optimal means that these representations asymptotically match the respective entropy of the two classes, namely 2 bits per edge for 3-connected planar graphs, and 1.62 bits per triangle (or equivalently 3.24 bits per vertex) for triangulations.

These structures allow constant time access to vertex specific data, like coordinates, but our work does not address the compression of this geometric information.

Key-Words : graph encoding, succinct and compact representations, triangulations, geometric data structure, planar graphs

Résumé

Nous considérons le problème de concevoir des représentations compactes ou succinctes de structures de données géométriques. Dans ce cadre, en plus des questions de simple compression, l'attention est portée sur l'étude de structures de données nécessitant une petite quantité de ressources mémoire et permettant de répondre à des requêtes locales en temps $O(1)$. L'une des contributions de cette thèse consiste à proposer un cadre algorithmique général pour la conception de représentations compactes de structures telles que les graphes planaires et les maillages surfaciques. Comme application nous présentons différentes structures spécialement conçues pour représenter de manière compacte la connectivité (ou information combinatoire) de certaines classes de graphes localement planaires. Pour le cas des triangulations planaires à m faces, nous proposons une représentation compacte de l'information combinatoire nécessitant asymptotiquement 2.175 bits par triangle pour le coût en espace et qui permet la navigation entre triangles adjacents, ainsi que d'autres requêtes locales d'incidence entre sommets, en temps constant : cette structure est ainsi optimale pour la classe des triangulations ayant un bord de taille arbitraire. Une telle représentation reste valide et optimale dans le cas de triangulations d'une surface de genre g borné : $O(g \lg m)$ bits supplémentaires sont alors nécessaires. Cette représentation est bien adaptée pour faire une mise à jour locale efficace de la triangulation. Plus précisément, il est possible d'effectuer des mises à jour en temps $O(1)$ amorti après insertion de sommets, et en temps $O(\log^2 m)$ amorti après suppression de sommets et flip d'arêtes. Et en ce qui concerne les triangulations et les graphes planaires 3-connexes, correspondant aux maillages triangulaires et polygonaux homéomorphes à une sphère, nous proposons les premières représentations succinctes optimales : elles atteignent l'entropie respective des deux classes, 2 bits par arête pour les graphes 3-connexes, et 1.62 bits par triangle (ou 3.24 bits par sommet) pour les triangulations.

Ces structures permettent aussi l'accès en temps $O(1)$ aux informations associées aux sommets, notamment leurs coordonnées. Cependant nous ne traitons pas ici la compression de cette information géométrique.

Mots-clés : codage de graphes, représentations succinctes et compactes, triangulations, structures de données géométriques, graphes planaires.

Chapitre 1

Introduction

1.1 Représentations d'objets géométriques

L'une des questions centrales de l'informatique concerne le traitement et la manipulation de l'information. Cela peut se concrétiser sous différentes formes, suivant la classe des données concernées et la typologie des manipulations algorithmiques demandées. Codages et représentations *compressées*, représentations *explicites*, représentations *implicites* et enfin représentations *succinctes* et *compactes*, constituent différentes solutions possibles visant à décrire les objets que l'on veut manipuler.

Les travaux présentés dans cette thèse se situent dans le domaine de la géométrie algorithmique et de l'algorithmique des graphes : ainsi il est naturel de s'intéresser à des objets fondamentaux tels que les maillages, les graphes planaires, les cartes et les arbres. En particulier, les maillages sont l'un des objets plus diffusés et manipulés dans plusieurs domaines et applications récentes de l'informatique : le graphisme, la simulation, l'approximation et reconstruction d'objets 3D,... Ainsi la large diffusion des maillages et surtout la croissante complexité de l'information qui y est contenue, ont motivé les recherches récentes proposant de nouvelles représentations qui puissent tirer profit de la nature structurée de ces objets.

De nombreux efforts ont été faits dans le domaine de la compression de maillages. Dans ce cadre la plupart des travaux ont proposé des algorithmes visant à représenter le maillage avec un nombre minimum de bits : cela a surtout bénéficié aux applications nécessitant la transmission sur le réseaux ou le stockage sur disque de maillages de grosses dimensions. Cependant ces méthodes ne fournissent aucune aide aux algorithmes nécessitant de manipuler en temps réel de tels objets : l'information globale d'un maillage (par exemple la forme d'un objet 3D) n'est accessible qu'à la fin de la phase de décodage (par exemple à la fin de la transmission sur le réseau).

Dans un autre cadre, un certain nombre de travaux ont porté sur l'obtention de structures explicites (basées sur des pointeurs), utilisables dans la mémoire vive, fournissant des outils pratiques pour la manipulation des maillages : de différentes manières, ces travaux ont cherché à tirer profit de la structure géométrique et combinatoire (de la carte sous-jacente) de ces objets. Rappelons néanmoins que ces stratégies ne suffisent pas à manipuler efficacement les objets géométriques lorsque leur dimension devient trop grande : en générale ces représentations sont assez gourmandes en ressources mémoire et le traitement d'objets de très grande taille affecte de manière significative leur efficacité pratique (la mémoire principale ne suffisant plus à stocker entièrement l'objet, il devient nécessaire d'avoir recours à des opérations de lecture/écriture sur mémoire externe).

Toutes ces questions ont conduit au développement d'algorithmes et de structures de données visant d'une part à minimiser les ressources mémoires utilisées, et d'autre part à permettre une accessibilité facile à la description des objets. Les solutions développées ont été nombreuses et différentes. Certains travaux ([64] [66] [65]) ont proposé des stratégies pratiques visant à coder et décrire de manière incrémentale les maillages afin de minimiser les besoins en mémoire principale (ce qui a permis dans la pratique de gérer des modèles 3D de très grosses dimensions). D'un point de vue plutôt théorique, d'autres travaux ([14], [74], [94]) ont proposé des représentations *succinctes* ou *compactes* capables de fournir un accès aux données en temps constant, tout en nécessitant une quantité de ressources mémoire proche de l'optimal. C'est dans ce dernier cadre algorithmique que se situent nos travaux.

1.2 Représentations et structures succinctes

Structures explicites classiques

Le problème principal concernant les structures de données classiques basées sur des représentations explicites concerne leur redondance et leur utilisation de ressource mémoire largement supérieur à la quantité d'information qui est effectivement stockée. Ce phénomène se produit déjà dans le cas de structures simples basées sur des pointeurs, telles que par exemple les arbres binaires ou les listes d'adjacences : l'inefficacité de leur utilisation des ressources mémoire se manifeste encore plus lorsqu'on traite des données structurées, comme dans le cas des objets manipulés par les algorithmes géométriques.

Considérons comme exemple le cas des triangulations planaires, l'un des objets fondamentaux en géométrie algorithmique. L'une des représentations les plus courantes conçues pour être utilisées dans la mémoire principale est basée sur les incidences faces/sommets : chaque triangle est représenté par 3 références à ses sommets incidents et 3 autres références à ses faces voisines ; en plus, chaque sommet contient une référence à l'un des triangles incidents. Ainsi, dans le cas d'une machine utilisant des pointeurs sur 32 bits, la représentation de la connectivité d'une triangulation nous coûte plusieurs centaines de bits par triangle : ce qui est largement supérieur aux 2.17 bits par triangle qu'on pourrait espérer atteindre (dans le cas triangulaire planaire), en suivant des arguments de comptage et de théorie de l'information (ces sujets seront traités aux chapitres 2 et 3).

Le problème d'obtenir une représentation plus compacte devient encore plus difficile lorsqu'on a besoin de représentations dynamiques : la conception de structures qui sont censées augmenter de taille demande la réservation et donc le gaspillage d'une considérable quantité de mémoire, souvent de taille linéaire par rapport à la taille de l'objet. Une motivation assez naturelle à la base de l'utilisation et de la conception de structures explicites réside cependant dans leur facilité à gérer de manière efficace l'accès aux données, aussi bien en termes statiques (recherche et navigation dans la structure), qu'au niveau dynamique (possibilité d'insertion et suppression de nouvelles données).

Structures plus compactes

On peut alors se demander s'il existe des structures de données capables de gérer certaines opérations d'accès ou de modifications des données, tout en utilisant le minimum possible de mémoire. Il est alors primordial d'établir ce que l'on entend par *quantité minimale* de mémoire. Dans le cas de structures de données simples et non structurées, telles que les tableaux statiques usuels, l'espace mémoire minimal est celui occupé par les données elles-mêmes. Mais dans le cas de structures plus complexes, telles que les arbres, les graphes ou les maillages, en plus des données *externes* (par exemple les coordonnées des sommets), il faut aussi tenir compte de l'information provenant de la structure combinatoire de l'objet à représenter. Et voici que la combinatoire énumérative nous vient en aide : il est facile de se rendre compte qu'en calculant le nombre d'objets de taille n d'une certaine classe, la quantité de mémoire minimale nécessaire pour représenter chaque objet est donnée par le logarithme en base 2 de ce nombre. Intuitivement ce logarithme (ce qu'on appelle *entropie*) nous fourni le nombre minimal de bits nécessaires en moyenne pour distinguer un objet particulier parmi tous ceux de la même taille. Ainsi, une fois qu'on sait énumérer une certaine classe d'objets, un repérage optimal (ou plus précisément *codage* optimal) consiste à utiliser son numéro dans l'ordre d'énumération. Malheureusement le fait de savoir compter ne constitue pas en soi une manière de coder efficacement un objet : tel était le cas, par exemple, de certaines classes de cartes planaires (triangulations et cartes 3-connexes), pour lesquelles le problème d'énumération avait été résolu il y a longtemps ([126] et [127]) mais des algorithmes de codage optimal (de complexité linéaire) n'ont été proposés que très récemment (ces sujets seront traités de manière plus détaillée au chapitres 2 et 3). Et même s'il possible de coder les objets d'une certaine classe avec le nombre minimal de bits, cela ne constitue pas un gros avantage du point de vue de structures qui doivent servir à représenter un objet dans la mémoire principale en permettant l'accès et la manipulation des données. Par exemple, si on dispose juste d'une version *compressée*, même non optimale, d'une carte planaire le fait de trouver certaines informations (le degré d'un sommet, les adjacences entres faces) ou de maintenir la structure après modification locale (insertion d'un sommet, flip d'arête) n'est pas chose facile : en général il est nécessaire de décompresser l'objet globalement.

Un exemple : les arbres binaires. Il nous parait utile d'examiner un exemple, les arbres binaires à n nœuds[1] : on peut d'abord supposer qu'il n'y a pas de données externes associées aux nœuds, ainsi le but est ici de représenter sa structure combinatoire. Si on nous demande quel est le nombre minimal de bits qu'on peut espérer utiliser, il est facile de donner une réponse : en effet les arbres binaires enracinés ayant n nœuds sont énumérés par le n-ième nombre de Catalan $\frac{1}{n+1}\binom{2n}{n}$. En prenant le logarithme en base 2 de cette quantité, et en utilisant la formule d'approximation de Stirling pour factorielle $n!$, on arrive facilement à la quantité $2n - \Theta(\lg n)$, qui est le nombre minimal de bits nécessaires pour coder un tel objet. Si let but est d'obtenir une version compressée d'un arbre binaire (ne supportant aucune opération d'accès ou modification), la solution est donnée par un codage optimal réalisé par les mots de parenthèses équilibrés (voir section 2.1). Cependant il n'est pas du tout évident dans cette représentation de pouvoir accéder, par exemple, au fils gauche ou au fils droit d'un nœud donné, sans lire d'abord une grosse partie du mot de parenthèses.

Travaux fondateurs. Les remarques précédentes ont motivé de nombreuses recherches dans la communauté de l'algorithmique et des structures de données. Les premiers efforts datent du début des années 90 : en particulier les papiers de Jacobson [74] et Munro et Raman [96] peuvent être considérés comme les travaux fondateurs du domaine des représentations *succinctes* ou *compactes*.

La principale contribution a été de montrer que pour certaines classes d'objets, combinatoirement simples mais très largement diffusés en informatique, il était possible d'atteindre les deux finalités mentionnées ci-dessus : avoir une structure qui utilise le moins de mémoire possible, et qui permette en même temps un accès efficace aux données stockées. Ainsi, dans le cas des arbres binaires, il a été possible d'enrichir le codage usuel à base de mots de parenthèses, afin de permettre de répondre à certaines requêtes en temps constant, même en gardant asymptotiquement à $2n$ bits la quantité de mémoire utilisée. Les travaux existants concernent principalement le cas des arbres, statiques et dynamiques. Peu de résultats étaient connus pour

[1]Une présentation plus détaillée des remarques de cette section sera donnée au chapitre 3

des structures plus complexes : dans le cas des triangulations statiques la meilleure représentation nécessitait 7 bits par sommet (3.5 bits par face)[56].

1.3 Nos contributions

Un schéma algorithmique général

Dans cette thèse nous considérons en particulier des structures de données géométriques telles que les maillages triangulaires et polygonaux et nous proposons un schéma général pour construire des représentations succinctes de ces objets. Nos travaux constituent, pour l'instant au moins d'un point de vue théorique, une réponse à la nécessité de fournir des structures de données pouvant gérer des objets géométriques de grande taille. Si jusqu'à présent, surtout dans la communauté de la géométrie algorithmique, les efforts portaient sur l'amélioration de la complexité en temps des algorithmes, la diffusion croissante de modèles géométriques très volumineux rend nécessaire la conception de structures de données économes en espace.

Esquisse du schéma. Le problème consiste donc à concevoir des représentations d'objets géométriques, qui soient à la fois un codage compact et une structure de données, ce qui revient à :

- utiliser une quantité de ressource mémoire la plus proche possible du minimum théorique ;

- pouvoir répondre à certains types de requêtes locales de manière efficace, notamment en temps $O(1)$;

- permettre éventuellement la mise à jour locale en temps (poly)logarithmique.

Ainsi, l'une des contributions de cette thèse consiste à décrire un schéma algorithmique général pour construire de telles structures. Notre stratégie s'inspire des travaux précédents traitant le cas de structures plus simples (mots bien parenthésés et arbres), et peut se résumer en les étapes suivantes :

- construire un catalogue exhaustif d'objets de très petite taille (micro objets) ;

- décomposer l'objet initial en micro objets : notre objet devient alors un graphe d'adjacence entre des nœuds, chaque nœud contenant un pointeur vers le micro-objet correspondant dans le catalogue ;

- regrouper les nœuds par proximité (mini-objets) afin de pouvoir utiliser des pointeurs locaux plus économiques pour décrire les relations de voisinage dans le graphe des micro-objets.

Nous pouvons montrer que le coût de représentation du graphes d'incidence entre les mini/micro objets est sous-linéaire comme le coût du catalogue. Il ne reste plus que le coût des pointeurs vers le catalogue : c'est la contribution de ces pointeurs qui donne le terme asymptotiquement dominant de la complexité en mémoire de la structure. Intuitivement, le caractère succinct ou compact provient du fait que les micro objets apparaissant plusieurs fois dans la décomposition de l'objet initial ne sont représentés explicitement qu'une seule fois (dans le catalogue exhaustif).

Applications de notre schéma

Triangulations avec bord. Le chapitre 5 constitue la première application de notre schéma général et présente une représentation compacte pour les triangulations planaires. Plus précisément, le cas des triangulations planaires ayant n sommets et m faces (éventuellement avec bord), nous proposons une représentation compacte qui nécessite asymptotiquement 2.17 bits par face et permet la navigation entre triangles en temps constant. Notre codage améliore les bornes supérieures connues jusqu'à présent et est optimal pour la classe des triangulations planaires avec bord. Cette représentation est encore valide dans le cas de triangulations de genre supérieur g, nécessitant alors asymptotiquement $36(g-1)\log m$ bits auxiliaires. Il est aussi possible d'enrichir notre structure afin de fournir une implantation efficace pour les test d'adjacence entre sommets en temps constant. Le cas de triangulations ayant un nombre borné de bords peut aussi se traiter.

Cas dynamique. Au chapitre 6 nous avons adapté notre représentation pour le cas des triangulations dynamiques : il est donc possible de modifier localement la triangulation en temps poly-logarithmique. Ainsi l'insertion de sommets de degré 3 peut s'effecteur en temps $O(1)$ amorti, alors que d'autres opérations de mise à jour (suppression de sommets et bascule d'arêtes) nécessitent $O(\log^2 m)$ amorti. Ce résultat constitue la première structure dynamique succincte pour des objets géométriques : les travaux précédents concernent le cas de structures plus simples (mots de parenthèses, arbres binaires, dictionnaires), ou bien présentent des représentations non optimales dont les ressources mémoire sont difficiles à caractériser explicitement.

Optimalité pour triangulations et graphes 3-connexes. Au chapitre 7 nous proposons les premières représentations succinctes optimales pour les maillages triangulaires et polygonaux de typologie sphérique : ainsi nos représentations nécessitent asymptotiquement 3.24 bits par sommet pour les triangulations planaires et 2 bits par arête pour les graphes planaires 3-connexes . La voie que nous avons suivie consiste à exploiter les bijections ([99],[42]) existantes entre ces deux classes de graphes planaires et certaines classes spéciales d'arbres bourgeonnants. Ainsi notre schéma général, couplé avec de nouvelles stratégies de décomposition du graphe initial basées sur des arbres couvrants ont conduit à des représentations asymptotiquement optimales : ici on utilise largement le fait que le catalogue des micro morceaux ne doit pas forcement contenir des objets appartenant à la même classe que l'objet initial. Cette propriété constitue un point crucial de notre schéma, et l'une des différences principales avec les travaux précédents, qui ont conduit à des représentations qui sont juste compactes et non succinctes (pour les graphes).

Remarques supplémentaires

Modèle de calcul : algorithmes trans-dichotomiques. Comme dans la totalité des travaux existants concernant les structures de données succinctes, nous adoptons comme modèle de calcul une machine *word-RAM* avec des mots de taille $\Theta(\log n)$, où n est le paramètre de taille de l'entrée (ce modèle sera considéré avec plus de précision à la section 2.2.2). On fait souvent référence aux algorithmes *trans-dichotomiques* lorsqu'on veut indiquer les stratégies basées sur ce modèle de calcul : comme souvent mentionné dans la littérature, ces algorithmes relèvent d'un intérêt plutôt théorique. S'il est vrai que les structures compactes et succinctes rentrent aussi dans ce cadre, il est néanmoins à souligner que les représentations proposées dans cette thèse ont été source d'inspiration pour une structure de données compacte et implantable dans la pratique que nous avons développée pour le cas des triangulations planaires : ce travail constitue un premier effort vers l'étude du compromis entre optimalité des ressources mémoire et temps de requête et navigation pour des structures de données géométriques (chapitre 8).

Attache des données géométriques. Suivant l'approche adoptée par les travaux existants (codages compacts de graphes [94, 31, 14], algorithmes de compression de maillages [120, 109]), nos représentations sont orientées vers le codage de l'information de *connectivité* des objets géométriques. Ce type d'information concerne la structure combinatoire de l'objet : par exemple dans le cas des maillages, la connectivité décrit les relations d'incidences entre sommets, arêtes et faces. Il existe aussi un autre type d'information, géométrique, qui décrit la position des sommets d'un maillage dans l'espace. Il est facile de vérifier que la connectivité constitue une partie très importante de l'information globale d'un objet 3D : la géométrie nécessite de quelques dizaines de bits par sommet, alors que pour décrire la structure combinatoire il faut souvent plusieurs centaines de bits par sommet, pour des maillages de taille assez grande (voir chapitre 3). Bien que dans cette thèse nous ne considérons pas le problème de coder l'information géométrique, nos représentations sont compatibles avec les schémas usuels de compression de ce type d'information (par exemple les méthodes de prédiction proposées par les stratégies de compression de maillages). En particulier, la description hiérarchique des maillages (à plusieurs niveaux) à la base de notre schéma suggère d'adopter un repère géométrique relatif aux micro régions, permettant ainsi d'exprimer la position des sommets avec des coordonnées locales moins coûteuses en mémoire.

Première partie

Préliminaires

Chapitre 2

Graphes et algorithmique

2.1 Quelques notions sur les graphes, cartes et maillages

Le but de cette section est de fournir des définitions et des notions de base concernant les graphes, les cartes et les maillages, sans vouloir présenter une étude détaillée et extrêmement rigoureuse de ces trois types d'objets. Nous renvoyons les lecteurs désireux d'approfondir leurs différents aspects et propriétés à d'autres ouvrages, références dans les domaines de la géométrie algorithmique ([16, 34]), de la combinatoire des cartes ([113]) et de la théorie des graphes ([35]).

Ici nous essayons de fournir des notions précises et de clarifier les différences et points communs entre graphes, cartes et maillages, compte tenu des thèmes et domaines divers abordés dans cette thèse. C'est dans ce cadre que nous nous plaçons dans cette thèse, où nous proposons des algorithmes et des structures de données pouvant s'appliquer indistinctement aux trois types d'objets mentionnés ci-dessus (les cartes et les maillages, au moins du point de vue combinatoire et en faisant abstraction des propriétés géométriques, peuvent représenter différents aspects du même objet).

Graphes

Définition 1. *Un* graphe $G = (V, E)$ *est une paire constituée de :*
- *un ensemble de* sommets $V = (v_1, \dots, v_n)$
- *une famille* $E = (e_1, \dots, e_m)$ *d'éléments, dites* arêtes*, du produit cartésien* $V \times V = \{(u, v) | u \in X, v \in X\}$

Une *boucle* est une arête dont les extrémités coïncident ($u = v$). Si une arête apparaît plusieurs fois dans E alors il s'agit d'une *arête multiple*. Un graphe est dit *simple* s'il ne contient pas d'arêtes multiples ni de boucles. Un graphe est k*-connexe* s'il faut supprimer au moins k sommets pour qu'il ne soit plus connexe.

Cartes Bien que déjà intéressants d'un point de vue algorithmique, les graphes ne suffisent pas à capturer les propriétés d'objets communs en géométrie : par exemple, dans le cas des maillages, il est usuel de parler de "faces" (notion qui n'apparaît pas dans la définition donnée précédemment). Dans ce cadre il est convenable d'introduire une structure mathématique supplémentaire, à l'aide de la définition suivante (de nature topologique, et illustrée par la figure 2.1) :

Définition 2. *Une* carte $\mathcal{C}$ *est un plongement cellulaire d'un graphe (*dessin*) dans une surface*[1] $\mathcal{S}$*, considéré à homéomorphisme près. De plus le plongement satisfait les conditions suivantes :*

- les sommets du graphe sont représentés par des points ;

- les arêtes sont représentées par des arcs de courbes ne se coupant pas en dehors des sommets ;

- le complément $\mathcal{S} \setminus \mathcal{C}$ *est une union disjointe de régions appelées* faces *(ne contenant ni de sommets ni d'arêtes) qui sont homéomorphes au disque ouvert de* R^2*.*

L'*enracinement* d'une carte consiste à marquer et orienter une arête de telle sorte qu'elle ait la face infinie (ou externe) à sa droite : une telle carte est dite *enracinée*. Les *relations d'incidences* décrivent la manière dont les éléments d'une carte sont reliés : un sommet v et une arête e sont incident lorsque v est une extrémité de e ; une face f est incidente à une arête e (resp. un sommet v) si e (resp. v) appartient au bord de f. D'un point de vue combinatoire, une carte est entièrement donnée par les relations d'incidence sommets/aretes/faces et l'arrangement cyclique des arêtes autour des sommets ou des faces. Ainsi, dans le cas des graphes, la seule information disponible concerne les incidences arêtes / sommets ; alors que c'est avec la notion de carte qu'on dispose de l'information topologique concernant, par exemple, les incidences faces / sommets.

Triangulations et maillages Un *simplexe* de dimension k est un polytope constitué de l'enveloppe convexe de $k + 1$ points indépendants dans l'espace de dimesion d, pour $k \leq d$.

Définition 3. *Un* complexe simplicial C *est un ensemble de simplexes vérifiant les deux conditions suivantes :*

– *toute face d'un simplexe de* C *est aussi un simplexe de* C *;*
– *l'intersection de deux simplexes est vide ou bien est constituée d'un simplexe de dimension inférieure (face commune de dimension maximale).*

[1]Dorénavant nous utiliserons le terme *surface* pour désigner toute variété compacte orientable de dimension 2.

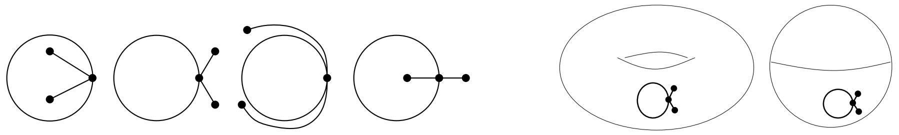


FIG. 2.1 – Les premières images montrent quatre plongements d'un même graphe planaire, définissant deux cartes différentes. Les trois premiers plongements, ayant des faces de degré 1 et 5, représentent la même carte : les deux premiers diffèrent seulement pour le choix de la face infinie, tandis que le troisième peut s'obtenir du deuxième par déformation continue. Le quatrième dessin correspond à une carte différente ayant deux faces de degré 3. Les deux dernières images montrent deux dessins du graphe sur un tore et une sphère respectivement : seulement le deuxième définit une carte, puisque sur le tore la face externe n'est pas homéomorphe à un disque ouvert.

C est un *k-complexe* si la dimension maximale des simplexes qui le composent est k, et dans ce cas il est *pur* lorsque toute face est l'une des faces d'un k-simplexe de C. Le *l-squelette* d'un complexe C est le sous-complexe constitué par ses faces de dimension au plus l : le 1-squelette d'un complexe est donc isomorphe à un graphe ayant pour nœuds et arêtes respectivement les faces de dimension 0 et 1 de C.

Définition 4. *Un* maillage *est une structure géométrique formée d'une carte $G = (V, E)$ munie d'un ensemble de coordonnées de sommets décrivant le plongement de G dans l'espace. Un maillage est* manifold *(ou variété) si toute arête de G est incidente à deux faces exactement et les faces incidentes à tout sommet forment un cône simple. S'il existe un ou plusieurs cycles disjoints d'arêtes incidentes à une seule face, le maillage est dit* manifold avec bords*.*

L'information combinatoire décrite par les relations d'incidence entre sommets, arêtes et faces d'un maillage sera aussi appelée *information de connectivité*.

Cartes, graphes, 3-connexité. Le lien entre graphes et cartes se révèle encore plus étroit lorsque des contraintes supplémentaires concernant la connexité du graphe sont imposées. Plus précisément, les triangulations planes 3-connexes (ou autrement dit, les triangulations planes sans arêtes multiples) sont équivalentes aux *graphes plans maximaux* (et aussi aux 2-triangulations de la sphère), comme exprimé par le théorème suivant :

Théorème 5 (Whitney, 1933)**.** *Un graphe planaire 3-connexe admet un unique plongement planaire à homéomorphisme et inversion de la sphère près.*

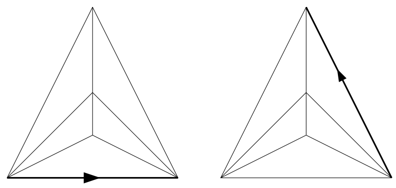


Fig. 2.2 – Les deux premières images montrent le plongement d'une triangulation planaire sans arête multiple : en tant que graphe 3-connexe le plongement est essentiellement unique. En tant que cartes enracinés, les deux triangulations sont différentes.

Mais encore plus significatif dans ce contexte se révèle le théorème de Steinitz (1916) unifiant des propriétés des graphes, des complexes de dimension 2 et des surfaces discrètes qui, dans sa formulation moderne en théorie des graphes, s'exprime par :

Théorème 6. *Un graphe G est isomorphe au* 1*-squelette d'un polyhèdre convexe de dimension* 3 *(variété* 2*-dimensionnelle) si et seulement si G est planaire et* 3*-connexe.*

Planarité locale et relation d'Euler Une fois introduits les concepts de graphe (planaire), carte et maillage, il ne reste qu'à mentionner une propriété fondamentale, introduite par Euler, caractérisant ces trois types d'objets.

Cette relation, admettant une preuve simple dans le cas de surfaces de dimension 2, exprime essentiellement une dépendance linéaire entre le nombre de sommets, arêtes et faces d'un complexe ou d'une carte (n'oublions pas que le 1-squelette d'un 2-complexe est un graphe planaire).

Il en existe plusieurs formulations, équivalentes, exprimables en termes de 2-complexes, polyèdres[2] et cartes.

Théorème 7. *Pour tout polyèdre de genre g ayant n sommets, e arêtes et f faces nous avons :*

$$n - e + f = 2 - 2g$$

Corollaire 8. *Soit G une carte planaire ayant c composantes connexes et f' faces de dimension* 2*, dont le bord est constitué par deux arêtes (où n, e et f sont définis comme ci-dessus), alors*

$$e \leq 3n - 3 - 3c + f'$$

[2]Un *polyèdre* est un maillage *manifold* sans bords, son 1-squelette définissant ainsi une carte de genre g (g étant le genre de la surface sur laquelle la carte est plongée).

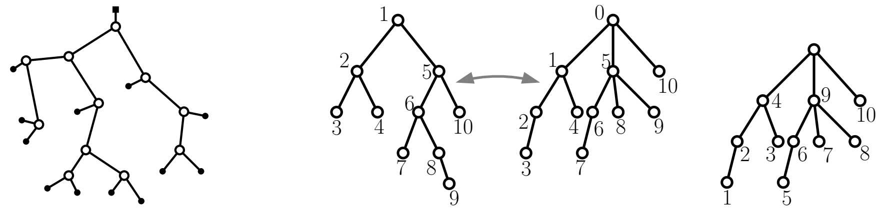


FIG. 2.3 – Quelques exemples d'arbres et leur codage. La première image montre un arbre binaire (plein) ayant 11 nœuds internes, 11 + 2 feuilles et dont le code préfixe de longueur 2×11 est : 1101001011001001011000. La bijection entre arbres binaires (dont les nœuds ont degré au plus 3) et les arbres ordonnés est aussi illustrée (les nœuds sont numérotés selon l'ordre préfixe). La dernière image montre un arbre ordonné, fourni de l'ordre postfixe sur ses sommets, et ayant comme code de Dyck (mot de parenthèses équilibrées) : (((())())((())()())()).

Dans le cas particulier de graphes (ou cartes) dont les faces ont degré au moins 3, la dernière relation garantit notamment que le nombre d'arêtes est linéairement borné par le nombre de sommets : cette remarque, ainsi que la relations d'Euler, seront largement utilisées dans le reste de cette thèse (voir chapitres 4 à 7).

Arbres

Un *arbre* est par définition un graphe connexe sans cycles. Ou encore, suivant la terminologie des cartes, un *arbre plan* est une carte planaire ayant une seule face, dite externe ou infinie. En d'autres termes, les arbres plans diffèrent des arbres classiquement dits *ordonnés* par le seul fait qu'il ne sont pas enracinés.

Un arbre *binaire* est un arbre plan tel que tout nœud a 3 voisins (et donc deux fils, s'il est enraciné). Parfois on parlera aussi d'arbres binaire lorsque chaque nœud a au plus un degré 3. Un nœud est une *feuille* si son degré est 1, autrement il s'agit d'un nœud *interne*. Les arêtes incidentes au feuilles sont aussi appelées *bourgeons*, tandis que les restantes sont dites *arêtes internes*.

Avant de conclure cette section, nous ne pouvons point ne pas mentionner quelques propriétés célèbres concernant l'énumération des arbres plans et leur codage (d'autant plus qu'elles seront souvent citées et utilisées dans cette thèse).

Lemme 9. *Les propriétés suivantes sont vérifiées (voir figure 2.3) :*

- Il existe une bijection entre les arbres plans (arbres ordonnés) à $n+1$ arêtes et les arbres binaires (pleins) à n nœuds internes ;

- les arbres plans à $n+1$ arêtes sont optimalement[3] *codés par les mots de Dyck de longueur $2n$, et énumérés par le n-ème nombre de Catalan $\frac{1}{n+1}\binom{2n}{n}$;*

- les arbres binaires (pleins) à n nœuds internes sont optimalement codés par un code préfixe de longueur $2n+1$, et également énumérés par le n-ème nombre de Catalan.

Un *mot de Dyck* est un mot binaire de longueur $2n$ que l'on peut obtenir en effectuant un parcours du contour d'un arbre plan enraciné (à $n+1$ arêtes) de la manière suivante (le parcours suit la face infinie en sens trigonométrique, en partant de la racine) : chaque fois qu'une arête (non racine) est visitée pour la première fois l'on écrit un '1', tandis que lors de la deuxième visite d'une arête on écrit[4] '0'.

Étant donné un arbre binaire à n nœuds internes, son *code préfixe* est un mot binaire de longueur $2n+1$ que l'on peut obtenir, lors du parcours du contour de l'arbre, en écrivant un '1' lorsque un nœud (non racine) est visité pour le première fois, et un '0' lorsque l'on rencontre une feuille.

2.2 Complexité en espace et en temps

2.2.1 Notations asymptotiques

Vu l'utilisation intensive faite dans cette thèse d'une classe de notations décrivant les comportements asymptotiques, nous estimons utile de fournir quelques définitions concernant ces notions, si communes en informatique théorique. Il est commun de dire, étant données deux fonctions f et g de la variable réelle, que "f tend asymptotiquement à g" lorsque $\lim_{n\to\infty}\frac{f(n)}{g(n)}=1$, ce qui est dénoté par $f(n)\sim g(n)$. Certains symboles ont été introduits afin de comparer facilement l'ordre de grandeur de fonctions différentes, pouvant se définir de la manière suivante :

– $f(n)=O(g(n))$ s'ils existent deux constantes positives c, n_0 telles que $|\, f(n)\,|\leq c\,|\, g(n)\,|$, pour tout $n\geq n_0$;
– $f(n)=o(g(n))$ si $\lim_{n\to\infty}\frac{f(n)}{g(n)}=0$;
– $f(n)=\Omega(g(n))$ s'il existe deux constantes $c, n_0>0$ telles que $|\, f(n)\,|\geq c\,|\, g(n)\,|$, pour $n>n_0$.
– $f(n)=\Theta(g(n))$ si $f(n)=O(g(n))$ et $f(n)=\Omega(g(n))$.

Cette définition du symbole Ω, introduite par D.E. Knuth, est la plus usuelle en informatique. Deux fonctions f et g sont dites avoir le même *ordre de grandeur* si $f(n)=\Theta(g(n))$: il est à rappeler que $f(n)\sim g(n)$ implique

[3] Bien que la notion de codage optimal ne soit pas encore définie, nous pouvons anticiper au lecteur que dans le cas des arbres on atteint l'optimalité lorsque le code produit a longueur asymptotiquement $2n$.

[4] Les mots (ou codes) de Dyck sont souvent connus en algorithmique sous le nom de *mots de parenthèses équilibrées* (dans ce cas on représente les '0' et '1' par des parenthèses ouvrantes "(" et fermantes ")").

$f(n) = \Theta(g(n))$, tandis que le contraire est faux en général. La notation *grand-O* permet d'écrire par exemple $\log_2 n = O(\log_{10} n)$; de manière plus générale, ces notations nous éviteront souvent de traiter des expressions compliquées et d'oublier des détails et des termes non intéressants. Nous essayerons toutefois de fournir une description la plus détaillée possible (où toutes les constantes et termes d'erreur seront fournis explicitement) lorsque notre attention sera focalisée sur une évaluation précise de la complexité en espace, là où les exigences de clarté et de lisibilité le permettront. Nous aurons souvent recours dans cette thèse à l'évaluation de la quantité $\log_2 \binom{p}{q}$: bien qu'il existe des évaluations plus fines, il nous suffira de rappeler que $\log_2 \binom{p}{q} \leq \min(q \log_2 p, p)$.

2.2.2 Modèle de machine de calcul

Tout au long de ce travail nous ferons référence à la complexité d'un algorithme en terme de nombre d'opérations élémentaires effectuées sur des mots de taille $\lg n$, tandis que la taille ou complexité en mémoire d'un objet sera mesurée en terme de bits. Nous adopterons la notation $\lg n$ pour indiquer la quantité $\lceil \log_2(n+1) \rceil$.

Machine word-RAM Suivant l'approche commune déjà adoptée par les différents travaux existants concernant les structures de données succinctes [9, 14, 11, 22, 96, 94, 104, 105, 103, 106, 97, 95, 110], nous allons considérer comme modèle de calcul une machine RAM[5] qui supporte en temps constant l'adressage direct et indirect à la mémoire et un certain ensemble standard d'opérations arithmétiques et booléennes sur des mots de taille w. Et plus précisément, le modèle de machine *word-RAM* choisi, dispose des opérations suivantes :

– addition, soustraction, multiplication et division ;

– les opérations booléennes *AND*, *OR* et *XOR* ;

– l'opération de *shift(i)*, permettant de décaler les chiffres d'un mot binaire, vers la gauche ou vers la droite, d'une certaine quantité i ($i \leq w$).

Par rapport à l'adressage mémoire, nous supposons qu'il est possible d'accéder en temps constant à tout élément d'un mot en mémoire, une fois donné un pointeur vers ce mot et un entier indiquant l'élément en question.

Une question assez importante concerne la taille w du mot machine qui est censé correspondre à la taille du problème, dans le sens suivant :

– un mot machine est assez grand pour stocker la taille m du problème, et tout paramètre en entrée ($w \geq \lg m$) ;

[5] La *machine RAM* est une simple abstraction d'un ordinateur sequentiel conventionnel, étant constituée d'un programme de contrôle et d'une mémoire à accès aléatoire : dans sa formulation originale (due à Cook et Reckhow [33]) ce modèle permettait des opérations arithmétiques et booléennes élémentaires, les mots machine pouvant contenir des nombres entiers de taille arbitraire.

– tout pointeur ou index peut se stocker sur w bits ;

En ce qui concerne l'aspect dynamique, nous permettons d'allouer de l'espace mémoire dynamiquement : et plus précisément que l'allocation d'un bloc de mémoire constitué d'un mot puisse se réaliser en temps $O(1)$.

Implanter en temps $O(1)$ des opérations non élémentaires Il est à noter qu'il n'est pas possible de calculer n'importe quelle fonction élémentaire en temps $O(1)$ dans le modèle introduit ci-dessus, au moins de manière directe : c'est le cas par exemple des fonctions $\lg i$ et $\lfloor\sqrt{i}\rfloor$ $(i < n)$, très souvent utilisées dans la plupart des algorithmes illustrés dans ce mémoire.

Dans le cas de la fonction $\lfloor\sqrt{n}\rfloor$, il est connu que la méthode de Newton permet de minimiser la complexité du calcul, nécessitant néanmoins un temps $\Theta(\lg\lg i)$ dans le cas le pire (voir [102]).

Dans le cas de la fonction logarithme, le calcul de $\lg i$ revient à trouver la position du premier bit significatif (le '1' bit de tête) dans la représentation binaire de i.

Or, il existe une manière d'implanter en temps $O(1)$ ces deux opérations à l'aide de précaculs et consultations de table (*table look-up*), en utilisant des ressources mémoires de taille négligeable [6].

Nous présentons ici cette stratégie pour le cas de la fonction $\lg i$, ce qui nous permet d'esquisser le schéma algorithmique à la base des stratégies de codage considérées dans les chapitres suivants. La solution consiste à précalculer et stocker dans un tableau A tous les résultats de $\lg i$, pour $i < 2^{\frac{\lg n}{2}} = \Theta(\sqrt{n})$, ce qui nécessite $o(n)$ bits. Maintenant il suffit de "décomposer" la représentation binaire de i en deux sous mots de taille $\frac{\lg n}{2}$: ces deux mots binaires peuvent ainsi être utilisés comme index dans le tableau A, retrouvant ainsi la position du premier bit significatif de i.

Algorithmes trans-dichotomiques Avec les hypothèses ci-dessus nous nous plaçons dans le cadre des algorithmes *trans-dichotomiques*, introduits dans [40, 41]. Fredman et Willard répondirent pour la première fois à la nécessité de disposer d'un modèle de calcul qui évite les abus potentiels d'une machine RAM avec coût unitaire sur des mots de longueur arbitraire, permettant toutefois d'implanter en temps constant certaines opérations standard de taille raisonnable, correspondant au paramètre de taille du problème à traiter. Ces considérations, de caractère plutôt théorique, visent néanmoins à tenir compte des possibilités réelles des ordinateurs qui disposent de mots de taille fixée et bornée.

Nous soulignons que tous les algorithmes exposés dans cet ouvrage pourraient dans la réalité être implantés par des ordinateurs usuels avec les performances et complexités (en espace et en mémoire) garanties théoriquement :

[6] Il est à mentionner que Brodnik [22] a montré comment implanter en temps $O(1)$ la fonction $\lg i$ à l'aide seulement d'opérations arithmétiques et booléennes standard.

aucun des calculs effectués ne recours à quelque sorte de parallélisme caché, qui peut s'obtenir par exemple à l'aide d'opérations élémentaires sur des mots ou nombres arbitrairement grands.

Cependant, l'intérêt principalement théorique des algorithmes trans--dichotomiques ainsi que des structures de données succinctes a été plusieurs fois mentionné dans la littérature et dépend plutôt d'autres remarques. D'une part les constantes cachées dans la notation O limitent les performances que l'on peut atteindre en pratique. D'autre part les termes d'erreur de la forme $O(\frac{\lg n}{\lg \lg n})$ caractérisant la presque totalité des représentations succinctes, ne sont négligeables qu'asymptotiquement, tandis que dans la pratique, ils sont susceptibles d'affecter de manière significative les besoins des algorithmes en espace mémoire.

Une réponse partielle à ces questions pourra se trouver dans le chapitre 8 où nous avons fait un premier pas vers l'étude du compromis entre ressources mémoire utilisées et temps de requête, et montré l'intérêt aussi pratique des algorithmes et structures de données conçus au cours de cette thèse.

2.3 Quelques notions de Théorie de l'information

Les notions d'*entropie* et de *mesure de l'information* sont centrales dans plusieurs domaines de l'informatique. Notre intention ici est de préciser ces notions (certainement connues par le lecteur), compte tenu du cadre adopté dans cette thèse, où nous étudions le codage et la représentation d'objets algorithmiques. Pour une présentation détaillée nous renvoyons le lecteur à d'autres références, complètes et pédagogiques (un texte classique dans ce domaine est par exemple [7] ; [6] traite spécialement l'information et sa mesure).

2.3.1 Entropie de Shannon

Il existe plusieurs définitions et formulations de la notion d'entropie : en terme de variables aléatoires, sources et transmission de message, énumération d'ensembles d'objets. De manière intuitive l'entropie sert à décrire l'information relative à une variable aléatoire X (c'est une mesure du degré d'incertitude par rapport à la valeur prise par X), ainsi qu'à établir dans quelle mesure un message peut se compresser (d'autres notions de mesure ont été introduites afin de capturer la complexité de la description d'un objet algorithmique, comme par exemple la *complexité de Kolmogorov* : dans ce cadre [87] constitue une excellente introduction à la théorie algorithmique de l'information). Nous présentons ici la version classique en terme de variables aléatoires (à une variable X on peut naturellement associer une source d'information[7], produisant des messages a_i, chacun avec probabilité p_i).

Définition 10. *Étant donnée une variable aléatoire discrète X avec probabilités $P(X = a_i) = p_i$, l'*entropie *est la fonction*

$$H(X) = -\sum p_i \log_2 p_i \tag{2.1}$$

Une autre caractérisation de l'entropie

Dans le cadre de cette thèse, où nous nous intéressons au codage d'objets géométriques ou combinatoires plutôt qu'à la compression de suites de symboles, nous préférons adopter une autre définition d'entropie.

Soit $\mathcal{C} = (\mathcal{C}_n)$ une classe d'objets, n étant un paramètre de taille (par exemple la longueur d'une suite de symboles, ou le nombre de sommets d'un arbre ou un graphe), telle que l'ensemble $\mathcal{C}_n$ des objets de taille n soit de cardinalité finie $|\mathcal{C}_n|$, pour tout n fixé. L'entropie $||\mathcal{C}_n||$ est définie par

$$||\mathcal{C}_n|| := \log_2 |\mathcal{C}_n|$$

Il est facile de se rendre compte que les deux notions sont équivalentes, lorsque tous les éléments de l'ensemble sont équiprobables.

Propriétés et unicité de l'entropie

Lemme 11. *Soit $n \geq 0$ fixé, $p_i \geq 0$ (pour tout $i = 1 \dots n$) et $\sum_{i=1}^n p_i = 1$, alors la fonction entropie $H(p_1, \dots, p_n)$ vérifie*
- H atteint son maximum lorsque $p_i = \frac{1}{n}$ $(i = 1 \dots n)$.
- $H(X, Y) = H(X) + H(Y|X)$
- $H(p_1, \dots, p_n) = H(p_1, \dots, p_n, 0)$

Le théorème suivant fournit une caractérisation précise de la fonction entropie :

Théorème 12. *Soit $H(p_1, \dots, p_n)$ une fonction définie pour tout entier n et toute valeur de $p_1, \dots, p_n$ (t.q. $p_i \geq 0$, pour tout $i = 1 \dots n$, et $\sum_{i=1}^n p_i = 1$). Si H est une fonction continue en ses arguments et satisfaisant les trois propriétés du lemme précédent, alors H s'écrit sous la forme suivante*

$$H(p_1, \dots p_n) = -\lambda \sum_{i=1}^{n} p_i \log_2 p_i \ ,$$

où λ est une constante positive.

[7]Ici pour source d'information ont entend un quelconque processus de génération de messages successifs.

2.3.2 Codage, entropie et longueur moyenne d'un code

Une fois introduits les concepts d'information et d'entropie, il nous reste à rendre explicite la notion de *codage*, qui sera largement utilisée au cours des prochains chapitres.

Considérons une fonction $D : \{0,1\}^* \longrightarrow \mathcal{C}$, ayant pour domaine l'ensemble des *mots de codes* et comme image l'ensemble $\mathcal{C}$ des messages produit par la source (ou des objets combinatoire à coder). La fonction D est appelée *décodage*, et l'expression $D(y) = x$ signifie que le mot binaire y est un code pour le message (ou objet) x. L'ensemble $D^{-1}(x) = \{y \mid D(y) = x\}$ des mots de code caractérise le *codage* D^{-1} (D^{-1} n'est pas forcement une fonction).

L'une des découvertes fondamentales de Shannon [115], est que la *longueur moyenne minimale* d'un code est donnée par l'entropie de la source :

Théorème 13 (Shannon)**.** *Soit $P(x)$ une distribution de probabilités et L la longueur moyenne minimale d'un code. Alors, en dénotant $H(P) = -\sum_x P(x) \lg P(x)$ nous avons*

$$H(P) \leq L \leq H(P) + 1 \ .$$

Ainsi on dit qu'un codage est optimal lorsque sa longueur moyenne atteint asymptotiquement l'entropie de la source. Dans notre cas, le problème de définir un *codage optimal* pour une classe d'objets donnée $\mathcal{C}$ consiste à trouver une application injective $D^{-1} : \mathcal{C} \longrightarrow \{0,1\}^*$ qui à chaque objet de taille n dans $\mathcal{C}$ fait correspondre un mot binaire (son mot de code), dont la longueur peut être bornée par asymptotiquement par $\log_2 |\mathcal{C}_n|$.

Méthodes de compression de données

La plupart des algorithmes existants concernant le codage d'objets géométriques (compression de maillage, codage de graphes) recourent souvent à des techniques classiques pour la compression de données, notamment le *codage de Huffman*[60], le *codage arithmétique* [108] ([128],[59] offrent une excellente présentation de cette méthode et de son analyse) et le *codage à base de dictionnaires* [131, 132]. Bien que cette thèse porte sur le codage et la représentation d'objets géométriques, aucun des algorithmes présentés dans les prochains chapitres ne recourt à une des méthodes mentionnées ci-dessus : en effet, les représentations succinctes faisant l'objet de ce travail constituent elles même un codage (au sens donné à la section précédente) des objets auxquels nous nous intéressons. Pour le lecteur désireux d'approfondir les propriétés des méthodes de compression les plus célèbres, nous renvoyons à d'autres textes classiques (une référence complète dans le domaine de la compression de données est par exemple [111]).

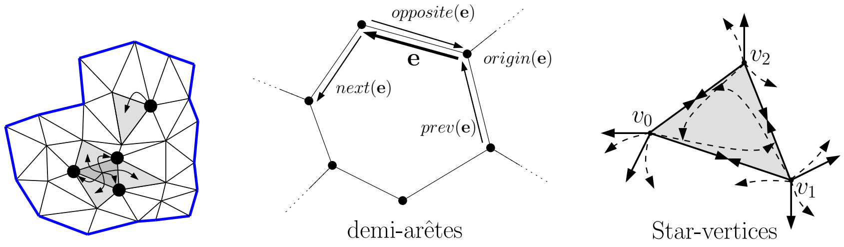


FIG. 2.4 – Structures de données explicites pour les maillages.

2.4 Structures de données classiques

Si l'on est juste intéressé par la structure combinatoire d'un graphe (sans la notion topologique de plongement), il existe des représentations explicites (utilisant des pointeurs) pouvant traiter algorithmiquement de manière simple ce type d'objet. Une représentation à base de *matrices d'adjacence* ou de *listes d'adjacences* permet en effet de tester efficacement les relations d'incidence arete/sommet : il est néanmoins à souligner que ces représentations nécessitent $\Omega(n^2)$ ou $\Omega(e \lg n)$ bits, pour un graphe ayant n sommets et e arêtes.

2.4.1 Structures de données géométriques

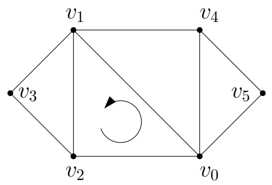
Les maillages, et en particulier les maillages triangulaires, représentent une structure de donnée de base parmi les plus utilisées et étudiées dans le domaine de la géométrie algorithmique. Il n'est pas donc étonnant que de nombreux efforts ([23, 78, 15, 8]) aient été faits pour la conception d'implantations et représentations efficaces de ce type d'objets (la figure 2.4 en montre quelques exemples).

Parfois les graphes sont munis d'une structure géométrique ou topologique supplémentaire : c'est par exemple le cas des cartes et des maillages (voir section 2.1). Il s'avère donc intéressant de concevoir des représentations spécifiques et plus performantes pour de tels objets.

Représentations explicites de maillages Dans le cadre des structures de données explicites, un certain nombre de travaux portent sur l'étude de représentations explicites par pointeurs, spécialement conçues pour tirer profit de la structure (de carte) sous-jacente à un maillage.

Parmi les plus célèbres, nous devons mentionner la représentation basée sur les *demi-arêtes* (voir [78] pour une présentation détaillée), où l'on profite de l'ordre cyclique des demi-arêtes autour des sommets pour coder la connectivité du maillage.

Plus précisément, chaque demi-arête contient une référence (ou pointeur)



sommets	degrés	voisins
v_0	4	$(v_1,3)(v_2,2)(v_5,1)(v_4,2)$
v_1	4	$(v_0,3)(v_4,1)(v_3,1)(v_2,0)$
v_2	3	$(v_0,0)(v_1,2)(v_3,0)$
v_3	2	$(v_1,1)(v_2,1)$
v_4	3	$(v_0,2)(v_5,2)(v_1,0)$
v_5	2	$(v_0,1)(v_4,0)$

FIG. 2.5 – Implantation de la représentation *Star-vertices*.

à l'un de ses sommets incidents, ainsi que des références vers la demi-arête opposée et la demi-arête qui la précède (et éventuellement celle qui la suit) dans la même face incidente ; de plus, les sommets peuvent contenir une référence vers l'une des demi-arêtes incidentes. Spécialisée au cas des triangulations, cette représentation conduit à stocker $19n$ références ou pointeurs (vu qu'il existe $2 \times 3n$ demi-arêtes dans une triangulation ayant n sommets).

Dans le cas particulier de maillages triangulaires il existe une représentation moins coûteuse [15], où les objets de base sont les faces. Chaque triangle contient des références à ses trois faces adjacentes et aux trois sommets incidents. De plus, chaque sommet contient une référence à l'une de ses faces incidentes, ce qui permet de stocker $13n$ références, pour une triangulation à n sommets.

Toutes ces structures permettent une navigation efficace dans le maillage : les requêtes d'adjacences mentionnées ci-dessus (parcours des sommets autour d'une face, parcours des voisins d'un sommet, voisinage entre deux faces, incidence entre sommets/aretes) peuvent s'effectuer en temps $O(1)$. Il reste néanmoins d'autres requêtes locales, telles que le test d'adjacence entre deux sommets, qui nécessitent un temps $O(d)$ proportionnel au degré des sommets.

Une structure plus compacte Au delà des représentations mentionnées ci-dessus qui sont d'une certaine manière redondantes, il existe d'autres structures de données plus compactes, visant à obtenir une réduction (d'un facteur constant) du nombre de références et donc de l'information à stocker.

La structure proposée dans [75], appelée *star vertices*, est concue pour représenter des maillages planaires arbitraires, mettant au centre de l'attention l'information relative aux adjacences entre sommets (voir la figure 2.5).

Plus précisément, pour chaque sommet on stocke son degré et la liste de ses demi-aretes incidentes (les voisins listés en sens trigonométrique).

La $k-$ème demi-arete d'un sommet v est codée par un couple (v_{i_k}, r_k) : où v_{i_k} est l'index du sommet adjacent, et r_k est un entier indiquant le prochain sommet u adjacent à v_{i_k} tel que v, v_{i_k} et u appartiennent à la meme face (u n'est que le r_k-ème voisin de v_{i_k}).

Pour un sommet ayant degré d il faut stocker donc $1+2\cdot d$ index (chacun sur $\lg n$ bits) et la taille globale de la structure dépend alors du degré moyen des sommets : ainsi pour une triangulation planaire ayant n sommets et environ $3n$ aretes (le degré moyen des sommets étant 6), cette représentation nécessite 13 index par sommet. Il existe aussi une version plus compacte qui ne nécessite plus que $7n$ références (pour une triangulation ayant n sommets) : mais dans ce cas l'information concernant la structure des faces n'est plus explicite et la complexité de la plupart des requêtes locales devient proportionnelle au degré des sommets.

Enfin, il est à remarquer que toute représentation explicite (utilisant des références ou vrais pointeurs), ne permet jamais de descendre au-dessous de $\Omega(n \lg n)$ bits.

2.5 Énumération (et entropie) de graphes

Lorsqu'on se place dans le domaine de la compression (ou dans celui du codage), il est parfois utile de considérer d'abord le problème de l'énumération des objets que l'on veut coder : une meilleure évaluation des performances d'un algorithme de codage, passe par exemple par le calcul de l'entropie d'une certaine classe d'objets, qui fourni une borne inférieure sur le taux de compression (longueur moyenne du code).

2.5.1 Triangulations d'un ensemble de points dans le plan

Dans le domaine de la géométrie algorithmique, un certain nombre de travaux [2, 116, 1, 112] portent sur le problème d'énumération concernant les triangulations[8] d'un ensemble de n points donnés dans le plan.

Dans ce cadre, il n'est pas connu de solution exacte pour des points en position générale, et la plupart des résultats proposent de meilleures bornes supérieures et inférieures pour le nombre de triangulations planes d'un ensemble de n points fixé. Un résultat récent de Sharir et Welzl [116] a établi une meilleure borne supérieure pour le nombre de triangulations d'un ensemble arbitraire de n points (fixés) dans le plan, qui est d'au plus $43^n \sim 2^{5.42n}$ (la meilleure borne précédente étant de 59^n, due à Santos et Seidel [112]). D'autres travaux portent sur le problème d'établir des bornes inférieures : un résultat récent [1] conduit à une borne de $\Omega(8.48^n)$, pour le nombre de triangulations d'un ensemble de n points.

Ces résultats sembleraient être en contradiction avec les travaux d'énumération de cartes planaires dus à Tutte ([126, 127]), puisque le nombre de triangulations de la sphère ayant n sommets est environ $2^{3.24n}$ (voir la section suivante).

[8]Dans ce paragraphe une triangulation est un graphe plan maximal sans croisements : toutes les arêtes peuvent se dessiner dans le plan par des segments de droite.

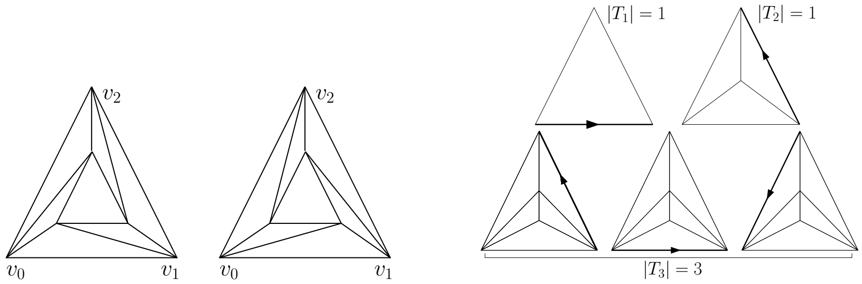


FIG. 2.6 – Énumération de triangulations. Les deux premières images montrent deux triangulations différentes d'un ensemble de points dans le plan : ces deux triangulations, du point de vue des cartes planaires, coïncident puisqu'elles sont isomorphes. Sur la droite sont dessinées les triangulations planaires ayant jusqu'à 2 sommets internes : les 3 triangulations dans T_3 diffèrent en tant que carte enracinées.

L'explication réside dans la nature différente des objets pris en compte : dans le premier cas, l'ensemble de points est fixé et deux triangulations (géométriques) ayant des plongements différents sont distinctes. Tandis que dans le cas de Tutte, deux triangulations (cartes planaires maximales, avec une face marquée) sont considérées équivalentes s'il existe un isomorphisme entre les deux, fixant la face marquée : on compte ainsi les triangulations enracinées (voir figure 2.6).

Puisque dans cette thèse nous considérons en particulier l'information de connectivité des maillages (abstraction faite de la configuration de points dans le plan ou l'espace), ce sont les résultats d'énumération de cartes planaires qui nous intéressent de près.

2.5.2 Cartes

Triangulations Dans le cas des maillages triangulaires homéomorphes à une sphère (où toutes les faces ont degré 3 et il n'existe pas d'arêtes multiples), le premier résultat à citer est le suivant [126] :

Lemme 14 (Tutte (1962))**.** *Le nombre de triangulations planaires* 3*-connexes enracinées ayant* $n+2$ *sommets est donné par*

$$T_n = \frac{2(4n-3)!}{(3n-1)!(n)!}$$

Cette dernière quantité vaut asymptotiquement (pour n assez grand) $\frac{16}{27}\sqrt{\frac{3}{2\pi}}n^{-5/2}(\frac{256}{27})^n$, ce qui permet facilement de calculer l'*entropie* (par unité de taille) des triangulations planaires, exprimée en *bits par sommet*(*bpv*) :

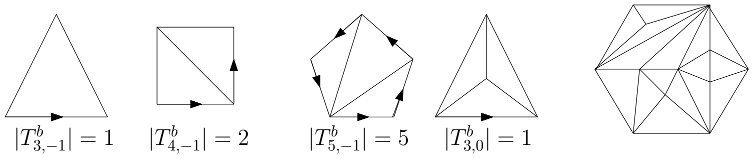


FIG. 2.7 – Exemples de triangulations planaires ayant un bord : en tant que cartes enracinées, ces triangulations sont comptées avec multiplicité, selon le choix de la racine.

$$\frac{1}{n}\log_2 T_n \;\approx\; \log_2(\frac{256}{27}) \;\approx\; 3.2451 \; bpv$$

Graphes 3-connexes Dans le cas de maillages polygonaux arbitraires (sans arêtes multiples, les faces ayant taille arbitraire) homéomorphes à une sphère, nous faisons appel à un autre résultat de Tutte [127]. Et plus précisément, l'estimation asymptotique suivante du nombre Φ_e de cartes planaires 3-connexes (polygonales) ayant e arêtes

$$\Phi_e \;\approx\; \frac{9}{2}\sqrt{\frac{6}{\pi}}e^{-\frac{5}{2}}\,4^e$$

permet d'obtenir directement l'entropie de cette classe de maillages (exprimée cette fois en *bits par arête= bpe*) : $\frac{1}{e}\log_2 \Phi_e \;\approx\; \log_2 4 \;=\; 2\,bpe$.

Triangulations avec un bord

Au cours de cette thèse nous ferons référence parfois à une classe légèrement différente de triangulations, ayant un bord simple (sans auto-intersections) de taille arbitraire (voir figure 2.7). Puisque nous sommes toujours intéressés par des triangulations géométriques (sans arêtes multiples ni boucles), ce cas correspond à la classe $\overline{T}^b_{k,n}$ des cartes planaires 3-connexes inscrites dans un polygone de taille k, ayant $k+n+1$ sommets, $m = k+2n$ faces, $e = \frac{k+3m}{2}$ arêtes et dont toutes les faces internes ont degré 3. Pour ces triangulations il existe une formule d'énumération exacte due à Mullin [93][9] :

$$|\overline{T}^b_{k,n}| = \frac{2\cdot(2k-3)!\,(2k+4n-1)!}{(k-1)!\,(k-3)!\,(n+1)!\,(2k+3n)!}, \tag{2.2}$$

En ce qui concerne l'entropie de cette classe, elle diffère de celle des triangulations de la sphère : en particulier nous fourniront à la section 5.6

[9] $\overline{T}^b_{k,n}$ compte les triangulations en terme du nombre de sommets internes et taille du bord : dans les prochains chapitres il sera plus utile de compter en terme du nombre m de triangles, en adoptant la notation $T^b_{k,m}$ (les deux formules sont équivalentes).

un calcul détaillé de l'entropie exprimée en terme de *bits par face* (où $|T^b_m|$ dénote le nombre de triangulations à bord ayant m faces) :

$$\frac{1}{m}\log_2|T^b_m| \approx \log_2\frac{17^2}{2^6} \approx 2.175\,bpf$$

2.6 Quelques outils algorithmiques

2.6.1 Décompositions d'arbres

Un problème important qui se pose lors de la conception de représentations succinctes, consiste à obtenir une décomposition de la structure de données initiale en sous structures connexes (pour des données géométriques) qui respectent certains critères de taille : notamment le but est d'obtenir une partition (ou couverture) en sous structures dont les tailles sont équilibrées.

Comme sera mieux détaillé au chapitre 3 (voir sections 3.1 et 3.2, il est usuel de traiter le codage de structures géométriques telles que les triangulations et les graphes planaires en passant par des arbres couvrants : cette remarque, et le fait que les stratégies de décomposition d'arbres joueront un rôle crucial tout au long de cette thèse, nous incitent à citer quelques résultats récents abordant ce sujet [94, 45].

Les deux lemmes qui suivent se révéleront des outils très précieux et une source d'inspiration pour d'autres stratégies de décomposition que nous allons décrire aux chapitres 6 et 7.

Partitionnement d'arbres binaires Le premier résultat que nous mentionnons concerne la décomposition d'arbres binaires [94] et son application directe est un ingrédient de base de notre représentation succincte de triangulations du chapitre 5.

Lemme 15. *Étant donné un arbre binaire $\mathcal{B}$ ayant n nœuds (chacun de degré au plus 3) et un entier positif M, il est possible de construire en temps $O(n)$, une partition $\mathcal{B}$ en sous-arbres $\mathcal{B}_i$, telle que la taille de tout sous-arbre satisfait $3M \geq \|\mathcal{B}_i\| \geq M$.*

Une telle décomposition peut s'obtenir à l'aide d'un algorithme glouton effectuant un parcours post-fixe de l'arbre (voir figure 2.8). Lors de la visite d'un nœud n_i on détermine d'abord la taille du sous-arbre enraciné en n_j : si n_i est une feuille, le sous arbre $\mathcal{B}_i$ ne contient qu'un sommet ; alors que si n_j est un nœud interne, $\mathcal{B}_i$ est l'arbre enraciné en n_i et constitué des deux sous-arbres (déjà visités) enracinés en n^l_i et n^r_i (fils gauche et droite de n_i). Si la taille de $\mathcal{B}_i$ est au moins M alors le sous-arbre $\mathcal{B}_i$ fera partie de la partition de $\mathcal{B}$; autrement $\mathcal{B}_i$ sera passé comme sous-arbre incomplet au nœud ancêtre de n_i. Cette procédure garantit que chaque sous-arbre de la partition sera de taille au plus $3M$, sauf le dernier sous-arbre restant incident à la racine de $\mathcal{B}$, étant le seul incomplet, éventuellement de taille $< M$.

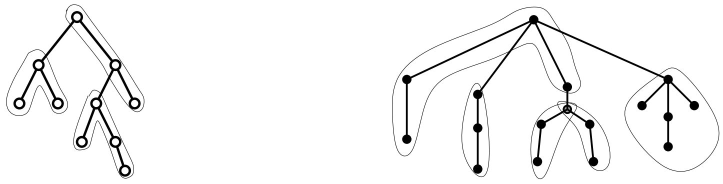


FIG. 2.8 – A gauche : partitionnement d'un arbre binaire à l'aide du lemme 15, avec paramètre $M = 3$. A droite : décomposition d'un arbre ordonné obtenue par l'application du lemme 16 avec paramètre $M = 3$: les sous-arbres forment une couverture et ne partagent que leurs racines.

Partitionnement d'arbres ordonnés Encore une fois, le problème consiste à trouver une décomposition d'un arbre en sous arbres connexes, ayant tous à peu près la même taille. Puisque ce problème, dans le cas des arbres ordonnés, n'a pas en général de solution, nous pouvons nous contenter de déterminer une famille de sous-arbres qui partagent au plus leur racines, formant une *couverture* de l'arbre initial.

Bien qu'il existe une correspondance bijective entre arbres binaires et arbres ordonnés, celle-ci ne permet pas d'appliquer directement le résultat précédent et d'obtenir ainsi la couverture désirée (essentiellement, à cause des ordres préfixe et post-fixe qui ne sont pas préservés par cette correspondance). Le lemme suivant fournit un résultat satisfaisant pour la solution de ce problème (voir [45]) :

Lemme 16. *Soit $\mathcal{B}$ un arbre ordonné à n nœuds et M un paramètre entier ≥ 2. Alors il est possible de calculer en temps $O(n)$ une famille de sous-arbres qui forment une couverture de $\mathcal{B}$ et s'intersectent au plus en leur racines, satisfaisant les contraintes de taille suivantes : pour tout sous-arbre $\mathcal{B}_i$ nous avons $\|\mathcal{B}_i\| \leq 3M - 4$. De plus, si $\mathcal{B}_i$ ne contient pas la racine de l'arbre original alors $\|\mathcal{B}_i\| \geq M$.*

2.6.2 Opérations de Rank/Select sur des vecteurs binaires

Considérons un vecteur binaire $v = b_0 \ldots b_{p-1}$, constitué de p bits et ayant poids (le nombre de '1') q. Il existe de nombreuses études et utilisations que les algorithmiciens ont faites de cette structure, munie des opérations suivantes :

• $Rank1(i)$ (resp. $Rank0(i)$) compte le nombre de '1' (resp.de '0') dans le préfixe de v : $b_0 \ldots b_i$;

• $Select1(k)$ (resp. $Select0(k)$) donne la position du k-ème '1' (resp. '0') dans v.

Tout d'abord il y a des remarques triviales, mais utiles dans la suite, que nous voulons mentionner.

- Les opérations de *Rank* et *Select* sont l'inverse l'une de l'autre, dans le sens que $Rank1(Select1(i)) = i$ $(1 \leq i \leq q)$ et $Select1(Rank1(i)) = i$, si $b_i = 1$ (des relations similaires valent pour $Rank0$ et $Select0$).

- ces deux opérations peuvent être implantées avec une structure de bitmap : ce qui d'un point de vue "complexité" n'est pas trop efficace, vu que l'inspection du vecteur nécessite dans le pire des cas de visiter $O(n)$ positions, et l'espace utilisé est de n bits.

- d'un point de vue théorique, des arguments de comptage nous assurent que $\lg \binom{p}{q}$ bits sont nécessaires pour coder et distinguer tous les vecteurs binaires de taille p et poids q.

- à l'aide des deux opérations il est possible d'exprimer et résoudre de manière intuitive des problèmes de *membership* :

- étant donné un sous ensemble S de $1 \dots n$, l'opération de $Rank1(i)$ consiste à compter le nombre d'éléments qui précèdent le $i$ème ; l'opération $Select1(i)$ revient à trouver le $i$ème plus petit élément dans S.

En ce qui concerne la conception d'une implantation efficace des structures de $Rank/Select$: de nombreux travaux ont considéré en particulier l'aspect optimalité ressource mémoire/temps de requête ([104], [103] et [32]). Bien que ces travaux ne soient pas directement utilisés et strictement nécessaires[10] pour la lecture des chapitres 4-8, nous allons mentionner ici quelques résultats et propriétés. Le résultat suivant est optimal et l'un des plus cités du domaine (voir [103]) :

Lemme 17. *Il existe une implantation d'un bit-vecteur de taille n et poids k qui supporte en temps $O(1)$ le Rank/Select sur les '0' et les '1', nécessitant au plus* $\lg \binom{n}{k} + O(n \lg \lg n / \lg n)$ *bits.*

Le résultat qui suit n'est pas optimal (en terme de ressources mémoire utilisées) et l'intérêt de le mentionner ici relève plutôt de la simplicité de sa preuve et de son utilité pour décrire de manière détaillée l'une des premières représentations compactes proposées (appariement de systèmes de parenthèses, section 3.5.1). De plus, les arguments utilisés dans sa preuve miment, d'un point de vue général, le schéma algorithmique adopté par la plupart des représentations succinctes et compactes (qui sera mieux détaillé et particularisé au cas des structure de données géométriques au chapitre 4).

[10] Sans rentrer ici dans les détails, nous pouvons anticiper au lecteur que les bit-vecteurs (fournis de Rank/Select) auxquels nous aurons recours dans cette thèse seront de "petite" taille. Plus précisément, leur longueur sera au plus $O(\lg n)$, où n est le paramètre de taille de l'objet à traiter : ce qui permettra de les représenter implicitement à l'aide d'un catalogue exhaustif (de taille $2^{O(\lg n)}$), et donc sans faire appels à des structures de données existantes plus sophistiquées.

Lemme 18. *Il existe une implantation d'un bit-vecteur de taille n et poids k qui supporte en temps $O(1)$ le Rank/Select sur les '0' et les '1', nécessitant asymptotiquement de $n + o(n)$ bits.*

Démonstration. Ici nous donnons les détails concernant l'implantation de l'opération de *Rank*1, le *Select*1 pouvant se réaliser de manière similaire avec un peu plus de travail.

L'idée de base consiste à subdiviser le vecteur en blocs de longueur $\lg^2 n$, chacun ensuite décomposé en sous-blocs de taille $\frac{1}{2}\lg n$, et de stocker un certain nombre d'informations auxiliaires pour faciliter la localisation et navigation entre sous-blocs, à l'intérieure d'un bloc et entre blocs. Plus précisément il faut stocker :

- un tableau contenant, pour chaque bloc, le nombre de '1' qui précèdent la dernière position dans le bloc ;

- un tableau (un pour chaque bloc), contenant, pour chaque sous-bloc dans le bloc courant, le nombre de '1' qui précèdent la dernière position du sous-bloc ;

- un catalogue exhaustif contenant tous les résultats de l'opération *Rank*1, appliquée à toutes les positions possibles, sur tous les vecteurs distincts de taille $\frac{1}{2}\lg n$.

Il est donc évident que le résultat de l'opération *Rank*1, appliquée au vecteur original, est simplement la somme de trois valeurs precalculées dans les tableaux ci-dessus (une fois donnée une position i, il est immédiat de trouver le bloc, le sous-bloc, et la position du bit dans le sous-bloc correspondants). Il nous reste à observer que le stockage de ces trois tableaux nécessite une quantité de mémoire auxiliaire qui est asymptotiquement négligeable : le premier tableau contient $\Theta(\lg^2 n)$ éléments, chacun sur $\lg n$ bits ; les tableaux concernant les sous-blocs contiennent en total $\Theta(\frac{n}{\lg n})$ éléments, chacun de taille $O(\lg\lg n)$; enfin, le catalogue exhaustif ne contient que $O(2^{\frac{1}{2}\lg n}) = O(\sqrt{n})$ éléments, chacun nécessitant $O(\lg n \lg\lg n)$ bits. Pour conclure, il suffit de rappeler que le terme dominant (pour les besoins mémoire de cette implantation) est donné par le stockage du vecteur binaire original, de longueur n, qui est utilisé pour indexer dans les tableaux auxiliaires. $\square$

2.6.3 Tableaux dynamiques : état de l'art

Dans cette section nous allons mentionner quelques résultats récents concernant les propriétés et implantations de tableaux dynamiques. Nous allons ici illustrer une structure de données particulière appelée *tableau extensible* qui est l'un des ingrédients que nous utiliserons dans la conception d'une représentation succincte dynamique pour les triangulations (voir chapitre 6)[11].

[11] La lecture de cette section n'est donc pas nécessaire pour la compréhension des chapitres 5 et 7 qui ne traitent que de représentations statiques.

Lorsque se pose le problème de concevoir une stratégie efficace pour maintenir des données structurées susceptibles d'être mises à jour, il est crucial de tenir compte de l'organisation dynamique de la mémoire.

Les récents efforts ont été motivés par la nécessité de concevoir de structures avec des bonnes performances, en terme des opérations d'accès et mise à jour, ainsi que des ressources mémoire utilisées, et ont données lieu à plusieurs implantations de tableaux dynamiques : nous allons mentionner ici les *tableaux restructurables (resizable)*, les *tableaux extensibles*, les *tableaux dynamiques* et les tableaux *à données de taille variable.*

Quelques définitions Un tableaux dynamique est dit *restructurable* (on a parfois utilisé le terme *extensible* dans la littérature) s'il peut maintenir une collection de n éléments (numérotés de 0 à $n-1$ et ayant tous la même taille r), supportant certaines opérations statiques et dynamiques :

– $Read(i)$: retourne l'élément d'index i.
– $Write(i, x)$: écrit x à la position i.
– $Grow$: incrémente n, avec la création d'un nouvel élément d'index n.
– $Shrink$: décremente n, avec la suppression de l'élément d'index $n-1$.

La *taille nominale* d'un tableau ayant n éléments, chacun stocké sur r bits, est de nr bits.

Parfois il est utile de considérer une collection de tableaux dynamiques, munie des opérations dynamiques suivantes pour l'allocation mémoire :

– $create(r)$ retourne un nouveau tableau dynamique, ayant des éléments de taille r ;
– $destroy(A)$ rend disponible la mémoire réservée au tableau A de la collection.

Dans ce cas, les tableaux peuvent traiter des éléments de tailles fixées différentes et la taille nominale totale de la collection est définie par $\sum_i n_i r_i$ (où n_i et r_i sont respectivement le nombre et la taille des éléments dans le tableau A_i de la collection).

Tableaux restructurables Il existe un certain nombre de travaux proposant des solutions "space-efficient" pour le problème de la conception de tableaux dynamiques. Nous commençons par mentionner ici l'un des premiers travaux considérant ce problème et introduisant des idées intéressantes [22]. Bien que ce résultat n'intervienne pas directement dans notre travail, l'une de ses améliorations [106] est à la base de l'organisation mémoire de la structure dynamique pour les triangulations décrite au chapitre 6.

Théorème 19. *Il existe une implantation de tableaux restructurables pouvant stocker n éléments, chacun de taille fixe w, qui utilise $n + O(\sqrt{n})$ mots mémoire (les opérations d'accès et mise à jour nécessitant un temps $O(1)$ et temps $O(1)$ amorti respectivement).*

Borne inférieure Le lemme suivant [22] fournit une borne inférieure concernant la taille de l'espace auxiliaire nécessaire pour l'implantation d'une structure de données supportant l'insertion et la suppression d'éléments (selon un ordre arbitraire) : ce résultat s'applique notamment aux *piles*, aux *queues*, *queues de priorité*, *queues randomisées* et aux *tableaux restructurables* mentionnés ci-dessus.

Lemme 20. *Toute structure de données dynamique supportant l'insertion d'éléments et leur suppression (dans un certain ordre arbitraire) nécessite dans le pire des cas $\Omega(\sqrt{n})$ espace auxiliaire (où n est le nombre d'éléments stockés).*

Dans le cas particulier des *tableaux restructurables*, ce dernier résultat garantit l'optimalité de cette structure de données atteignant asymptotiquement la quantité minimale d'information auxiliaire utilisée.

Tableaux extensibles Le résultat suivant améliore les implantations précédentes et sera largement utilisé au chapitre 6 (pour les détails de son implantation et analyse nous renvoyons au travail original [106]) :

Lemme 21. *Considérons une collection de a tableaux extensibles ayant taille nominale s. Alors cette collection peut être stockée sur $s + O(aw + \sqrt{saw})$ bits, pouvant supporter, dans notre modèle de calcul (w étant la taille du mot machine), les opérations d'accès (écriture/lecture) en temps $O(1)$, et les opérations de mise à jour (*create, grow *et* shrink*) en temps $O(1)$ amorti.*

Bien que ce résultat ne soit pas "optimal" au sens du lemme 20, le fait de pouvoir stocker des mots de taille r arbitraire rend ce type de tableaux particulièrement intéressant pour nos besoins.

Tableaux dynamiques avec insertion Il est à souligner que les tableaux extensibles décrit ci-dessus, ne supportent pas l'insertion/suppression d'éléments au milieu de la structure. Par ailleurs ces opérations de mise à jour de la structure ne peuvent pas s'effectuer en temps constant (même amorti), comme mis en évidence dans [39].

Si nous relâchons la condition que toutes les opérations puissent s'effectuer en temps constant (amorti ou non), le lemme suivant [106] fournit un résultat ultérieur dans l'étude du compromis entre espace auxiliaire utilisé et complexité en temps des mises à jour :

Lemme 22. *Pour toutes constantes positives c et ε il existe une implantation de tableaux dynamiques qui peut stocker une suite de données de taille w^c (où w est la taille du mot machine), supportant l'insertion/suppression* *en temps* $(\lg w)^{1+\varepsilon}$ *amorti et l'accès en temps constant* [12], *et n'utilisant que* $o(w^c)$ *bits supplémentaires.*

Tableaux "variable-bit length" Pour conclure cet état de l'art il nous reste à mentionner ici un travail récent [11] qui traite le cas de tableaux dynamiques pouvant présenter des éléments de taille variable (rappelons que les structures données ci-dessus ne pouvaient traiter que le cas d'éléments de taille fixe).

Plus précisément, cette structure de données permet de maintenir n mots binaires $a_1, \ldots, a_n$, dont les longueurs satisfont $\|a_i\| \leq w$ (comme auparavant, w est la taille d'un mot machine).

Si les performances de cette structure, en terme d'accès et mise à jour des données, correspondent aux structures précédentes, la quantité des ressources mémoire nécessaire pour son implantation est exprimée cette fois par le théorème suivant :

Théorème 23. *Il existe une représentation d'un tableau pouvant stocker dynamiquement des données* a_i *de taille variable* $\|a_i\|$ *qui nécessite* $O(w + \sum_{i=1}^{n} \|a_i\|)$ *bits (l'accès aux données et la mise à jour pouvant se faire en temps* $O(1)$ *et* $O(1)$ *amorti respectivement).*

Ce résultat récent a contribué à l'amélioration de certaines représentations compactes de graphes (voir section 3.7). En effet, le fait de pouvoir traiter des données de taille variable est intéressant et utilisé par la plupart des représentations compactes et succinctes : ces structures de données cherchent à tirer profit d'étiquettes et pointeurs locaux de taille non fixe. Cependant, ce type de tableaux dynamiques ne constitue pas un outil exploitable dans le cadre de nos travaux, où notre attention est portée sur l'aspect optimal des structures de données. Essentiellement cela est du au fait que le théorème 23 ne permet pas d'exprimer explicitement une estimation précise des besoins mémoire (à cause des constantes cachées dans la notation $grand - O$, relative au terme dominant).

[12] On entend en temps constant par rapport à la taille des données stockées.

Chapitre 3

Codage de graphes et maillages

3.1 Compression de maillages

Dans cette section nous nous proposons de fournir une présentation général des développements dans le domaine de la compression de maillages et du codage de graphes. Pour une description détaillée et exhaustive des nombreuses méthodes existantes, il existe d'excellents ouvrages [49, 5] et thèses [63, 85, 51, 43, 10], traçant l'histoire de ce domaine). Nous chercherons ici à esquisser les liens et les points communs entre les stratégies proposées. En particulier, compte tenu des thèmes abordés dans cette thèse, nous allons prêter une attention spéciale aux méthodes conçues pour la compression mono-résolution et sans perte de l'information de connectivité d'un maillage[1].

Géométrie et connectivité. Un objet géométrique est souvent représenté par un maillage, qui est constitué de deux types d'informations : la connectivité qui décrit la combinatoire de la carte sous-jacente, et la géométrie qui décrit la positions des sommets dans l'espace. La plupart des travaux de recherche se consacrent au codage de l'information de connectivité : ceci s'explique par le format de représentation (dans la mémoire principale) ou de stockage (sur disque ou d'autres supports) usuels des maillages. En effet, il est facile de vérifier que la connectivité constitue une partie quantitativement importante de l'information globale décrivant un maillage : pour des objets de grosse taille, son coût se révèle de l'ordre de la centaine de bits par sommet, alors que la géométrie en nécessite quelques dizaines.

[1] Dans ce cadre, le but est juste de réduire le plus possible la taille de la représentation d'un objet : celui-ci doit pouvoir se reconstruire de manière exacte, et sa représentation n'est connue qu'à la fin de la phase de décodage.

Par exemple, il est usuel de représenter des maillages sur disque dur (fichiers en format VRML) en utilisant la liste des index des sommets (pour chaque face, on stocke son degré et la liste des sommets incidents, en respectant l'ordre cyclique).

Dans le cas de maillages surfaciques triangulaires ayant f faces et n sommets, cela nécessite de stocker $3f \approx 6n$ index, chacun sur au moins $\lg n$ bits (le stockage nécessite $6 \cdot 32$ bits, si l'on utilise des vrais pointeurs).

Ainsi, pour des maillages ayant environ 10.000 sommets, la connectivité demande $6 \cdot 14 = 84$ bits par sommet : un coût qui est du même ordre que la simple information décrivant la géométrie ($96 = 3 \cdot 32$ bits pour les trois coordonnées de sommets en 3D). Pour des maillages de plus grosse taille, ayant environ 2^{32} sommets, il faudra utiliser 192 bits par sommet, ce qui est largement supérieur au coût des coordonnées. Le décalage entre connectivité et géométrie est encore plus marqué lorsqu'on adopte une structure utilisable en mémoire vive : les $13n$ références de la représentation décrite dans [15] nécessitent 416 bits par sommet.

3.1.1 "Growing-region approaches"

La plupart des techniques de compression de maillage (et codage de graphes) s'appuient sur le fait que l'ordre d'énumération du graphe qui représente la connectivité du maillage n'est pas fondamental. On cherche alors à calculer un nouvel ordre en appliquant une stratégie déterministe de parcours du maillage (la figure 3.2 illustre la stratégie utilisée par l'algorithme *Edgebreaker*, voir aussi la section suivante).

Ainsi le but est de définir un parcours du maillage, pendant lequel on calcul un nouvel ordre sur les sommets et sur les faces, et une manière efficace de coder les incidences entre les cellules progressivement visitées. Plus précisément, plusieurs stratégies visent à maintenir et à faire grossir une région du maillage, qui est définie par une séquence d'arêtes (ou sommets) formant un ou plusieurs cycles : ceux-ci constituent ce que l'on appelle *cut-border* ou *liste active*, qui sert à séparer les faces déjà visitées de la région restant encore à découvrir.

Pendant cette phase de parcours, on maintient une arête spéciale appelée *gate* (ou parfois un sommet *pivot*), qui sert à déterminer l'ordre dans lequel les prochaines cellules du maillage seront découvertes. À chaque étape, des cellules élémentaires incidentes au gate sont conquises : suivant certaines *opérations d'extension* qui dépendent de la stratégie de codage. Ces opérations d'extension (souvent appelées *growing operations* dans la littérature) permettent de faire grandir la région visitée ainsi que de décrire les relations d'incidence entre cellules à l'aide d'un ensemble restreint de symboles (ces opérations et leur codage sont montrées dans la figure 3.1 pour le cas de l'algorithme *Edgebreaker*).

Une particularité de ces méthodes concerne la topologie du cut-border et de la région visitée : cette dernière est souvent constituée d'une face au début, et les opérations locales d'extension visent à la maintenir simplement connexe (le cut-border étant un cycle simple). Il est toutefois nécessaire d'introduire deux opérations agissant sur la topologie du cut-border, appelées *split* et *merge* : leur utilisation intervient lorsque des auto-intersections de la région visitée sont inévitables (ce qui est le cas dans la plupart des méthodes). Plus précisément, l'opération *split* est appelée lorsque on cherche à déconnecter le cut-border en deux cycles simples (auto-intersections du *cut-border*) : ceux-ci, ainsi que les deux régions de faces non visitées (définies par les deux cycles) seront traitées récursivement de la même manière. De façon complémentaire, il existe aussi une opération *merge* qui sert à fusionner deux cycles disjoints, pour en faire un seul (simple). Ces opérations se rendent indispensables dans le cas de surfaces de genre supérieur. Finalement, on peut introduire une opération *end*, que l'algorithme de codage/décodage détecte parfois automatiquement sans recours à un symbole supplémentaire, qui sert à établir la fin du parcours lorsque toutes les cellules se trouvant dans une région délimitée par un *cut-border* ont été visitées. Pour des raisons de lisibilité, nous omettons ici les détails concernant l'implantation des phases de codage/décodage, qui dépendent essentiellement de la stratégie particulière choisie.

Bien que les stratégies de compression diffèrent dans la manière dont elles explorent le maillage et décrivent (codent) les relations d'incidence entre les cellules progressivement visitées, le cadre général que nous venons de présenter s'applique à une large partie des méthodes existantes (ce qui a été aussi souligné dans [70] et [49]).

3.1.2 Algorithmes de compression : état de l'art

Parcours topologique

En premier lieu, il ne serait pas convenable d'omettre l'un des papiers fondateurs du domaine de la compression de maillages surfaciques : l'algorithme *Topological Surgery*, introduit par Taubin et Rossignac [119], est l'une des premières stratégies proposant un parcours (en profondeur) du maillage et le codage à l'aide d'arbres couvrants (dans ce cas particulier, un arbre couvrant du dual, et un arbre couvrant du graphe primal)[2].

Cette stratégie a été très féconde et source d'inspiration pour un autre algorithme appelé *Edgebreaker* (voir [109] pour le papier fondateur) : la simplicité de son approche et de son implantation, les bons taux de compression observés dans la pratique et l'existence de bornes supérieures garanties

[2]En effet, d'autres travaux avaient déjà abordé le problème de coder la connectivité à l'aide d'arbres couvrants [77] peu de temps avant, étant moins connus puisque introduits dans un domaine différent (le codage de graphes) : les similarités entre les approches proposées dans ces deux domaines (compression de la connectivité des maillages et codage de graphes) ont été récemment explorées dans [70].

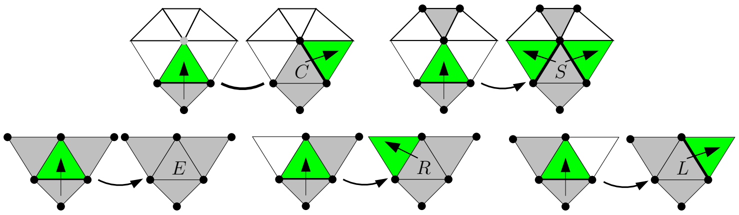


FIG. 3.1 – Ces images illustrent les règles locales du parcours de *Edgebreaker* (cas triangulaire planaire) : les triangles en gris constituent la région du maillage déjà visitée, alors que le triangles blancs sont les faces qui restent à parcourir. On distingue 5 cas différents, suivant l'état des adjacences du triangle vert (le prochain à être conquis) : l'algorithme de compression produit une suite de symboles, un pour chaque triangle visité, à partir des lettres C, L, E, R, S. Le cas le plus fréquent correspond à la conquête d'un nouveau sommet incident au triangle vert (représenté par la lettre C) : dans un maillage triangulaire à m faces cela arrive environ $\frac{m}{2}$ fois. Ainsi il est possible de garantir que la longueur moyenne du code est $2m$: il suffit d'utiliser un bit pour coder le symbole C, et des codes de longueur 3 pour les autres 4 symboles.

sur la longueur du code produit (2 bits par triangle, dans sa formulation originale), en font l'un des algorithme de compression les plus célèbres et étudiés. Le succès de *Edgebreaker* se manifeste de manière éloquente par l'abondance de ses généralisations et améliorations. Les travaux successifs proposés concernent notamment : les maillages réguliers [117], les maillages volumiques tétraédriques [118], les maillages non triangulaires [82].

D'autres travaux [69] ont aussi fourni une simplification des phases de codage/décodage (notamment la phase de décodage et reconstruction de la connectivité du maillage prenait un temps quadratique dans la version originale). Pour le cas de maillages triangulaires planaires (sans bord), une analyse plus fine (voir [80]) a établi une meilleure borne supérieure concernant le taux de compression, qui de 4 bits par sommet est passé à 3.67 bits par sommet. Pour le cas de surfaces triangulées ayant une topologie quelconque, ayant un nombre de bords et genre arbitraires, des améliorations ont été aussi proposées ([86] et [89]), soulignant le caractère *topologique* de *Edgebreaker* : dans sa formulation originale, les singularités topologiques pouvaient se traiter de manière moins naturelle, par exemple avec l'ajout d'un sommet pour clôturer un bord.

Méthodes basées sur le degré des sommets (faces)

D'autres algorithmes de compression, se basant toujours sur une visite en profondeur du maillage, ont adopté une approche différente visant à coder le graphe à l'aide des degrés de ses sommets (et éventuellement des faces,

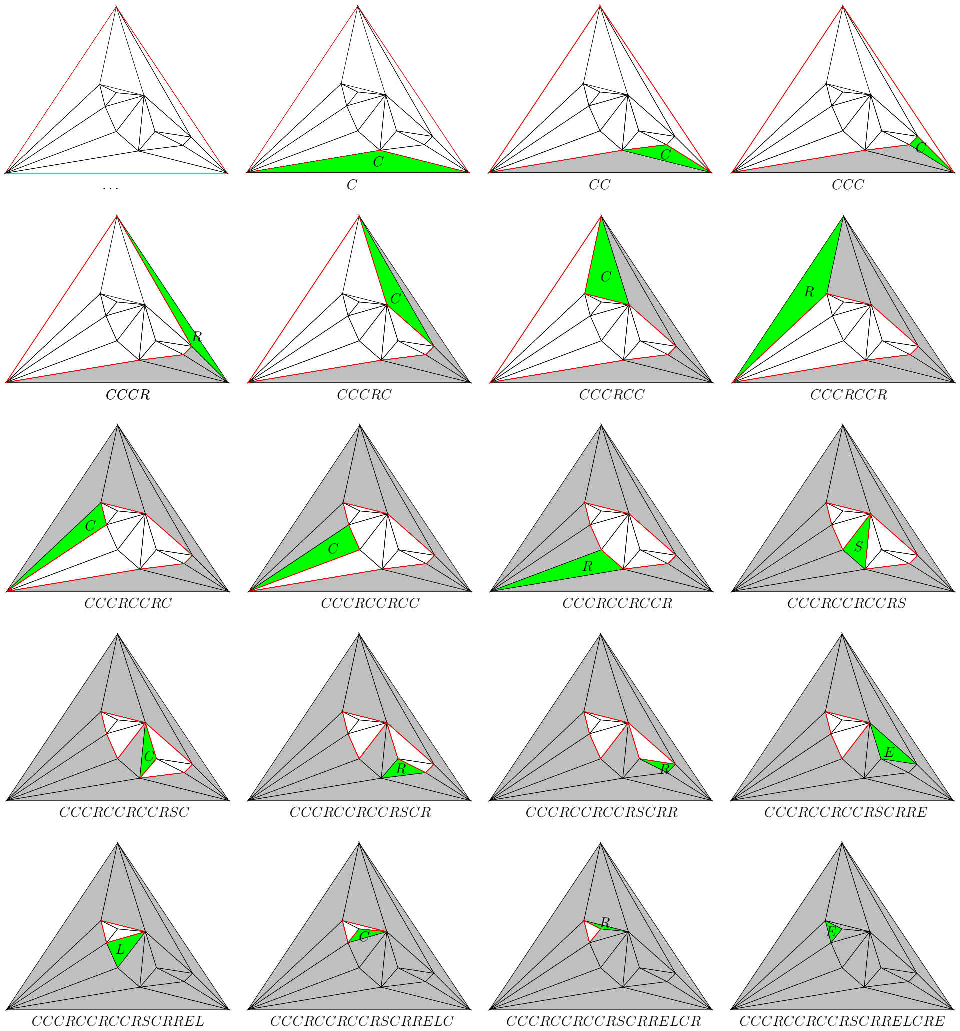


FIG. 3.2 – Ces images illustrent le codage effectué par *Edgebreaker* dans le cas d'un maillage triangulaire planaire. Son parcours correspond à une visite en profondeur en spirale du graphe dual : à chaque étape on cherche à conquérir, à partir d'un triangle donné (en vert), la face adjacente, d'abord à droite, qui n'a pas encore été visitée. Pendant le déroulement de l'algorithme de codage on maintien un ensemble d'aretes (en rouge) formant des cycles : ces cycles constituent ce qu'on appelle la *liste active* (ou *cut-border*), qui sert à séparer la région visitée (en gris) de la région qui reste à parcourir. Le code produit par cet algorithme de compression est une suite de symboles, un pour chaque triangle visité, construit à partir des lettres C, L, E, R, S.

dans le cas polygonal).

Le travail fondateur, du à Touma et Gotsman [120], a vite montré l'efficacité de cette stratégie (dans sa formulation originale, conçue pour le cas de maillages triangulaires, avec bords et de genre arbitraire) : lors du parcours du maillage (qui s'effectue de manière incrémentale en visitant les sommets) le code produit est constitué d'une suite de symboles, pour la plupart des entiers représentant le degré des sommets conquis. Parfois il s'avère nécessaire d'utiliser des codes exceptionnels, appelés *split* ou *merge*, pour décrire la déconnection de la région qui reste à visiter (cela correspond aux intersections du *cut-border*, voir section 3.1.1). Dans la pratique cette méthode est très efficace : le nombre de codes *splits* (et *merge*) est souvent limité, dans le cas de maillages réguliers on peut tirer facilement profit de la faible dispersion des degrés des sommets autour de la moyenne (dans le cas triangulaire, le degré moyen étant 6). Ainsi, l'adoption d'un codeur arithmétique permet d'atteindre des taux de compression très compétitifs : moins de 0.2 bits par sommet pour les maillages réguliers (presque tous les sommets étant de degré 6) et entre 2 et 3.5 bits par sommet pour les autres cas, en pratique.

Des généralisations de cette méthode au cas de maillages polygonaux ont été successivement introduites ([62, 79]). Cette fois, le code est constitué de deux suites, représentant les degrés des sommets et ceux des faces. Malheureusement, il n'existe pas d'analyse précise et rigoureuse pouvant fournir des bornes supérieures garanties (intéressantes) pour ces méthodes. S'il est vrai qu'en pratique le nombre de *splits* peut être considéré comme constant (et des améliorations ont été introduites réduisant ultérieurement leur nombre [4]), en général et dans le cas le pire cela est faux. Comme récemment montré par Gotsman [48], l'entropie de la suite des degrés d'un maillage triangulaire de la sphère est d'au plus ≤ 3.2364 bits par sommet, et donc est strictement inférieure à l'entropie des triangulations planaires (3.2451 bits par sommet, voir section 2.5), ce qui implique qu'il est impossible, en général, de concevoir une stratégie de codage qui n'utilise que les degrés des sommets et un nombre négligeable de *splits*. De manière similaire, il est possible de montrer que même dans le cas polygonal (graphes 3-connexes), la somme des entropies des suites des degrés des sommets et des faces est bornée par 1.9988 par arête, encore une fois strictement inférieure aux 2 bits par arête correspondant à l'entropie des graphes planaires 3-connexes.

L'approche basée sur les degrés a été aussi généralisée au cas de maillages volumiques [64] : en particulier le codage par les degrés fournit de très bonnes performances dans le cas de maillages hexaédriques (la stratégie étant valable aussi dans le cas tétraédrique, mais avec des taux de compression moins intéressants). Cette approche a inspiré aussi une stratégie de compression progressive [3] très performante dans la pratique : son efficacité est plus difficile à caractériser d'un point de vue théorique, car les degrés des sommets utilisés pour le codage ne correspondent pas au maillage initial.

En conclusion, nous soulignons que certaines questions concernant le codage basé sur les degré/valence restent ouvertes et suscitent encore l'intérêt de la communauté de la compression de maillages. En particulier, les travaux récents de Isenburg et Snoeyink ([72],[71]) apportent une meilleure compréhension du rôle des *splits*.

Compression de la connectivité : d'autres méthodes. D'autres méthodes spécialement développées pour la compression (mono-résolution et sans perte) de la connectivité d'un maillage ne rentrent pas dans les deux catégories traitées auparavant.

Toujours basé sur une approche à base de conquête du maillage, l'algorithme *Cut-border* proposé par Gumhold et al. [54] diffère des méthodes précédente dans la manière dont la conquête et le codage sont effectués, et peut aussi se généraliser en dimension supérieure au cas des maillages tétraédriques [53] (d'autres études concernant le taux de compression ont été faites plus récemment [52]). L'une des premières généralisations au cas de maillages polygonaux est due à Isenburg et Snoeyink : leur méthode, appelée *Face fixer* [68] s'inspire des algorithmes conçus pour le cas triangulaire (et en particulier de *Edgebreaker*). Un autre algorithme, introduit par Isenburg et Snoeyink [67] et appelé *Mesh collapse compression*, est basé sur l'opération de contraction d'arête : on effectue des contractions d'arêtes jusqu'à ce que le maillage soit réduit à un seul sommet, et le codage concerne encore une fois les degrés des sommets. S'il est vrai que certains algorithmes de compression (*Edgebreaker*, *Cut-border*, *Touma-Gotsman*) ont été étendus afin de traiter le cas de maillages plus généraux (polygonaux, volumiques, ...), il s'agit toujours de généralisations *ad hoc* dépendant largement de la classe d'objets considérés. Des travaux plus récents [101] proposent des méthodes valides pour une classe de maillages (manifold) plus générale, en dimension quelconque, avec ou sans bords, ayant des cellules arbitraires. D'autres travaux ont proposés des approches visant à reconduire le codage de la connectivité à la compression de la géométrie : voir [44] pour une méthode de compression progressive ; [83], [84] fournissent des stratégies de compression pour des maillages en dimension quelconque et avec une topologie arbitraire.

3.2 Codage et représentation de graphes

Algorithme	planaire	triangulé	3-connexe
Turan (1984)	$4e$		
Keeler et Westbrook (1995)	$3.58e$	$1.53e$	$3e$
Chuang et al. (1998)		$\frac{4}{3}e$	$2.377e$
He, Kao et Lu (1999)		$\frac{4}{3}e$	$2.835e$
Bonichon et al.(2003)	$2.90e$	$1.123e$	$2.90e$
Castelli Aleardi et Devillers (2004)			$2.62e$
Poulalhon et Schaeffer (2003)		$3.24n$	
Fusy et al. (2005)			$2e$

Tab. 3.1 – Comparaison des taux de compression de certaines algorithmes de codage de graphes (les termes d'ordre inférieure sont omis). Dans le cas G triangulé ou 3-connexe, on suppose G *simple* (sans arêtes multiples ni boucles). Ces codages ne constituent pas des *représentations succinctes*, dans le sens qu'ils ne permettent pas l'accès aux données (sauf pour la stratégie due à Chuang et al. [31], qui permet le test d'adjacence entre sommets en temps $O(1)$, mais avec une borne supérieure plus élevée).

3.2.1 Algorithmes de codage : état de l'art

Avec d'autres approches, exploitant des propriétés combinatoires des graphes, de nombreux algorithmes ([73] [125] [77] [17] [90] [57] [56] [17] [18]) ont été conçus pour le codage de la connectivité des graphes (ces différentes méthodes, ainsi que leurs bornes supérieures, sont montrées dans le tableau 3.1).

Le premier travail montrant qu'un graphe planaire peut se coder avec un nombre de bits proportionnel à sa taille est du à Turan [125] : ainsi pour un graphe planaire à e arêtes, $4e$ bits suffisent. Un meilleure solution a été successivement proposée par Keeler et Westbrook [77], conduisant à un codage à 3.58 bits par arête.

Un certain nombre de travaux ([31, 56, 29]) ont tiré profit des propriétés combinatoires des *ordres canoniques* et des *orderly spanning trees* associés (une généralisation des *arbres de Schnyder* [114]) : ces codages ont amélioré les résultats précédents pour certaines classes de graphes planaires (triangulés, 3-connexes, 2-connexes).

Un algorithme proposé par Bonichon et al.[17], permet de représenter les graphes planaires (non nécessairement 3-connexes) avec 2.90 bits par arête (3.37 bits par sommet dans le cas triangulaire) : leur codage passe par le

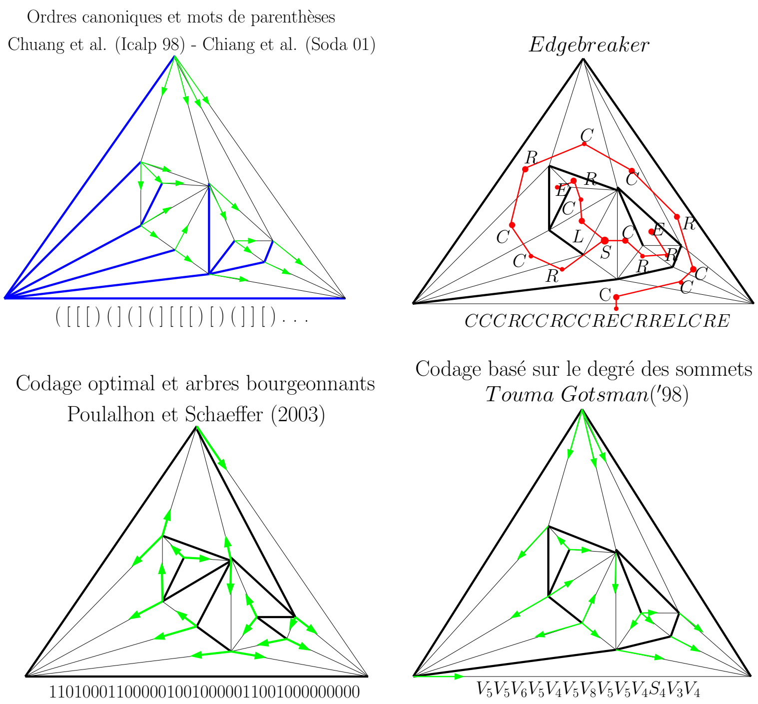


FIG. 3.3 – Ces images illustrent le cadre informel général, suggéré par Isenburg et Snoeyink [70], concernant les méthodes existantes de codage de graphes. Bien qu'elles soient assez différentes, ces méthodes présentent des similarités dans la manière de coder un graphe à l'aide d'arbres couvrants.

calcul d'une super-triangulation du graphe initial et tire profit des propriétés des réaliseurs et de leurs extensions.

Toujours en se basant sur le calcul d'une triangulation du graphe initial (de manière *canonique*), il est possible de coder un graphe planaire 3-connexe avec au plus 2.62 bits par arête [24] : lors d'un parcours sans *splits* (sans auto-intersections) du graphe (inspiré des ordres canoniques [76]), on utilise le degré des faces pour décrire la manière dont elles sont triangulées. Comme dans le cas d'autres méthodes basées sur le codage des degrés ([120, 79]), on peut tirer profit de la régularité de la distribution des degrés : cette fois une borne supérieure peut s'établir grâce à l'absence de codes *splits*.

Ce n'est que très récemment que des codages optimaux de complexité linéaire ont été trouvés pour certaines classes, très intéressantes, de cartes planaires : ce codage, basée sur des bijections (Poulalhon et Schaeffer [99] pour les triangulations, et Fusy et al. [42] pour les cartes 3-connexes) entre cartes et arbres bourgeonnants seront présentés plus en détail à la section 3.3.

Certaines similarités entre les méthodes de codage de graphes ([125], [77], [99]) et compression de maillages ([120],[53],[109]) ont été mises en évidence par Isenburg et Snoeyink [70] : plus précisément ils proposent un cadre informel général pour décrire ces stratégies en terme d'arbres couvrants (dans ce cadre rentrent aussi d'autres stratégies basées sur les *ordres canoniques* et systèmes de mots de parenthèses [31] : voir la figure 3.3).

Avec une approche tout à fait différente, d'autres travaux ([57], [90]) ont proposé un codage asymptotiquement optimal pour certaines classes de graphes planaires, qui peut s'obtenir en temps $O(n \lg n)$. Il est néanmoins à mentionner que ces méthodes recourent à une décomposition récursive du graphe basée sur les séparateurs (un codage exhaustif de complexité exponentielle est nécessaire pour coder les micro composantes de taille sous-logarithmique). Ces méthodes sont ainsi limitées sous plusieurs points de vue : d'une part ces codages nécessitent l'implantation d'algorithmes d'isomorphisme de graphes et le calcul de séparateurs planaires (Lipton et Tarjan [88]). D'autre part, l'optimalité en terme de taux de compression n'est qu'asymptotique, et aucune borne explicite ne peut être fournie pour la taille de la représentation.

3.3 Arbres bourgeonnants et codage optimal

Nous allons donner une présentation générale de deux résultats récents concernant le codage optimal des triangulations et cartes 3-connexes planaires qui nous seront utiles dans la suite. Il est à noter que ces résultats ne sont strictement nécessaires que pour les représentations proposées au chapitre 7.

3.3.1 Triangulations 3-connexes

Le premier résultat [100, 99] concerne une correspondance entre une classe spéciale d'arbres bourgeonnants et les triangulations planaires : il est à souligner que ce résultat fourni une preuve bijective du lemme 14 (d'énumération de Tutte pour les triangulations), ainsi que le premier algorithme (de complexité linéaire) de codage optimal pour cette classe de graphes (dont le taux de compression correspond à 3.24 bits par sommet).

Clôture partielle Le premier pas consiste à considérer la famille d'arbres bourgeonnants constituée des arbres plans à n nœuds et deux bourgeons par nœud, ayant comme cardinalité $\frac{2}{4n-2}\binom{4n-2}{n-1}$.

Il est maintenant possible de définir une opération de clôture partielle qui fait correspondre à chaque arbre bourgeonnant une carte planaire dont toutes les faces internes ont un degré 3. Plus précisément, cette opération de clôture consiste à effectuer un parcours du contour de l'arbre et à fusionner (lorsque il est possible) chaque bourgeon avec le sommet qui le précède sur

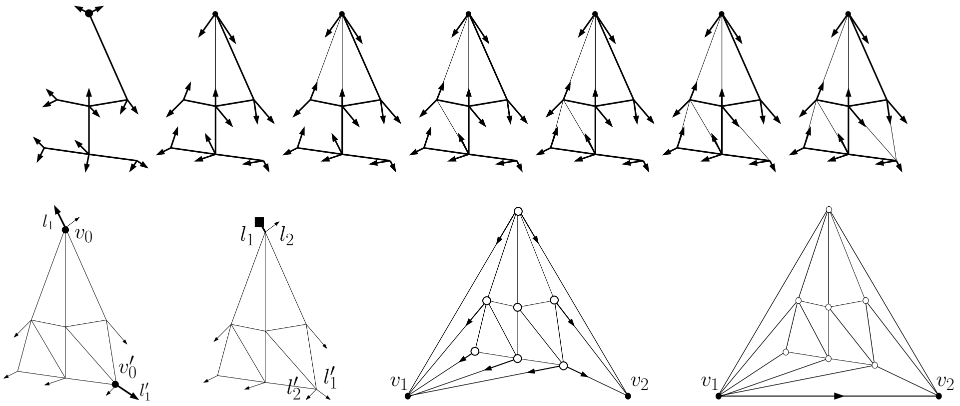


FIG. 3.4 – Correspondance entre arbres bourgeonnants et triangulations planaires. Dans cette exemple on part d'un arbre ayant 7 nœuds dont la clôture partielle est illustrée par les premières images. En bas est montrée la clôture complète donnant lieu à une triangulation ayant $7 + 2$ sommets.

la face externe, de manière à ce que la face formée soit triangulaire[3].

Clôture complète Maintenant on peut définir une opération de *clôture complète* qui fait correspondre à un arbre bourgeonnant *équilibré* (ou plutôt à sa clôture partielle), une triangulation planaire (pour la définition d'arbre équilibré voir ci-dessous). Premièrement il faut ajouter deux sommets v_1 et v_2, auxquels nous allons raccorder les bourgeons encore existants incidents à la face externe, où l_1, l_2 (resp. l'_1, l'_2) désignent les bourgeons incidents à v_0 (resp. v'_0), de la manière suivante : tous les bourgeons entre l_1 et l'_2 (compris) sont raccordés à v_1 ; tous les bourgeons entre l'_1 et l_2 sont raccordés à v_2 (un arbre bourgeonnant est équilibré s'il est enraciné en l_1 ou en l'_1). Finalement il suffit d'ajouter l'arête orientée (v_1, v_2), obtenant ainsi une triangulation planaire (enracinée) à $n + 2$ sommets (les étapes de cette procédure sont illustrées par la figure 3.4).

Coder les arbres (et donc les triangulations) Le codage (optimal) des triangulations passe par le codage des arbres bourgeonnants correspondants, comme exprimé par le lemme suivant (qui sera souvent mentionné dans la suite de cette thèse) :

Lemme 24. *Un arbre à n nœuds ayant deux bourgeons par nœud peut se coder par un mot binaire de longueur $4n - 2$ et poids $n - 1$.*

[3]Un bourgeon peut se refermer lorsque le sommet qui le précède sur le bord de la face infinie n'est pas incident à un bourgeon pas encore fusionné (les étapes de cette clôture partielle sont illustrées par les premières images de la figure 3.4).

Démonstration. Il suffit d'effectuer un parcours du contour de l'arbre, correspondant à une version légèrement modifiée du code préfixe décrit à la section 2.1 : lorsqu'on visite (en partant de la racine et avec un parcours en profondeur en sens trigonométrique) une arête interne pour la première fois on écrit le symbole '1' ; tandis que lors de la visite d'un bourgeon ou lors d'une deuxième visite d'une arête interne, on écrit '0'. Finalement il suffit de remarquer qu'un tel arbre bourgeonnant contient $2n$ bourgeons et $n-1$ arêtes internes. ☐

Le lemme précédent conduit directement à un codage optimal, à partir du code d'un arbre bourgeonnant ayant n nœuds : puisque il existe au plus $\binom{4n-2}{n-1}$ de tels mots, un codeur entropique (par exemple un codeur arithmétique), permet d'obtenir un code de longueur

$$\lg \binom{4n-2}{n-1} + o(n) \approx \lg \binom{4n}{n} \approx \frac{256}{27} n$$

Il est à souligner que bien que les trois classes d'objets considérés dans cette section (mots binaires de longueur $4n-2$ et poids $n-1$, arbres bourgeonnants à n nœuds, triangulations à $n+2$ sommets) aient des cardinalités différentes [4], leur entropies (au premier ordre) coïncident : ce qui nous permettra d'affirmer, au cours des chapitres suivants, qu'une triangulation ou un arbre bourgeonnant peuvent se coder optimalement par un mot de longueur environ $4n$ et poids n.

Remarques finales Dans cette section nous avons juste illustré de manière intuitive l'un des sens de la bijection introduite dans [99], exprimée par le résultat suivant :

Théorème 25 (Poulalhon et Schaeffer (2003))**.** *Il existe une bijection entre la classe des arbres équilibrés ayant n nœuds et deux bourgeons par nœud, et la classe des triangulations planes (enracinées)* 3*-connexes à $n+2$ sommets, cette correspondance pouvant se calculer en temps $O(n)$.*

La description de l'autre sens de la bijection nécessite un peu plus d'effort, et surtout la maîtrise de certaines propriétés combinatoires des *réaliseurs* des triangulations planaires ([114]). Bien que ces sujets soient très passionnants, leur présentation reste en dehors des finalités de cette thèse : nous renvoyons donc à d'autres excellentes lectures traitant ces sujets de manière rigoureuse et pédagogique ([20, 100, 124, 122, 123]).

[4] Les arbres bourgeonnants à nœuds étant en nombre de $\frac{2}{4n-2}\binom{4n-2}{n-1}$, et les triangulations (ou les arbres bourgeonnants équilibrés) ayant pour cardinalité $\frac{1}{n}\frac{2}{4n-2}\binom{4n-2}{n-1}$

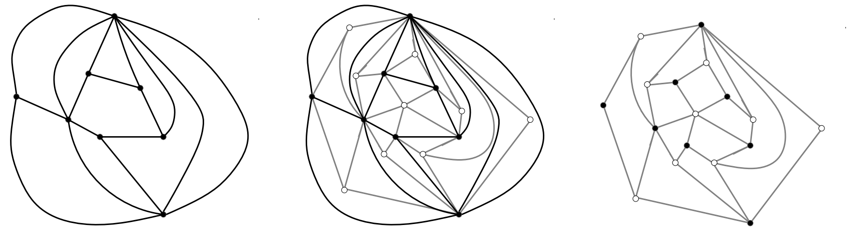


FIG. 3.5 – Correspondance entre dissections irréductibles et cartes planaires 3-connexes : les cercles noirs (resp. cercles blancs) représentent les sommets du graphe primal (dual).

3.3.2 Cartes 3-connexes, quadrangulations et arbres binaires

De manière similaire, une correspondance (conduisant à un codage optimal) entre cartes 3-connexes et arbres binaires a été proposée dans [42] : comme pour le résultat de la section précédente, nous nous intéressons seulement à la façon dont il est possible d'obtenir une telle carte, à partir d'un arbre binaire. Le premier pas consiste à considérer une autre classe de graphes, pour lesquels cette correspondance est une bijection.

Quadrangulations et cartes Une quadrangulation est une carte planaire dont toutes les faces ont degré 4. Une *dissection* de l'hexagone par des faces quadrangulaires est une carte planaire ayant une face externe de degré 6 et dont les faces internes ont degré 4. Une telle quadrangulation est dite *irréductible* si elle ne contient pas de 4-*cycles séparateurs* (cycles de 4 sommets dont la suppression déconnecte le graphe) : on parlera alors de *dissections irréductibles*.

Le stricte lien existant entre carte 3-connexes et quadrangulations est exprimé par le lemme suivant (légère variante de la construction classique du dual d'une carte planaire - voir la figure 3.5) :

Lemme 26. *Il existe une bijection entre les cartes planaires* 3*-connexes et les quadrangulations irréductibles.*

Démonstration. Considérons une carte planaire M : après avoir coloré en noir ses sommets (sommets du graphe primal), plaçons un sommet blanc au milieu de chaque face (sommets du graphe dual), ainsi que les arêtes incidentes aux sommets blancs de façon à ce que toute face soit triangulée. Ces nouvelles arêtes issues des sommets blancs forment une quadrangulation : toute face est formé de deux triangles incidents à la même arête (noire) de M. De plus il est possible de vérifier que la 3-connexité de M est équivalente à l'irréductibilité de Q (absence de 4-cycles séparateurs). □

Pour conclure, il reste à observer qu'il est possible d'associer une dissection d (enracinée) de l'hexagone à toute quadrangulation irréductible Q (enracinée) simplement en supprimant l'arête racine.

Clôture partielle et complète d'arbres binaires Suivant la même approche utilisée à la section précédente, nous considérons une classe spéciale d'arbres bourgeonnants, cette fois contenant les arbres binaires à n nœuds (dont tous nœuds ont degré 3). Il est possible de définir une opération de clôture partielle, qui agit de manière similaire à celle déjà introduite (pour les triangulations). Lors du parcours (en sens trigonométrique) du contour de l'arbre, les bourgeons précédés de trois arêtes internes sur le bord de la face infinie, sont fusionnés de façon à créer une face quadrangulaire.

Le résultat de cette phase est une carte planaire (appelée la *clôture partielle* de l'arbre) ayant des faces internes quadrangulaires et dont la face externe (de taille arbitraire) est incidente à un certain nombre de bourgeons "non fusionnés". La *clôture complète* consiste alors à ajouter un hexagone (englobant la carte) et renfermer les bourgeons encore existants aux sommets l'hexagone à ce que tout couple de bourgeons consécutifs forme une face quadrangulaire (voir dernière image de la figure 3.6).

Remarques finales Comme pour le cas des triangulations, nous n'avons pas la possibilité d'illustrer ici la correspondance inverse (*ouverture*) qui à une dissection quadrangulaire Q (et donc à une carte 3-connexe) associe un arbre binaire (dont la clôture complète est exactement Q). Pour une présentation détaillée nous renvoyons donc au travail original [42], dont le résultat principal est exprimé par :

Théorème 27 (Fusy, Poulalhon et Schaeffer (2005))**.** *Il existe une bijection entre la classe des arbres binaires (pleins) ayant n nœud internes et la classe des dissections quadrangulaires de l'hexagone (sans 4-cycles séparateurs) ayant n sommets internes, pouvant se calculer en temps $O(n)$.*

3.4 Structures de données succinctes et compactes

3.4.1 Généralités

Les objets algorithmiques et géométriques dont nous avons parlé jusqu'à présent (arbres, graphes, maillages) peuvent se représenter par un codage "linéaire" pour la transmission sur le réseau et le stockage sur disque, ou bien de manière explicite dans la mémoire principale si l'on cherche à pouvoir explorer l'objet. Dans le premier cas, lorsque l'on vise juste à réduire la taille mémoire d'un objet, le codage prend le nom de *compression* : comme déjà mentionné à la section 3.1 la compression de graphes et maillages a

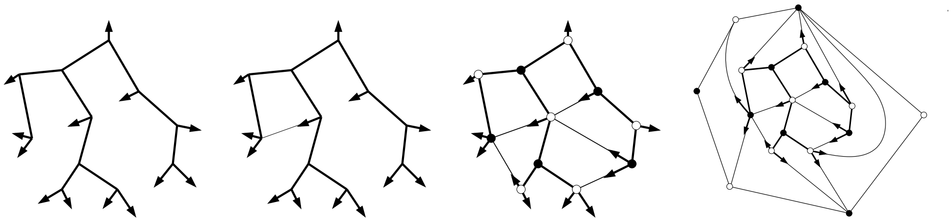


Fig. 3.6 – Ces images illustrent les clôture partielle (et complète) qui associe à tout arbre binaire à n nœuds, une dissection quadrangulaire de l'hexagone ayant n sommets internes.

suscité un vaste intérêt dans la communauté de la géométrie algorithmique, du graphisme et de la combinatoire et algorithmique des graphes. Dans cette thèse nous considérons une classe différente de représentation, qui puisse se considérer comme une structure de données pour les objets à manipuler. En particulier, dans le cadre des représentations succinctes ou compactes le but est de concevoir une représentation ayant une petite taille mémoire, qui permette en même temps de répondre à certaines requêtes locales sur l'objet de manière efficace.

Plus précisément, une telle structure est dite *succincte* si le coût en ressource mémoire correspond asymptotiquement à l'entropie de la classe des objets à représenter, tandis que l'on parle d'une structure *compacte* lorsque l'entropie est atteinte à un facteur constant près. Plus précisément, considérons une classe d'objets de taille n (par exemple les arbres binaires à n nœuds) ayant cardinalité $2^{\alpha n}$ (asymptotiquement pour n assez grand). Une représentation est dite *succincte* si l'espace utilisé est de $\alpha n + o(n)$ bits ; alors qu'il s'agit d'une structure *compacte* si elle nécessite $O(n)$ bits. Il est à noter que toute représentation valide ne peut pas utiliser $o(\alpha n)$ bits, car il faut pouvoir distinguer parmi tous les objets de la classe. Pour ce qui est l'exploration de l'objet, le but est d'effectuer en temps constant (parfois en temps polylogarithmique) certaines opérations de navigation et requêtes d'adjacences locales (tester dans un graphe si deux sommets sont adjacents, passer d'une face à ses voisines,...). Toutes ces notions seront présentées de manière plus rigoureuse au chapitre 4 qui a été spécialement conçu pour fournir un cadre plus précis et un schéma général pour la compréhension et description des représentations succinctes d'objets géométriques.

3.4.2 Mots de parenthèses, arbres et graphes : état de l'art

Considérons d'abord une structure de données simple et très célèbre : les arbres binaires. Pour cette classe d'objets, les résultats d'énumération mentionnés à la section 2 ainsi que la notion d'entropie nous assurent que le codage d'un arbre binaire ayant n nœuds nécessite $\lg \frac{1}{n+1}\binom{2n}{n} \approx \lg \frac{2^{2n}}{n^{3/2}} = 2n + o(n)$ bits. Du point de vue de la compression, un codage optimal à $2n$ bits est simplement donné par le mot de parenthèses (voir chapitre 2.1) correspondant au parcours préfixe du contour de l'arbre : cette représentation ne supporte pas un accès efficace à l'information contenue dans la structure combinatoire de l'arbre (en général la navigation locale nécessite l'inspection d'un nombre arbitraire $\Theta(n)$ de symboles). Du point de vue des structures de données explicites, une représentation utilisant un pointeur (ou index) par nœud permet d'effectuer facilement certaines opérations de navigation locale : dans ce cas il est néanmoins impossible d'utiliser moins de $\Omega(n \lg n)$ bits (il faut $\Theta(n)$ pointeurs de taille $\Omega(\lg n)$). Comme déjà mentionné dans l'introduction, ces remarques ont conduit au problème de concevoir une représentation d'un arbre binaire utilisant asymptotiquement 2 bits par nœud et permettant à la fois une exploration efficace de l'arbre en temps $o(n)$.

Graphes et maillages En ce qui concerne les structures de données complexes telles que les graphes et les maillages, la presque totalité des travaux existants sont basées sur les représentations succinctes d'arbres mentionnées ci-dessus. Ainsi les représentations de Jacobson [74] et Munro et Raman [96], ont conduit directement à des structures compactes tirant profit des décompositions en 4 pages des graphes planaires, qui supportent la navigation locale respectivement en temps $O(\lg n)$ et $O(1)$[5] : dans la version utilisant la représentation succincte optimale pour les mots de parenthèses il est possible d'effectuer le test d'adjacence entre sommets, en utilisant asymptotiquement au plus $2e + 8n$ bits (pour un graphe ayant n sommets et e arêtes). Dans le cas des triangulations à m triangles (avec éventuellement un bord) une telle structure nécessite entre $7m$ et $12m$ bits (selon la taille du bord).

Le meilleur résultat théorique, connus jusqu'à peu de temps, était du à Chuang *et al.* [31]. En se basant encore sur une représentation de mots de parenthèses (de systèmes multiples de mots de parenthèses) et exploitant les propriétés des ordres canoniques, il était possible d'améliorer ultérieurement le terme dominant pour l'espace mémoire utilisé : ainsi il existe des représentations compactes utilisant asymptotiquement $2e + 2n$ bits les graphes planaires 3-connexes, et $2e + n$ pour les graphes triangulés (équivalent à $3.5m$ bits pour des triangulations à m faces de la sphère). Ce travail a été amélioré et étendus plus récemment par Chiang *et al.* [29, 30] au cas de graphes pla-

[5]Il est à souligner que la plupart des représentations compactes mentionnées dans cette section proposent (au moins dans leur formulation originale) des opération de navigation légèrement différentes de celles considérées dans cette thèse (qui sont plus intéressantes d'un point de vue géométrique) : en général ces codages permettent de vérifier l'adjacence entre sommets, alors que nos algorithmes permettent de naviguer naturellement entre les faces du maillage (en plus de tester les adjacences).

naires plus généraux, cette fois nécessitant de $2e + 2n$ bits.

D'autres approches D'autres méthodes plus pratiques ont considéré le problème d'obtenir une représentation moins coûteuse en mémoire permettant une navigation efficace. Une première approche consiste à spécialiser et rendre moins redondantes les structures de données explicites classiques, afin d'obtenir un gain d'un facteur constant (*Star-vertices* [75]).

Plus récemment, et avec une approche totalement différente de celles présentées dans cette section, Blandford *et. al.* [14] ont proposé des représentations compactes pour la classe des graphes séparables, permettant d'implanter certaines requêtes (adjacence et degré) sur les sommets en temps $O(1)$ (voir la section 3.7 pour plus de détails).

3.5 Appariement de mots de parenthèses

Nous allons fournir ici une description détaillée des structures succinctes et compactes concernant les mots de parenthèses équilibrées introduites par Jacobson [74] et améliorées par Munro et Raman [96], qui ont été le véritable point de départ des travaux développés dans cette thèse[6].

Opérations sur des mots de parenthèses Étant donné un mot de parenthèses équilibrées il est naturel de disposer des opérations suivantes :

- *match*(i) : étant donnée une parenthèse ouvrante (fermante) à la position i, retourne la position de la parenthèse fermante (ouvrante) appariée.
- *excess*(i) : retourne l'excès à la position i, qui correspond à la différence entre le nombre de parenthèses ouvrantes et celui de parenthèses fermantes.
- *enclose*(i) : étant donné une paire de parenthèses (dont l'ouvrante est à la position i), retourne la paire de parenthèses plus proche les englobant.

Esquisse du schéma à suivre L'idée commune aux deux représentations mentionnés ci-dessus consiste d'un point de vue général à décomposer la structure de donnée initiale (le mot de parenthèse équilibré) en sous-structures (dans ce cas appelées *blocs*) de taille B (en générale $B = \Theta(\lg^c n)$, pour une certaine constante entière c) et distinguer un certain type de parenthèses spéciales (*pionniers*). Après avoir remarqué que le nombre des pionniers est globalement "négligeable", leur stratégie consiste à stocker de manière explicite la position de la parenthèse fermante appariée à chaque pionnier. Il faut aussi associer de l'information à chaque bloc, pour qu'il soit possible de trouver efficacement la parenthèse fermante correspondante à toute parenthèse, étant donnée sa position dans le bloc et la position du pionnier

[6]Bien que ces résultats ne soient pas directement utilisés dans la conception de nos représentations, est notre opinion que le fait de les présenter ici pourrait aider le lecteur dans la lecture et compréhension des chapitres 4 à 7.

qui la précède. Enfin, de l'information supplémentaire est nécessaire pour détecter les parenthèses qui sont pionniers et supporter certaines opérations (Rank/Select) dans le bloc.

Les deux structures qui suivent [74, 96] (ainsi que leur généralisations et améliorations [94, 45, 46, 106, 9]) diffèrent essentiellement dans le nombre de niveaux formant la décomposition, la taille des sous-structures et la nature des informations et structures de données supplémentaires utilisées pour supporter efficacement certaines requêtes et opérations locales dans/entre les sous-structures.

3.5.1 La représentation de Jacobson

La représentation décrite par Jacobson est basée sur une décomposition en un seul niveau consistant de $\Theta(n/\lg n)$ blocs, chacun de longueur $\Theta(\lg n)$. Les positions des parenthèses fermantes correspondantes aux pionniers sont stockées explicitement, ainsi qu'un bit-vecteur muni des opérations de *Rank/Select* qui sert à retrouver la position et le rank des pionniers (enfin une information supplémentaire est associée à chaque bloc, telle que la différence entre le nombre de parenthèses ouvrantes et fermantes, l'*excès*).

Comme on le verra à la fin de cette section, la solution proposée par Jacobson n'est pas optimale, dans la mesure où l'espace utilisée est bien loin de la borne inférieure des $2n + o(n)$ bits, et parce que les requêtes prennent un temps $O(\lg n)$ et non un temps $O(1)$.

Définition 28. *Une parenthèse ouvrante est* far *si la parenthèse correspondante appariée est placée dans un bloc différent. Une parenthèse far est un* pionnier *si la parenthèse appariée est placée dans un bloc différent de la parenthèse correspondante à la parenthèse far qui la précède (dans le même bloc).*

L'une des propriétés cruciales exploitées par Jacobson concerne la dépendances linéaire existante entre le nombre global de pionniers et le nombre de blocs :

Lemme 29. *Le nombre de pionniers dans une décomposition d'un mot de parenthèses en b blocs est au plus $2b - 3$.*

Démonstration. Considérons un graphe G ayant b sommets (un pour chaque bloc), où il existe une arête entre deux sommets u et v s'il existe une parenthèse "far" dans le bloc relatif à u qui est appariée par une parenthèse contenu dans le bloc relatif à v. Le nombre d'arêtes de G est au moins le nombre de pionniers (chaque pionnier est associé à une arête différente de G). De plus, puisque le mot de parenthèses est équilibrés, G admet un plongement planaire tel que les sommets sont placés le long d'une droite (respectant l'ordre des blocs), les arêtes ne se croisant pas : G est "outer-planar", ainsi son nombre d'arêtes (et donc de pionniers) est au plus $2b - 3$. □

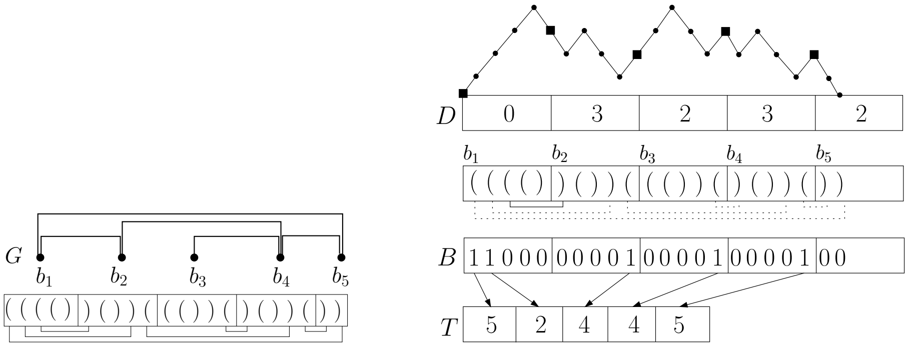


FIG. 3.7 – Ces images illustrent les propriétés et la structure de la représentation de mots de parenthèses équilibrées de Jacobson. L'image de gauche montre le graphe G associé au mot de parenthèses (lemme 29). La structure décrite dans le théorème 30 est montrée à droite : y sont représentées les structures auxiliaires B, T, D, et la décomposition du mot en blocs de longueur $\lg n$, ainsi que la distributions des parenthèses "far" correspondante (les pionniers sont dessinés en pointillé).

D'un point de vue plus général, la validité du lemme précédent réside dans la planarité caractérisant les mots de parenthèses équilibrées (leur structure arborescente associée). L'un des points essentiels mis en évidence dans cette thèse concerne justement le rôle crucial joué par la planarité (locale) dans la conception de représentations succinctes (optimales) de structures de données géométriques.

Théorème 30 (Jacobson [74])**.** *Étant donné un mot de parenthèses équilibré de longueur $2n$, il existe une structure de données qui implante l'opération match(i) en temps $O(\lg n)$, nécessitant de $\Theta(n)$ bits supplémentaires.*

Démonstration. Commençons par fournir une description des structures de données utilisées. Après avoir décomposé le mot en $b = \Theta(\frac{n}{\lg n})$ blocs de longueur $\lg n$ et calculé les positions des pionniers, il faut stocker :

- un bit-vecteur B (avec une structure de Rank/Select) de longueur $2n$ et poids b, qui décrit la distribution des pionniers : $B[i] = 1$ ssi la parenthèse à la position i-ème est un pionnier ;

- un tableau T de longueur b contenant, pour chaque pionnier, l'index du bloc contenant la parenthèse fermante correspondante.

- un tableau D, ayant une case pour chaque bloc, contenant la *profondeur d'imbriquement* de la parenthèse à la première position du bloc : $D[i]$ est simplement un entier (sur $\lg n$ bits), donné par la différence entre le nombre de parenthèses ouvertes et de parenthèses fermantes à la première position (dans le mot de départ) du i-ème bloc.

L'opération $match(i)$ peut s'implanter de la façon suivante (k désigne la position de la parenthèse fermante associée à celle ouvrante en position i).

- i n'est pas un far, alors sa parenthèse correspondante q peut se retrouver avec une recherche locale dans le bloc courant (ce qui nécessite au plus $\lg n$ inspections, vu la longueur de chaque bloc) ;

Autrement, il faut d'abord calculer l'index du bloc contenant la parenthèse k, de la manière qui suit :

- $r := Rank_1(B, i)$ fournit la position du pionnier qui précède la parenthèse i et $T[r] = b_k$ contient l'index du bloc cherché (rappelons que par définition, le pionnier r qui précède i appartient au même bloc que i : de même pour les parenthèses fermantes correspondantes).

- finalement, à l'aide des "profondeurs" stockées dans le tableau D, on peut calculer la profondeur de i et retrouver ainsi la position de k dans le bloc b_k (avec une recherche locale dans b_k, et sachant que i et k doivent être à la même profondeur).

Pour conclure, il ne reste qu'à évaluer la quantité d'information supplémentaire utilisée par les structures B, T et D. Le lemme 18 fournit une implantation simple et satisfaisante (dans ce cas particulier) du bit-vecteur B, nécessitant en totale de $2n + o(n)$ bits. Le lemme 29 garantit que le nombre de pionniers est "négligeable", et que le tableau T peut être ainsi stocké sur $(2b-3) \cdot \lg n \leq [2(\frac{n}{\lg n} - 3)] \cdot \lg n = 2n + o(n)$ bits. Et de manière similaire, le tableau D nécessite de $(\frac{n}{\lg n}) \cdot \lg n = n + o(n)$ bits. □

3.5.2 Une représentation succincte optimale

La première représentation succincte (optimale) pour les mots de parenthèses est due à Munro et Raman [94].

Afin de réduire la quantité d'espace utilisée et d'atteindre asymptotiquement la borne inférieure optimale (l'entropie par unité de taille étant 1 bits par symbole), il est possible d'avoir recours à une décomposition en 3 niveaux du mot initial : ainsi on choisit des sous-blocs de taille respectivement $\Theta(\lg^2 n)$, $\Theta((\lg\lg n)^2)$ et $\Theta(\lg\lg n)$. Comme auparavant, à chaque niveau est associée de l'information supplémentaire utile pour répondre à certaines requêtes dans/entre les sous-blocs (notamment, retrouver les positions des parenthèses appariées aux pionniers).

Théorème 31 (Munro et Raman [94]). *Étant donné un mot de parenthèses équilibré de longueur $2n$, il existe une représentation qui nécessite au plus $2n+o(n)$ bits permettant d'effectuer en temps $O(1)$ les opérations $match(i)$, $enclose(i)$ et $excess(i)$.*

Plus récemment une version simplifiée de la représentation succincte de mots de parenthèses a été proposée dans [46]. Cette fois le schéma de décomposition n'utilise que 2 niveaux, constitués de sous-blocs de taille respectivement $\Theta(\lg^2 n)$ et $\Theta(\lg n)$. Une légère modification de la définition de parenthèse *pionnier* garantit que le sous-mot constitué des pionniers est lui même un mot de parenthèses équilibré, ce qui permet d'obtenir une simplification considérable de la démonstration du théorème précédent. En particulier, cette amélioration est basée sur une utilisation plus rationnelle des structures de *Rank*/*Select* : cela permet notamment de s'affranchir d'une structure de donnée sophistiquée (construction d'un tableau de hachage parfait [55]) qui était nécessaire dans la version proposée dans [94]. De plus, l'optimalité asymptotique est garantie et le terme d'erreur $o(n)$ est amélioré et de la forme $O(\frac{n \lg\lg n}{\lg n})$ (ce terme était de la forme $\Theta(\frac{n \lg\lg\lg n}{\lg\lg n})$ dans la représentation originale [94]).

Ce dernier résultat, ainsi que d'autre représentations succinctes d'arbres binaires dynamiques [96, 106], sont plus proches de la stratégie que nous avons adopté dans nos travaux (pour ce qui est de l'utilisation de 2 niveaux seulement, et de la meilleure implantation des bit-vecteurs avec *Rank*/*Select*). Plus précisément, ces représentations rentrent dans le schéma algorithmique nous allons décrire et formaliser au chapitre 4, afin d'unifier les structures de données développées dans cette thèse : il est à noter que des représentations succinctes de mots de parenthèses et d'arbres peuvent s'obtenir comme cas particuliers de l'application de notre schéma (étant donné qu'un arbre, ainsi que le mot de parenthèses associé, a la structure d'une carte planaire).

3.6 Représentations succinctes d'arbres et graphes

Les résultats concernant les mots de parenthèses équilibrés introduits à la section précédente constituent un outil crucial dans la conception de représentations succinctes et compactes pour des structures plus complexes et intéressantes telles que les arbres (ordonnés et binaires) et les graphes (planaires).

Dans ce cadre il est naturel de considérer les opérations de navigation locales suivantes (certaines étant communes aux graphes et aux arbres) :

- $Adjacent(u, v)$: teste l'adjacence entre deux sommets u et v ;
- $degre(v)$: retourne le nombre de voisins d'un sommet ;
- $parent(v)$: retourne le nœud ancêtre de v (dans un arbre)
- $fils\_gauche(v)$: retourne le fils gauche du nœud v, dans un arbre binaire (de manière similaire se définit $fils\_droit(v)$) ;
- $taille\_sous\_arbre(v)$: retourne la taille du sous-arbre enraciné en v ;
- $fils(v, i)$: dans un arbre ordonné, retourne le i-ème fils du nœud v (dans une carte l'opération $voisin(v, i)$ correspondante retourne le i-ème voisin de v selon l'ordre trigonométrique) ;

3.6.1 Arbres binaires et ordonnés

Une fois acquise la représentation du théorème 31, il est presque immédiat d'obtenir une représentation succincte pour les arbres, en vertu de la correspondance bijective existante entre mots de parenthèses équilibrés, arbres ordonnés et arbres binaires (voir section 2.1).

Corollaire 32. *Étant donné un arbre ordonné à n nœuds, il existe une représentation utilisant $2n + o(n)$ bits, qui supporte en temps $O(1)$ les opérations $parent(v)$, $taille\_sous\_arbre(v)(v)$ et en temps $O(i)$ l'opération $fils(v, i)$. De plus, dans le cas des arbres binaires (dont les nœuds ont au plus 2 fils), il est possible d'implanter en $O(1)$ les opérations $fils\_gauche(v)$ et $fils\_droit(v)$*

La validité de ce résultat passe essentiellement par l'isomorphisme entre mots de parenthèses et arbres. Alors le codage de l'arbre est exactement donné par la représentation succincte du mot correspondant : un nœud v dans l'arbre (donné par son indice, par rapport à l'ordre préfixe de parcours) correspond à la position i d'une parenthèse ouvrante dans le mot. Certaines opérations sur l'arbre sont l'application directe d'opérations sur les mot : le parent d'un nœud v est simplement donné par la parenthèse ouvrante fournie par $enclose(i)$, où i est la parenthèse correspondante à v. Ou encore, $taille\_sous\_arbre(v)$ est donnée par la moitié de la différence entre le nombre de parenthèses fermantes et ouvrantes à la position correspondante à $match(i)$. Pour l'implantation détaillée d'autres requetes nous renvoyons au travail original [96].

3.6.2 Graphes planaires et plongement livresque

La première représentation compacte de graphes est une application des structures conçues pour les mots de parenthèses et tire profit d'un type spécial de représentation des graphes planaires.

Un *plongement à k pages* d'un graphe G à n sommets est défini par une permutation de $1 \dots n$ et en une partition des arêtes en *k pages* telles que : les sommets peuvent se placer le long d'une droite (le tranche d'un livre), et les arêtes appartenant à la même page peuvent se dessiner sans croisement (l'ordre des sommets étant fixé par la permutation et commun à toutes les pages). L'*épaisseur* d'un graphe est le nombre minimal de pages d'un plongement livresque quelconque de G. Une propriété très intéressante de cette classe de plongements, concerne les graphes planaires, pour lesquels il existe toujours un plongement à 4 pages qui peut se calculer en temps $O(n)$ [129].

Cette propriété des graphes planaires, ainsi que les représentations des mots de parenthèses équilibrées conduisent au résultat suivant (introduit dans [74] et amélioré dans [94]) :

Corollaire 33 (Munro et Raman [94])**.** *Étant donné un graphe G à k pages ayant n sommets et e arêtes, il existe une représentation utilisant $2kn + 2e + o(nk + e)$ bits, qui supporte en temps $O(k)$ le test d'adjacence*

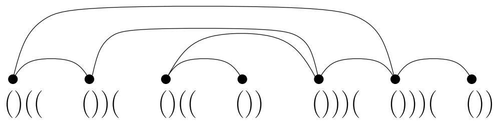


FIG. 3.8 – Graphes planaires et plongements à k pages.

entre sommets (ainsi que le calcul du degré d'un sommet). En particulier, la représentation d'un graphe planaire nécessite $8n + 2e + o(n)$ *bits et permet les mêmes opérations en temps* $O(1)$.

L'idée principale consiste à observer que le plongement des arêtes de G (dessinées dans une même page) correspond exactement à la structure d'imbriquement d'un mot de parenthèses équilibré. Intuitivement il suffit d'imaginer les sommets placés sur une droite horizontale et les arêtes de la même page dessinées dans le demi-espace supérieur : la figure 3.8 illustre comment associer aux arêtes G appartenant à la même page un mot de parenthèses équilibré.

3.6.3 Graphes planaires et ordres canoniques

Le résultat du corollaire précédent peut être amélioré de manière considérable à l'aide d'une généralisation de la structure succincte pour les mots de parenthèses (supportant cette fois un nombre arbitraire, constant, de types de parenthèses différentes), et surtout d'une caractérisation plus fine des graphes planaires 3-connexes. La formulation qui est la plus intéressante dans le cadre de cette thèse concerne les graphes planaires simples (triangulés ou non) et peut s'exprimer de la manière suivante :

Théorème 34 (Chuang et al. [31])**.** *Soit G un graphe planaire 3-connexe simple ayant n sommets et e arêtes. Alors il existe une représentation compacte de G qui supporte en temps $O(1)$ l'adjacence entre sommets (en temps $O(d)$ le calcul du degré d'un sommet ou le parcours de ses voisins) nécessitant $2n + 2e + o(n)$ bits.*
Si G est une triangulation planaire à n sommets ($e = 3n - 6$) alors une telle représentation nécessite $2e + n + o(n) = 7n + o(n)$ bits[7].

Le premier pas vers la preuve de ce résultat consiste à considérer une généralisation du théorème 31 aux cas de *systèmes de mots de parenthèses*, qui dans le cas d'alphabets sur 4 symboles peut s'exprimer de la manière suivante :

[7]Même si non explicitement dit, il est à mentionner que le terme d'erreur sous-linéaire $o(n)$ est toujours de la forme $\Omega(\frac{n \lg \lg n}{\lg n})$, puisque toutes les représentations succinctes d'arbres et de graphes traitées dans cette section sont basées essentiellement sur une représentation succincte de mots de parenthèses.

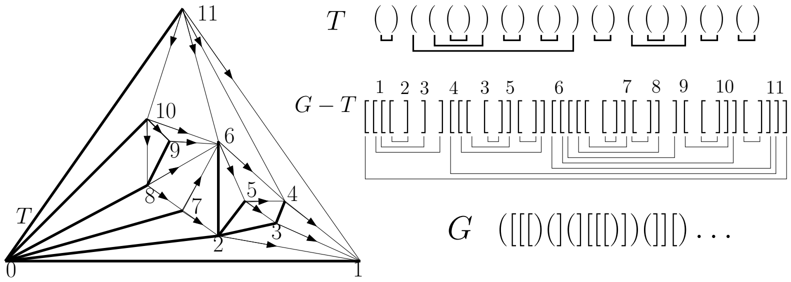


FIG. 3.9 – Un graphe G planaire triangulé à 12 sommets, induit avec l'*ordre canonique (rightmost)* sur ses sommets. L'arbre canonique T, enraciné en v_0, est représenté par le mot de parenthèses équilibré ()((())()())()(())()(). Les arêtes restantes dans $G \setminus T$ sont représentées par le mot : [[[[ ]]]][[[[ ]][ ]][[[[[[ ]][ ]]][[ ]]][]]]]. Ce mot peut s'obtenir par le parcours préfixe de l'arbre T de la manière suivante. Lors de la visite d'un nœud v, on parcourt ses arêtes incidentes dans $G \setminus T$ en ordre trigonométrique : on écrit '[' si l'arête est entrante, alors que si l'arête est sortante on écrit ']'. L'orientation d'une arête de $G \setminus T$ s'obtient à partir des étiquettes de ses deux sommets incidents : chaque arête est orientée entrante vers le sommet ayant l'étiquette plus petite.

Lemme 35 (Chuang et al. [31])**.** *Soit S un mot de parenthèses multiples sur l'alphabet $\{\ (,),[,]\ \}$, non nécessairement équilibré. Alors il existe une représentation compacte du mot S qui supporte en temps $O(1)$ les opérations match(i), enclose(i), rank(i), select(i), pour chaque type de parenthèse. De plus, une telle représentation nécessite de $|S| + o(|S|)$ bits ($|S|$ étant la longueur du mot S).*

Il ne reste alors qu'à choisir une bonne stratégie pour le codage du graphe G : suivant l'approche commune à plusieurs algorithmes de codage de graphes (voir section 3.2), la voie à suivre consiste à calculer un arbre recouvrant T de G et déterminer une manière efficace de coder T ainsi que les arêtes dans $G \setminus T$ (la figure 3.9 illustre de manière intuitive la stratégie adoptée pour coder T et $G \setminus T$).

Dans ce cadre, les *ordre canoniques* (introduits pour les graphes plans généraux dans [38], et étendus aux graphes 3-connexes dans [76]) fournissent un outil très puissant, en vertu de leurs propriétés combinatoires caractérisant de manière fine la planarité des graphes.

Pour une présentation complète et détaillée de tous ces résultats nous renvoyons le lecteur au travail original [31] et ses extensions [29, 30].

3.7 Structures compactes : graphes et petits séparateurs

Une stratégie tout à fait différente a inspiré les travaux de Blanford et al. [14, 11, 12, 13], qui ont proposé des représentations compactes pour certaines classes de graphes et maillages, pouvant gérer des opérations de navigation locale de manière efficace[8]. L'efficacité de leur approche repose sur les propriétés des graphes *séparables* (récemment décrites dans [92, 91]) : intuitivement un graphe est séparable si tous ses sous-graphes peuvent être partitionnés en deux parties ayant environ la même taille, juste en supprimant un nombre relativement limité de sommets[9]. Plus précisément, un graphe G ayant n sommets est *séparable* (où avec petits séparateurs) s'il existe un ensemble de $O(n^c)$ sommets (appelé *coupe*) dont la suppression déconnecte G en deux sous-graphes (appartenant à la même classe que G) ayant chacun au plus αn sommets (avec $\alpha < 1$, $c < 1$).

Le principe fondamental consiste à adopter une représentation explicite des relations d'adjacence des sommets du graphe, utilisant des pointeurs locaux de "petite taille" : les sommets sont d'abord étiquetés à l'aide des séparateurs de graphes, et leurs voisins sont représentés par un entier, de "petite" taille, codant la différence du sommet original. Le point crucial est que pour certaines classes de graphes (séparables), il existe une manière d'étiqueter les sommets telle que deux sommets adjacents auront des index assez proches. Cette remarque, et une méthode efficace de codage par différences, permet d'établir le résultat suivant [14] :

Lemme 36. *Soit G un graphe à n sommets ayant des petits séparateurs, et pour lequel chaque arête (v_1, v_2) peut se coder avec $O(\lg|v_1 - v_2|)$ bits. Alors l'espace utilisé pour coder toutes les arêtes est au plus $O(n)$ bits et les requêtes d'adjacence entre sommets peuvent s'effectuer en temps $O(1)$.*

Dans ce cadre, ils s'avère que certaines classes de graphes sont spécialement intéressantes : par exemple, les graphes planaires admettent des séparateurs de taille $O(n^{1/2})$ [88] et certains maillages *bien formés* en $\mathbb{R}^d$ ont des séparateurs de taille $O(n^{1-\frac{1}{d}})$ [91]. Il résulte alors que les maillages planaires, les maillages $3D$ volumiques *bien formés*, ainsi que les maillages surfaciques de genre borné satisfont les hypothèses du lemme 36 et admettent donc des représentations compactes de taille linéaire.

[8] En ce qui concerne le modèle de calcul, ces travaux adoptent le même modèle de machine *word-RAM* utilisé dans cette thèse

[9] Il existe plusieurs notions de séparabilité, en terme de sommets ou d'arêtes.

Deuxième partie

Représentations succinctes de graphes

Chapitre 4

Représentations succinctes : schéma général

Le but de ce chapitre est de présenter une formulation plus générale et précise du paradigme algorithmique sousjacent aux représentations compactes et succinctes mentionnées précédemment. En particulier nous nous proposons de fournir un schéma général[1] qui puisse unifier à la fois les représentations existantes (mots de parenthèses, arbres, graphes planaires) et les nouvelles représentations succinctes de maillages faisant l'objet principal de cette thèse (triangulations et cartes planaires, décrites aux chapitres 5-7).

4.1 Introduction

Comme il a été plusieurs fois déjà remarqué, la plupart des structures de données sont basées sur des représentations explicites par pointeurs.

Par exemple, cela arrive dans le cas des arbres binaires, d'habitude implantés en utilisant, pour chaque nœud, un pointeur vers son fils gauche et son fils droit : si d'une part cette implantation permet un accès efficace à l'information stockée dans l'objet, d'autre part l'utilisation de $O(m)$ pointeurs chacun sur $O(\lg m)$ bits, oblige à utiliser au moins $\Omega(m \lg m)$ bits pour représenter un arbre ayant m nœuds.

Or, du point de vue de la théorie de l'information, $2m$ bits devraient suffire, puisque il existe moins de 4^m arbres binaires de taille m.

Cette remarque a conduit, étant donné une classe $\mathcal{C}_m$ d'objets de taille m, au problème de concevoir, si possible, une *représentation succincte* de ces objets, c'est-à-dire une structure de données telle que :

– le coût du stockage de la représentation R d'un objet dans $\mathcal{C}_m$ correspond asymptotiquement (au premier ordre) à l'entropie de la classe,

[1]Une version préliminaire des résultats de ce chapitre, a été publiéà SoCG 2006[28].

ce qui s'exprime par (lorsque m tend à l'infini)

$$\text{taille}(R) = \lg |\mathcal{C}_m| \cdot (1 + o(1))$$

- la représentation permet de répondre à certaines types de requêtes locales sur l'objet en temps $O(1)$ (pire cas),
- la représentation permet la modification des objets en temps polylogarithmique amorti par opération.

La deuxième propriété assure que la représentation peut être considérée comme une structure de donnée pour les objets de $\mathcal{C}_m$.

La première propriété en fait un codage *succinct* : parfois, ne pouvant pas atteindre l'optimalité, il est commun de se contenter d'une version plus faible de *compacité* de la forme

$$\text{taille}(R) = O(\lg |\mathcal{C}_m|)$$

La dernière propriété décrit enfin l'aspect dynamique de la représentation.

Il existe aussi une autre catégorie de structures de données, appelées *implicites*, qui ne stockent que les données à traiter, arrangés dans un certain ordre préétabli dont la taille est de la forme

$$\text{taille}(R) = \lg |\mathcal{C}_m| + O(1)$$

Bien qu'elles ne nécessitent pas de stocker de l'information auxiliaire, ces structures de données suscitent moins d'attention : la difficulté de traiter efficacement des requêtes locales, la classe restreinte d'objets auxquels elles s'appliquent et surtout l'impossibilité de maintenir l'ordre sur les données (et donc la structure) après mise à jour sont parmi les raisons qui rendent les structure de données implicites moins intéressantes, au moins dans le cadre algorithmique adopté dans cette thèse.

4.1.1 Schéma général : esquisse

Le schéma général que nous allons adopter afin de concevoir et décrire une structure de données compacte ou succincte pour une classe d'objets de taille n est esquissé ci-dessous et sera mieux précisé dans la section 4.2 :

- D'abord l'objet est décomposé en sous morceaux de taille $\Theta(\lg n)$, appelés *micro morceaux*, de telle manière qu'ils soient assez petits pour que le catalogue de tous les différents micro morceaux possibles puisse se construire en temps $o(n)$ en utilisant au plus $o(n)$ bits. Un tel micro morceau est représenté par une référence dans le catalogue, et la somme des tailles de toutes ces références décrivant l'objet initial va coïncider avec l'entropie de la classe.
- Les relations d'incidences qui décrivent la façon dont le découpage en micro morceaux a été effectué sont représentées par un graphe G

des micro morceaux. Si on suppose qu'il existe $O(\frac{n}{\lg n})$ micro morceaux avec un nombre sous linéaire de relations d'incidences entre eux (ce qui est le cas, si le graphe G est par exemple planaire), alors le graphe admet une représentation explicite classique de coût $O(n)$ utilisant des pointeurs de taille logarithmique, ce qui fournit déjà une représentation compacte de l'objet.

- ensuite des *mini* morceaux de taille $O(\lg^2 n)$ sont construits en regroupant $\Theta(\lg n)$ micro morceaux, ce qui permet d'utiliser des pointeurs de taille $O(\lg n)$ uniquement entre mini morceaux, tandis que les relations de voisinage entre micro morceaux, appartenant au même mini morceau, peuvent se décrire avec des pointeurs locaux de taille $O(\lg \lg n)$. Puisque les mini et micro morceaux sont en nombre respectivement de $O(\frac{n}{\lg^2 n})$ et $O(\frac{n}{\lg n})$, cette approche à plusieurs niveaux comporte des coûts sous-linéaires, respectivement de $O(\frac{n}{\lg^2 n} \lg n)$ bits et $O(\frac{n}{\lg n} \lg \lg n)$ bits, pour la représentations des relations d'adjacence décrites par le graphe G, ce qui rend la structure de données succincte.

4.1.2 Représentations succinctes : rappels

Mots de parenthèses et arbres. D'un point de vue général ce paradigme algorithmique a été introduit pour représenter de manière compacte des mots de parenthèses équilibrées par Jacobson [74], et les améliorations de Munro et Raman [96] ont conduit à une représentation succincte (voir section 3.5). Le paramètre de taille d'un mot de parenthèse est son nombre de caractères et l'optimalité correspond à 1 bit par caractère. Dans ce contexte une requête naturelle consiste à retrouver, étant donnée une parenthèse ouvrante à une certaine position, la parenthèse fermante correspondante. En exploitant une célèbre bijection entre mots de parenthèses et arbres planaires (ou ordonnés), il est donc naturel d'obtenir une représentation succincte de cette dernière classe d'objets qui ne nécessite que $2n$ bits.

Graphes planaires. Une fois remarqué qu'une carte planaire peut se décomposer en plusieurs arbres recouvrants, il est naturel d'appliquer à ces derniers le paradigme conçu pour les arbres. Il est néanmoins à observer qu'une telle transformation des graphes aux arbres, n'étant pas en générale bijective, conduit à des représentations qui ne sont que compactes et non succinctes. Suivant cette approche une première représentation compacte pour les graphes planaires utilisant $2e + 8n$ bits a été donnée dans [96] et améliorée dans [31, 29]($2e + 2n$ bits).

Séparateurs Il est à rappeler qu'une stratégie de codage compact de graphes et d'autres structures a été proposée par Blandford *et al.* [14] utilisant une approche totalement différente : bien que l'idée sous-jacente soit similaire, à savoir l'utilisation de pointeurs locaux de "petite taille", l'approche à l'aide des petits séparateurs est totalement différente et ne peut pas rentrer dans le schéma général que nous allons décrire. De plus, cette stratégie nécessite des algorithmes efficaces pour le calcul de séparateurs et la complexité de la représentation n'est pas facile à caractériser, vue la difficulté de fournir explicitement des bornes sur le coût mémoire total.

4.1.3 Notre contribution

Dans nos travaux [27, 26] nous avons montré pour la première fois comment étendre le paradigme mentionné auparavant pour qu'il puisse s'appliquer directement aux triangulations et aux cartes planaires plus généralement (sans donc passer par la représentation des arbres ou des mots de parenthèses), obtenant ainsi des représentations succinctes pour ces classes d'objets. Cette approche s'est révélée utile aussi pour gérer des mises à jour locales et traiter le cas de triangulations de genre supérieur borné [26].

Du point de vue méthodologique nous allons formaliser le schéma générale qui est commun aux représentations succinctes de cartes planaires illustrées dans les chapitres 5-7.

Par rapport au cadre utilisé dans les premiers travaux concernant le codage des mots de parenthèses et des arbres [74, 96], et aussi dans notre premier travail concernant les triangulations, le schéma présenté dans ce chapitre cherche à rendre explicite le rôle fondamental joué par la propriété de planarité locale des objets à représenter (sans cette propriété il serait impossible de tirer profit de la structure à plusieurs niveaux à la base du schéma).

De plus, nous relâchons la condition, centrale dans tous les travaux précédents, que les micro morceaux doivent appartenir à une classe ayant la même entropie que la classe d'objets à représenter. Cette dernière propriété, couplée à des nouvelles techniques de décomposition spécialement conçues pour les triangulations et les cartes 3-connexes planaires, nous permettra par exemple de décrire les premières représentations succinctes optimales pour ces classes d'objets (voir chapitre 7).

4.2 Schéma général des micro-mini morceaux

Dans cette section nous allons fournir une description plus rigoureuse et précise du schéma général esquissé dans la section 4.1.1.

A un premier niveau de détail, notre schéma peut s'appliquer à une structure de données quelconque, ayant une entropie linéaire et pouvant se décomposer en une collection de sous-structures connectées par un graphe globalement creux : l'ensemble des structures satisfaisant ces propriétés inclut, par exemple, les systèmes de mots de parenthèses, les arbres, et plus généralement la classe des graphes plongés sur des surfaces de genre borné.

Nous allons spécialiser ensuite notre schéma pour représenter l'information combinatoire des cartes (correspondant à la connectivité des maillages surfaciques).

4.2.1 Additivité de l'entropie et du catalogue

Afin de pouvoir appliquer notre schéma à une classe d'objets combinatoires, nous demandons d'abord que l'entropie de cette classe soit linéaire.

Plus précisément, considérons une classe $\mathcal{C} = (\mathcal{C}_n)$ d'objets, où n est un paramètre de taille indiquant le nombre de cellules élémentaires d'un certain type (faces, sommets, ...), et supposons que l'ensemble $\mathcal{C}_n$ des objets de taille n soit de cardinalité finie $|\mathcal{C}_n|$. Rappelons que l'entropie $||\mathcal{C}_n||$ est la quantité définie par

$$||\mathcal{C}_n|| := \lg |\mathcal{C}_n|$$

et mesure la diversité de la classe. Une classe d'objets a entropie linéaire s'il existe une constante α telle que

$$||\mathcal{C}_n|| = \alpha n + o(n), \quad \text{pour } n \text{ qui tend vers l'infini.}$$

En d'autres termes, la cardinalité de la classe $\mathcal{C}_n$ croit approximativement (au premier ordre) comme une simple exponentielle $2^{\alpha n}$, ce qui implique que αn bits sont nécessaires (et en théorie suffisants) pour pouvoir distinguer et indexer tous les éléments. La constante α est parfois appelée l'entropie par unité de mesure : $\alpha = 2$ bits par nœud pour les arbres binaires, et comme mieux détaillé dans la section 2.5, $\alpha = 1.62$ bit par triangle pour les triangulations de la sphère, et $\alpha = 2$ bits par arête pour les graphes planaires 3-connexes. Il faut toutefois souligner qu'il existe d'autres classes d'objets combinatoires, telles que la classe des permutations de $\{1, \dots, n\}$ ou tels que les graphes généraux à n sommets, qui ont une entropie non linéaire : d'ordre $\Theta(n \lg n)$ pour les permutations, et d'ordre $\Theta(n^2)$ pour les graphes.

Notre stratégie vise à décomposer chaque objet de $\mathcal{C}$ en sous morceaux choisis parmi ceux qui sont inclus dans un catalogue de plus petits objets : contrairement à ce qui a été fait dans les travaux précédents portant sur les représentations succinctes [94, 96, 106], nous ne demandons pas que ce catalogue ne contienne que des éléments de $\mathcal{C}$.

Nous construirons un catalogue contenant les éléments d'une classe un peu plus large $\mathcal{D} = (\mathcal{D}_{m,k})$, telle qu'il existe une constante β et une fonction positive $g(m) = O(\lg m)$ t.q.

$$||\mathcal{D}_{m,k}|| \leq \alpha m + \beta\, k \lg m + g(m) \tag{4.1}$$

et $|\mathcal{D}_{m,k}| = 0$ pour $k \geq Km$ (pour une certaine constante K).

Les objets de cette nouvelle classe $\mathcal{D}_{m,k}$ sont censés être utilisés pour décrire des micro morceaux d'éléments de $\mathcal{C}$, où m est le nombre de cellules élémentaires, et k un paramètre, souvent appelé le *nombre de côtés*, mesurant la complexité du bord d'un micro morceau. Dans la relation 4.1 la constante α est censée être la même que pour la classe $\mathcal{C}_n$, et αm est la *partie additive* de l'entropie, tandis que le terme $\beta k \lg m$ représente une quantité supplémentaire d'entropie due à la décomposition en micro morceaux (l'idée est de permettre que chaque morceau puisse avoir un bord de taille et complexité arbitraires, en fait aussi grand que la taille du morceau lui même). Par définition de l'entropie, $||\mathcal{D}_{m,k}||$ bits suffisent pour distinguer chaque élément dans $\mathcal{D}_{m,k}$ et référencer un élément dans le catalogue correspondant.

Nous supposons que tout élément M de $\mathcal{C}_n$ puisse se décomposer, en temps $O(n)$, en :

- une collection $(M_1, \ldots, M_p)$ d'éléments de $\mathcal{D}$, où $M_j \in \mathcal{D}_{n_j,k_j}$ pour quelque n_j et k_j.
- un graphe orienté G, ayant comme ensemble de sommets $\{N_1, \ldots, N_p\}$, qui décrit la manière dont les micro morceaux $\{M_1, \ldots, M_p\}$ sont regroupés pour former M : l'idée est que les arcs de G décrivent les relations d'adjacences entre les côtés des M_j.

Plus précisément, un sommet N_j de G contient les informations suivantes :

– n_j, k_j, et l'index de l'objet M_j dans le catalogue $\mathcal{D}_{n_j,k_j}$,

– les références aux nœuds voisins de N_j dans le graphe G (pour chaque côté de M_j, on stocke l'index du voisin adjacent, et le numéro de côté correspondant).

En particulier le nombre d'arêtes du graphe G est borné par le nombre total de côtés des micro morceaux M_j.

En premier lieu, il nous faut une hypothèse garantissant que la partie additive de l'entropie coïncide avec l'entropie αn de la classe d'objets à laquelle M appartient.

Hypothèse 1. *La décomposition en sous morceaux est additive en le paramètre de taille (intuitivement, les cellules élémentaires ne sont presque pas partagées) :*

$$n_1 + \ldots + n_p = n + O\left(\frac{n}{\lg n}\right),$$

Observons qu'il est admissible qu'un certain nombre négligeable de cellules élémentaires soient partagées entre micro morceaux, comme exprimé par le terme $O(\frac{n}{\lg n})$.

Un point important à souligner, est que dans notre schéma il n'est pas demandé que la classe $\mathcal{D}_m = \bigcup_k \mathcal{D}_{m,k}$ ait la même entropie de $\mathcal{C}_m$: en particulier, dans les deux exemples de représentations de cartes analysés aux chapitres 5 et 7, nous aurons $||D_m|| \sim \alpha' m$ avec $\alpha' > \alpha$.

Par conséquent, afin qu'une représentation soit compacte nous avons besoin d'une deuxième hypothèse concernant le nombre de côtés des micro morceaux.

Hypothèse 2. *La décomposition concerne un nombre sous linéaire de côtés :*

$$k_1 + \ldots + k_p = O\left(\frac{n}{\lg n}\right).$$

Cette deuxième hypothèse permet d'affirmer que le coût total dû au stockage des références vers tous les M_i reste d'ordre αn. Il est à noter que nous ne faisons aucune hypothèse sur la taille des côtés : en particulier le nombre de cellules appartenant aux côtés d'un micro morceaux peut être de taille linéaire.

La prochaine hypothèse assure que les éléments de $\mathcal{D}$, nécessaires pour la décomposition, sont contenus dans un petit catalogue de taille négligeable, et que le nombre global de micro morceaux est sous linéaire.

Hypothèse 3. *Dans la décomposition chaque objet M_j satisfait les contraintes de taille suivantes : sa taille est entre $\frac{c}{3}\lg n$ et $c \lg n$, avec $c < 1/\alpha'$, où α' est l'entropie par unité de taille de la classe $\mathcal{D}$.*

Maintenant supposons donc que les références pointent vers un tableau A, contenant les représentations explicites de tous les éléments du catalogue $\mathcal{D}_m$, avec $m \leq c \lg n$ pour une certaine constante c.

Si la constante c est bien choisie et assez petite, le nombre d'entrées dans le tableau A est garanti sous-linéaire : en effet $||\mathcal{D}_m|| = \alpha' m \leq c\alpha' \lg n$, ainsi avec $c < 1/\alpha'$ le nombre d'entrées de A est $O(n^{c\alpha'})$.

Le coût total dû au stockage du tableau A reste donc sous-linéaire, tant que la quantité d'information utilisée pour représenter chaque élément est polynômiale en sa taille $m = \Theta(\log n)$.

En particulier, il est possible de stocker explicitement le résultat de certaines requêtes locales, telles que des requêtes d'adjacence entre cellules élémentaires, sans que cette information supplémentaire affecte de manière significative la taille globale de la structure de données[2].

4.2.2 Compacité

L'hypothèse 2 ci-dessus garantit que le graphe G possède au plus $O(n/\lg n)$ arêtes, et l'hypothèse 3 que le nombre de sommets est au plus $O(n/\lg n)$, ce

[2]Ceci sera mieux précisé à la section 4.2.5, tandis que l'implantation des requêtes (pour des triangulations) sera détaillée à la section 5.5.

qui permet déjà d'obtenir la compacité de la structure, comme exprimé par le Lemme suivant :

Lemme 37. *Sous les hypothèses 1, 2, and 3 le stockage du graphe G nécessite $O(n)$ bits.*

Démonstration. Observons que chaque nœud N_j du graphe G contient les informations suivantes : le nombre de cellules n_j et de côtés k_j du micro morceaux correspondant, l'indice de M_j dans le catalogue $\mathcal{D}_{n_j,k_j}$, ainsi que les références aux voisins de N_j dans le graphe G.

Puisque n_j et k_j sont plus petits que $c \lg n$ et $Kc \lg n$ respectivement, ils peuvent être codés avec $2 \lg \lg n + O(1)$ bits.

En sommant sur les $O(n/\lg n)$ nœuds de G, nous obtenons un coût de $O\left(\frac{n \lg \lg n}{\lg n}\right)$ bits.

L'équation (4.1) nous dit que chacune des références vers les M_j coûte $\alpha n_j + \beta\ k_j \lg n_j + O(\lg n_j)$ bits, ce qui permet d'évaluer le coût total de toutes les références vers le catalogue $\mathcal{D}$ en faisant la somme sur tous les M_j :

$$\begin{aligned}\sum_j \|\mathcal{D}_{n_j,k_j}\| &\leq \alpha \sum_j n_j + \beta(\sum_j k_j) \lg(\max_j n_j) + O\left(\sum_j \lg n_j\right) \\ &\leq \alpha n + O(\frac{\alpha n}{\lg n}) + \beta O(\frac{n}{\lg n}) \lg \lg n + \frac{n}{\lg n} O(\lg \lg n)\end{aligned}$$

En faisant appel aux hypothèses 1 et 2 il est facile de vérifier que le coût global de tous les références vers le tableau A se réduit donc à

$$\alpha n + O\left(\frac{n}{\lg n} \lg \lg n\right) \text{ bits}$$

Comme le graphe G possède $O(n/\lg n)$ nœuds, une référence à un voisin d'un nœud donné dans G ne coûte que $\lg n + O(1)$ bits. Or puisque le nœud N_j a $O(k_j)$ voisins, la quantité totale d'espace mémoire utilisé pour représenter les adjacences entre les nœuds de G est exprimée par $(\lg n + O(1)) \cdot \sum_j k_j = O(n)$. Pour conclure, la complexité (en terme d'espace mémoire) de la représentation compacte décrite ci-dessus peut s'obtenir en sommant les coûts que nous venons d'établir, ce qui donne un coût linéaire de $O(n)$ bits comme annoncé. □

4.2.3 Structure succincte

Dans la preuve du Lemme 37 la partie linéaire du stockage provient de deux contributions : la contribution des références vers le Catalogue $\mathcal{D}$, qui est dominée par l'entropie αn, et celle concernant les relations de voisinage relatives au graphe G.

Le but principal de cette section est de montrer que le deuxième coût, dû à la représentation explicite de G, peut se réduire à un terme sous-linéaire et donc asymptotiquement négligeable.

L'idée principale consiste à représenter explicitement le graphe G avec des références locales dont le coût est $O(\lg \lg n)$ bits, ce qui peut s'obtenir avec la stratégie de décomposition suivante.

Le graphe G est partitionné en sous-graphes connexes appelés *mini* morceaux, obtenus en regroupant environ $\Theta(\lg n)$ micro morceaux adjacents.

Plus précisément, supposons que nous puissions construire un graphe F, obtenu en fusionnant plusieurs nœuds du graphe G en un seul nœud de F : deux tels nœuds seront donc voisins dans F si et seulement s'il existe un arc dans G entre au moins deux de ses éléments.

L'ensemble des nœuds de F est $\{N'_1, N'_2, \dots N'_{p'}\}$, $|N'_i|$ désigne le nombre de nœuds de G qui ont été fusionnés pour obtenir N'_i et $deg'(N'_i)$ est le degré du nœud N'_i dans le graphe F.

Un nœud N'_i de F contient donc les informations suivantes :

- $|N'_i|$ et $deg'(N'_i)$
- l'information concernant tous les nœuds $N_j \in N'_i$, stockée dans une unique zone de mémoire.
- les références aux voisins de N'_i dans F.

En outre, le graphe F doit satisfaire les hypothèses suivantes :

Hypothèse 4. *Le nombre de micro morceaux qui sont regroupés dans chaque mini morceau N'_i satisfait :* $\frac{1}{3} \lg n \leq |N'_i| \leq \lg n$.

Le nombre d'arêtes incidentes à un nœud donné de F peut être borné en calculant la somme des degrés sur les nœuds :

$$deg'(N'_i) \leq \sum_{N_j \in N'_i} deg(N_j) = \sum_{N_j \in N'_i} k_j \leq |N'_i| K c \lg n = O(\lg^2 n)$$

(où K est la constante qui décrit la taille du bord des micro morceaux, introduite à l'équation (4.1)).

Maintenant il nous faut une hypothèse plus forte sur le nombre total d'arcs du graphe F.

Hypothèse 5. *Le nombre d'arcs du graphe F dépend de manière linéaire du nombre de ses sommets (l'hypothèse 4 et la façon dont on regroupe les nœuds de G, nous assurent que le nombre de nœuds de F est $O(\frac{n}{\lg^2 n})$) :*

$$deg'(N'_1) + \ldots + deg'(N'_{p'}) = O\left(\frac{n}{\lg^2 n}\right).$$

Les hypothèses précédentes nous permettent d'affirmer que le coût en mémoire du graphe G est asymptotiquement négligeable, comme établi par le Lemme suivant :

Théorème 38. *Sous les hypothèses 1, 2, 3, 4 et 5 le graphe G peut être stocké en utilisant $\alpha n + O\left(\frac{n \lg\lg n}{\lg n}\right)$ bits.*

Démonstration. Observons d'abord que dans la représentation du graphe G (Lemme 37) donnée ci-dessus un nœud N_j n'utilise que $O(\lg^2 n)$ bits ($O(\lg n)$ bits pour la référence du micro morceau M_j dans le catalogue $\mathcal{D}$, et $O(\lg n)$ bits pour chacun de ses voisins, qui sont au plus en nombre de $Kc\lg n$), ce qui fournit facilement une borne de $O(\lg^3 n)$, sur la taille de la mémoire utilisée pour stocker l'information codant tous les nœuds de G qui se fusionnent en un seul même nœud N'_i.

Cette remarque est fondamentale pour garantir qu'une référence locale vers un nœud $N_j \in N'_i$ (un pointeur vers la zone de mémoire correspondante de N'_i) nécessite au plus $\lg\lg n + O(1)$ bits.

Nous allons modifier l'information associée au nœud $N_j \in N'_i$ du graphe G. Pour se référer à un nœud voisin N_l de N_j, au lieu d'utiliser un adressage dans l'espace mémoire tout entier réservé à G, nous utilisons d'abord une référence au nœud $N'_k \ni N_l$, et ensuite une référence à l'adresse de N_l, dans la partie de mémoire réservée aux éléments de N'_k. Pointer vers le nœud N'_k se fait de manière indirecte, à l'aide de son index dans le tableau stockant les voisin du nœud N'_i, qui sont au plus en nombre de $O(\lg^2 n)$. Une référence vers un voisin N_l de N_j coûte donc au plus $O(\lg\lg n)$ bits.

L'analyse de la taille en mémoire du graphe G entier est basée sur un argument similaire à celui utilisée dans le lemme 37 : ici le coût d'une référence vers un voisin est réduit de $O(\lg n)$ bits à seulement $O(\lg\lg n)$, ce qui garanti un coût sous-linéaire pour G.

Le coût additionnel de la représentation de F est sous-linéaire puisque le graphe ne possède que $O(\frac{n}{\lg^2 n})$ nœuds (pour l'Hypothèse 4) et arcs (Hypothèse 5), nécessitant chacun $O(\lg n)$ bits. □

4.2.4 Planarité locale

Le schéma général décrit ci-dessus s'applique bien au cas des arbres : un arbre à n sommets peut se décomposer récursivement en micro arbres de taille logarithmique, en considérant comme cellules du bord les arêtes qui relient les nœuds de micro arbres différents.

De manière plus générale, il est intéressant d'observer que les hypothèses 1, 2, 3, 4 et 5 sont naturellement satisfaites lorsque nous considérons l'information combinatoire de maillages surfaciques (correspondants aux cartes plongées sur des surfaces).

Comme exemple, nous verrons aux chapitres suivants qu'il est possible de partitionner une carte à n faces en micro régions, à l'aide d'une décomposition d'un arbre couvrant (du graphe dual, ou parfois du graphe primal) en micro arbres. Ces micro régions (obtenues par une opération de regroupement local, à partir des micro arbres) forment ainsi une décomposition $(M_1, \ldots, M_p)$

qui satisfait les conditions suivantes :

- chaque micro région contient entre $\frac{c}{3} \lg n$ et $c \lg n$ faces ;
- chaque face appartient exactement à une et une seule micro région ;
- les arêtes du bord des micro région sont regroupées par *côtés* (suites d'arêtes séparant deux micro régions différentes) ;
- la borne sur le nombre d'arcs du graphe G décrivant les adjacences entre côtés des micro cartes dérive de la formule d'Euler. Ainsi la planarité locale des structures considérées garantit que le graphe G et F sont globalement creux et satisfont les hypothèses précédentes : ce qui conduit à la validité du théorème 38.

Il est naturel de croire que notre schéma puisse s'appliquer de manière assez générale à une classe de structures satisfaisant des conditions de planarité locale, et qu'il nous permette d'obtenir donc des représentations compactes lorsque le "degré des cellules" est borné (cette hypothèse est d'une certaine manière importante pour assurer la navigation dans le structure en temps constant, comme esquissé dans la section suivante).

Néanmoins, l'obtention de représentations succinctes (optimales) demande plus d'attention, notamment en ce qui concerne la stratégie choisie pour décrire la décomposition en mini et micro morceaux[3].

4.2.5 Requêtes d'adjacence locales en temps constant

Il reste encore à montrer la façon dont la structure à plusieurs niveaux (décrites par les graphes G et F) permet d'implanter de manière efficace certaines requêtes locales d'adjacence (requêtes de voisinage concernant des cellules élémentaires).

Lemme 39. *Considérons un objet M muni d'une décomposition $[(M_1, \ldots, M_p); G; F]$ définie comme auparavant. Si chaque morceau M_j est une carte planaire connexe ayant toutes ses faces de degré borné et un bord de taille arbitraire $O(n_j)$, il est alors possible de tester en temps $O(1)$ si deux éléments de M (sommets ou faces) sont adjacents ou non.*

Démonstration. L'idée principale est que, une fois la décomposition effectuée, les cellules élémentaires sont partagées par au plus un ou deux micro morceaux (selon la façon dont le graphe G est défini), à l'exception d'un petit ensemble de *cellules multiples* partagées par un nombre arbitraire de micro morceaux.

[3]L'application de notre schéma au cas des triangulations munies d'un arbre couvrant quelconque nous obligera à considérer le catalogue $\mathcal{D}_{m,k}$ contenant toutes les triangulations planaires à m faces et avec un bord simple subdivisé en k côtés. Le théorème 38 conduit alors à une représentation qui est compacte pour les triangulations de la sphère, et succincte seulement pour la classe des triangulations avec un bord (les entropies des deux classes étant différentes comme mentionné à la section 2.5).

La planarité du graphe G assure que le nombre de cellules multiples est au plus $O(\frac{n}{\lg n})$: ce qui implique qu'il est possible de répondre aux requêtes d'adjacences concernant les cellules multiples à l'aide d'une information additionnelle, dont le stockage ne nécessite qu'une quantité globalement négligeable d'espace mémoire auxiliaire.

Intuitivement, les requêtes locales concernant des cellules appartenant à un seul micro morceau M_i peuvent être traitées à l'aide de l'information stockée dans la représentation explicite du micro morceau, permettant une navigation en temps $O(1)$).

Sans rentrer dans les détails (qui dépendent du type de requête et de cellule prises en considération et qui seront fournis aux chapitres 5 et 7), nous nous limitons à souligner ici que le cas de cellules multiples appartenant au bord d'un micro morceaux peut se traiter en ajoutant de l'information au graphe G (à ses arcs ou à ses nœuds) et éventuellement au graphe F.

Il faut aussi considérer avec précaution le fait qu'une cellule partagée par un nombre arbitraire de micro morceaux peut ne pas avoir un nombre constant de représentations : de toute manière le nombre de ces cellules est négligeable sur l'ensemble de tous les micro morceaux, ce qui permet de les détecter et de les gérer au niveau du graphe F. □

4.2.6 Attacher de l'information aux cellules élémentaires

Le schéma de représentation décrit dans ce chapitre ne permet de traiter que l'information de connectivité d'un objet combinatoire. Puisqu'il s'agit de représentations conçues pour des objets géométriques, il semblerait naturel de vouloir traiter d'autre type d'information, telle que les coordonnées des sommets ou d'autre attributs associés aux cellules élémentaires de l'objet (couleurs des faces, ...).

Ce problème peut se résoudre par l'enrichissement de la structure à l'aide d'une information supplémentaire associée aux nœuds de la carte G. En particulier, il est possible d'ajouter au nœud N_j la liste des données géométriques associées aux cellules du micro morceau M_j. Plus précisément, cette liste contient les données associées aux cellules internes de M_j et d'un sous-ensemble des cellules constituant les côtés, de telle façon que les cellules (non multiples) appartenant à deux micro morceaux voisins ne sont stockées qu'une seule fois.

Il s'avère que les cellules partagées par plusieurs micro morceaux (cellules multiples) peuvent présenter plusieurs copies : en réalité cela n'affecte pas de manière significative la taille de la représentation, puisque cet événement n'arrive que "rarement". Plus précisément, on établit d'abord la liste des cellules multiples partagées par les micro morceaux M_j : on peut distinguer entre les cellules multiples partagées par les micro morceaux relatifs à un même nœud N'_i, et celles qui sont en commun avec des micro morceaux relatifs à différents nœuds du graphe F. Ces dernières sont très peu nombreuses, environ $\Theta(\frac{n}{\lg^2 n})$: le coût du stockage des données géométriques associées à ces cellules est négligeable (on suppose que les données externes puissent se stocker sur $O(\lg n)$ bits chacune). Pour ce qui est des autres cellules multiples, leur nombre est globalement $O(\frac{n}{\lg n})$ mais la façon dont on a regroupé et fusionné les nœuds du graphe G nous garantit que les cellules de ce type relatives à un même nœud N'_i sont au plus $O(\lg^2 n)$: ainsi une référence à une de ces cellules ne coûte que $O(\lg \lg n)$ bits. Il suffit alors de stocker, pour chaque nœud N'_i, la liste des données géométriques relatives à une sélection de ces cellules multiples, de telle manière que pour chaque cellule on ne stocke qu'une seule fois ses données externes associées. Le même argument utilisé dans l'analyse du caractère succinct de la représentation de G suffit à garantir que le nombre global de copies concernant de telles cellules est négligeable

4.2.7 Construction de la représentation en temps linéaire

Pour conclure la présentation de notre schéma, il reste à mentionner quelques hypothèses supplémentaires permettant la construction en temps linéaire $O(n)$ d'une représentation succincte.

En particulier nous supposons, étant donné un objet $M \in \mathcal{C}_n$ de taille n, de pouvoir calculer un code binaire pour M dont la longueur est au plus $O(\lg |\mathcal{C}_n|)$, et cela en temps $O(n)$. Cette hypothèse[4] est nécessaire afin de garantir que les références et pointeurs manipulés par notre structure sont tous de taille $w = \Theta(\lg n)$: rappelons que dans le modèle de machine *word-RAM* que nous avons adopté la manipulation d'une référence ou d'un nombre de cette taille s'effectue en temps $O(1)$.

[4]Il est à noter que nous ne demandons pas nécessairement de disposer d'un codage optimal pour les objets de la classe $\mathcal{C}_n$, au moins pour ce qui est de la construction de la décomposition et du catalogue $\mathcal{D}$.

Chapitre 5

Une représentation compacte pour les triangulations

5.1 Préliminaires

Ici nous considérons le problème de concevoir une représentation compacte de l'information de connectivité d'une 2-surface triangulée, qui soit à la fois un codage et une structure de données supportant efficacement l'accès à l'information et la navigation dans la triangulation.

5.1.1 Notre contribution

Étant donnée une triangulation ayant m triangles, la structure que nous allons proposer n'utilise que $2.175m+O\left(m\frac{\lg\lg m}{\lg m}\right)$ bits et permet l'accès d'un triangle à une face voisine en temps $O(1)$ (d'autres opérations de navigation locale dans la triangulation sont aussi disponibles, toutes nécessitant un temps constant dans le pire des cas)[1].

La quantité d'espace mémoire utilisée est prouvée asymptotiquement optimale pour la classe des triangulations planaires avec un bord, étant égale au premier ordre à l'entropie de cette classe, qui correspond à 2.175 bits par face (voir la section 5.6).

Notre approche s'étend directement au cas des triangulations à m faces d'une surface de genre g. Dans ce cas l'espace occupé par la structure est de $2.175m + 36(g-1)\lg m + O\left(m\frac{\lg\lg m}{\lg m} + g\lg\lg m\right)$ bits, ce qui reste asymptotiquement optimal pour des surfaces de genre $g = o(m/\lg m)$.

Pour le cas $g = \Theta(m)$ la représentation reste valide, bien que le terme dominant qui exprime l'espace mémoire utilisé (toujours calculable explicitement), devienne du même ordre de grandeur que le coût d'une représentation explicite basée sur des pointeurs.

[1]Dans une présentation légèrement différente, hors du cadre général défini au chapitre précédent, les résultats de ce chapitre ont été publiés à WADS 2005 [27].

Différences avec les approches précédentes

D'un point de vue général, le schéma adopté dans ce travail a des similarités avec celui proposé pour les arbres binaires dynamiques dans [94, 106] : une décomposition en sous régions de petite taille de la structure initiale (dans notre cas une triangulation) et une description hiérarchique à deux niveaux d'une telle décomposition.

Néanmoins la manière dont une structure géométrique telle qu'une triangulation peut être décomposée, est plus compliquée et mérite une certaine attention : surtout en ce qui concerne le regroupement des micro régions adjacentes et la description de leurs relations de voisinages.

Notamment le fait de ne pas imposer des contraintes sur la taille et la complexité des sous triangulations (qui restent planaires et avec un bord) implique que notre représentation ne sera optimale que pour la classe des triangulations avec bord : le terme asymptotiquement dominant est $2.175m$, qui correspond précisément à l'entropie (par unité de taille) de cette classe de cartes.

5.1.2 Opérations et requêtes sur la triangulation

Comme dans les travaux existants concernant les représentations succinctes et compactes conçues pour les arbres et les graphes, nous allons considérer un ensemble d'opérations qui permettent d'accéder localement à l'objet représenté : vu l'intérêt que nous portons aux structures de données géométriques, nous allons considérer principalement les opérations suivantes, toutes implantables en temps $O(1)$, qui permettent une navigation standard entre faces d'une triangulation :

- $Triangle(v)$: retourne un triangle incident au sommet v ;
- $Index(\triangle, v)$: retourne l'index (compris entre 0 et 2) du sommet v dans le triangle $\triangle$;
- $Voisin(\triangle, v)$: retourne le triangle adjacent à $\triangle$ et opposé au sommet $v \in \triangle$;
- $Sommet(\triangle, i)$: retourne le sommet du triangle $\triangle$ ayant index i ($0 \leq i \leq 2$).

Il est intéressant de remarquer la différence entre l'ensemble d'opérations proposé ici et celui adopté dans d'autres travaux sur les graphes (déjà présentés dans le chapitre 3) : ces derniers mettent l'accent sur le test d'adjacence entre sommets, opération moins usuelle en géométrie algorithmique, qui n'est pas implantable directement en temps $O(1)$ en utilisant seulement les opérations proposeés ci-dessus.

Cependant il s'avère toutefois possible avec notre structure de gérer en temps $O(1)$ ce dernier test (adjacence entre sommets), ainsi que d'autres requêtes locales (degré d'un sommet), nécessitant normalement un coût proportionnel au degré des sommets, même avec des représentations explicites standards en géométrie [15]. Le fait que ces types de requêtes ne dépendent que d'une information locale (telle que celle contenue dans le voisinage d'un sommet dans un graphe) et la planarité locale des objets à représenter jouent un rôle essentiel dans la conception de stratégies efficaces pour ces requêtes locales (un aperçu de leur implantation sera donné dans la section 5.5).

5.2 Application du schéma général

Nous allons maintenant montrer comment le schéma décrit au chapitre 4 pour la conception de représentations succinctes s'applique au cas de structures de données géométriques, et en particulier aux triangulations. La triangulation initiale est notée $\mathcal{T}$ et son paramètre de taille est m (où m est le nombre de triangles de la triangulation). Lorsque la triangulation $\mathcal{T}$ n'est pas planaire, son genre est noté g.

5.2.1 Vue d'ensemble : esquisse

Dans notre cas, la stratégie suivie consiste à décomposer la triangulation initiale en sous triangulations (*mini triangulations*) ayant environ $\Theta(\lg^2 m)$ triangles, chacune se décomposant en sous-triangulations planaires de taille $\Theta(\lg m)$(*micro triangulations*) : cette décomposition en micro triangulations constitue une partition de l'ensemble initial des triangles de $\mathcal{T}$.

Structures à plusieurs niveaux

Une fois obtenue une décomposition de l'objet initial $\mathcal{T}$ en sous régions satisfaisant les hypothèses du chapitre 4, il ne reste qu'à construire une structure hiérarchique à trois niveaux.

Le premier niveau est donné par un graphe reliant les $\Theta\left(\frac{m}{\lg^2 m}\right)$ mini triangulations, ou plus précisément une carte, puisque l'ordre cyclique des voisins autour d'une mini triangulation compte. Cette carte est représentée de manière classique à l'aide de pointeurs de taille $O(\lg m)$, et étant donné le nombre de ses nœuds, son stockage nécessite des ressources mémoire sous-linéaires.

Le deuxième niveau est formé d'une carte reliant les micro triangulations. Les nœuds de cette carte sont regroupés en respectant leur appartenance aux mini triangulations, ce qui permet d'utiliser des pointeurs locaux de taille $O(\lg \lg m)$ pour stocker les relations d'adjacence entre micro régions.

L'information concernant une micro triangulation n'est pas explicitement décrite au deuxième niveau : nous stockons simplement une référence (de taille variable) vers un troisième niveau, constitué par un catalogue exhaustif $\mathcal{D}$ contenant toutes les différentes triangulations à bord colorié (avec k sommets noirs) de taille au plus $\Theta(\lg m)$. Les sommets du bord d'une micro triangulation, partagés par deux ou plusieurs micro triangulations, sont

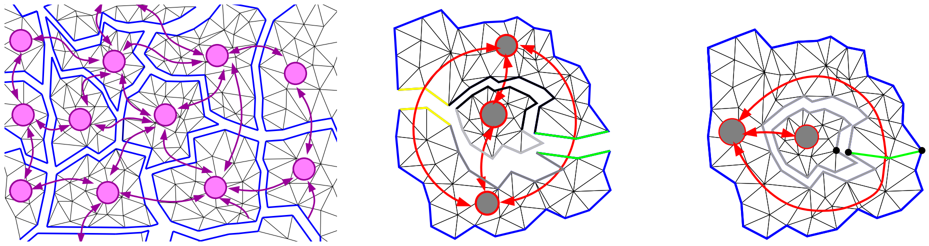


FIG. 5.1 – Décomposition en micro triangulations, avec la carte G correspondante. G peut contenir des arêtes multiples et des boucles (les côtés correspondant des micro triangulations adjacentes sont colorées avec la même couleur).

coloriés de manière à tenir compte de ce partage (blanc pour 1 ou 2, noir pour plus que 2). Grâce à l'additivité de la décomposition (Hypothèse 1) la taille globale de tous ces références vers le catalogue donne le terme dominant $2.175m$, tandis que les contributions de tous les autres informations stockées sont asymptotiquement négligeables.

Micro triangulations

Les *mini triangulations*, de taille entre $\frac{1}{3}\lg^2 m$ et $\lg^2 m$, sont notées $\mathcal{ST}_i$, tandis que les *micro triangulations*, ayant entre $\frac{1}{12}\lg m$ et $\frac{1}{4}\lg m$ triangles, sont notées $\mathcal{TT}_j$. Toutes les micro triangulations prises en considération seront des triangulations planaires ayant un seul bord simple (de taille arbitraire). Vues comme étant des sous-triangulations de $\mathcal{T}$, ces micro triangulations pourraient être de genre supérieur et avoir plusieurs bords : cette contradiction est résolue en autorisant une arête de $\mathcal{T}$ à apparaître plusieurs fois sur le bord de $\mathcal{TT}_j$ (voir la figure 5.1). Plus précisément, une arête du bord peut être partagée par deux micro triangulations différentes, ou bien peut apparaître deux fois sur le bord d'une même micro triangulation (ce qui se produit lorsque le bord de la micro triangulation n'est pas simple). Un côté d'une micro triangulation $\mathcal{TT}_j$ est un ensemble maximal d'arêtes du bord (consécutives et entre deux sommets noirs) partagées par $\mathcal{TT}_j$ et une même micro triangulation $\mathcal{TT}_{j'}$ (avec éventuellement $j' = j$). De cette manière les bords des micro triangulations sont partitionnés en un ensemble cyclique de côtés.

Cartes G et F

Les relations d'adjacence entre les mini triangulations voisines $\mathcal{ST}_i$ sont décrites par un graphe appelé F, tandis que celles entre les micro triangulations $\mathcal{TT}_j$ seront représentées par un graphe G.

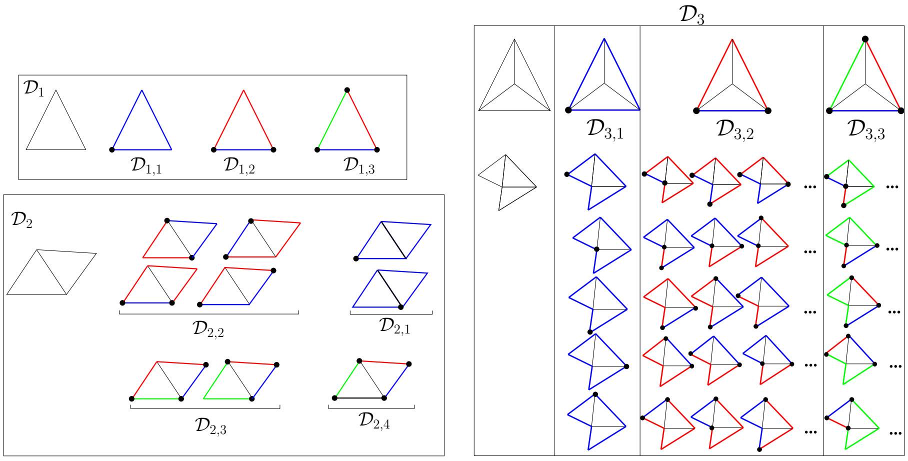


FIG. 5.2 – Catalogue exhaustif $\mathcal{D}$ des micro triangulations : ces images montrent les micro triangulations coloriées appartenant aux sous-catalogues $\mathcal{D}_{p,k}$ pour $p \leq 3$, k étant le nombre de côtés du bord (pour des raisons de clarté, la représentation du catalogue $\mathcal{D}_3$ est incomplète, les pointillés indiquant d'autres micro triangulations coloriées).

Le sous-graphe obtenu par la restriction de G aux micro triangulations appartenant à une unique mini triangulation $\mathcal{ST}_i$ sera dénotée par G_i.

Plus précisément, F et G sont des *cartes* (selon la notion donnée au chapitre 2) : leur connectivité sera décrite en stockant, pour chacun de leurs nœuds, la liste des arcs incidents. Ceux-ci seront stockés dans un tableau de manière à tenir compte de l'ordre cyclique des côtés des micro triangulations. Comme on le verra plus précisément dans la suite, les cartes F et G peuvent avoir des boucles (correspondant aux auto-intersections du bord d'une micro triangulation) et des arêtes multiples (lorsque deux micro triangulations partagent plusieurs côtés différents). Nous tenons à souligner que, par construction, ces cartes ne pourront qu'avoir des faces de degré au moins 3, puisqu'il existe seulement un arc pour chaque côté des micro triangulations : cette remarque, associée à la planarité locale des cartes F et G se révélera fondamentale dans l'analyse de la taille mémoire de notre structure. Dorénavant, afin d'éviter des confusions dans nos notations, nous utiliserons les termes *arcs* et *nœuds* pour indiquer les arêtes et sommets des cartes F et G, alors que nous maintiendrons les mots *sommets* et *arêtes* pour indiquer les cellules élémentaires de $\mathcal{T}$.

5.2.2 Catalogue des micro-régions

Les objets auxquels nous nous intéressons appartiennent à la classe $\mathcal{C} = T_p^b$ des triangulations planaires ayant un bord de taille arbitraire, le paramètre p indiquant la taille des triangulations en terme du nombre de faces. Comme détaillé à la section 5.6 cette classe a une entropie linéaire $\lg|T_p^b| = 2.175p + o(p)$.

Notre stratégie consiste à décomposer un objet de $\mathcal{C}$ en sous-triangulations faisant partie d'un catalogue $\mathcal{D}$: comme déjà mentionné auparavant nous ne demandons pas que ce catalogue $\mathcal{D}$ contienne des éléments de la même classe $\mathcal{C}$.

Dans notre cas, la décomposition que nous produirons de la triangulation T nous suggère de considérer la classe $\mathcal{D} = \{D_{p,k}\}$, où $\mathcal{D}_{p,k}$ contient toutes les triangulations ayant p triangles, dont les arêtes du bord (le bord étant de taille au plus $p+2$) sont partitionnées en k côtés ($1 \leq k \leq p+2$) (quelques exemples de catalogues de petites taille sont montrées dans la figure 5.2).

Il est ainsi facile de voir que les constantes α et β apparaissant dans la formule 4.1 valent respectivement ici 2.175 et 1, puisque[2] :

$$\|\mathcal{D}_{p,k}\| = \lg|\mathcal{D}_{p,k}| = \lg(2^{2.17p} \cdot \binom{p+2}{k}) \leq 2.17p + k\lg p + O(k)$$

Puisque notre décomposition de $\mathcal{T}$ ne contiendra que des micro triangulations de taille entre $\frac{1}{12}\lg m$ et $\frac{1}{4}\lg m$, l'hypothèse 3 (où la constante c vaut ici $\frac{1}{4}$) suffit déjà à nous garantir que le catalogue $\mathcal{D}$ (le stockage de ses éléments) nécessite une quantité négligeable de ressource mémoire, comme exprimé par (la constante α' étant ici égale à 3.17)

$$|\mathcal{D}_p| = \sum_{k=1}^{p+2} |\mathcal{D}_{p,k}| \sim 2^{2.17p} \cdot 2^p \leq 2^{3.17\frac{1}{4}\lg m} = o(m)$$

La description de la structure A stockant l'organisation du catalogue $\mathcal{D}$ ainsi que sa construction en temps $o(m)$ seront données à la section 5.3.1.

5.2.3 Décomposition en sous-régions

Étant donnée une triangulation $\mathcal{T}$ ayant m triangles, il est possible d'obtenir une décomposition $(M_1, \ldots, M_r)$ en micro triangulations (chaque M_j appartenant au catalogue $\mathcal{D}$), ainsi que les cartes correspondantes F et G, de la manière suivante.

- Calculer d'abord un arbre couvrant B du graphe dual $\mathcal{T}^*$: le résultat est un arbre binaire B ayant m nœuds (puisque chaque sommet dans $\mathcal{T}^*$ a degré 3) ;
- appliquer à B l'algorithme de décomposition d'arbres donné dans le lemme 15, avec paramètre $M = \frac{1}{3}\lg^2 m$: le résultat est une partition de B en mini arbres B_i, chacun de taille entre $\frac{1}{3}\lg^2 m$ et $\lg^2 m$.

[2]Un point important à souligner est que $\lg\binom{p}{k} \leq \min(k\lg p, p)$: étant donnée la taille des paramètres k et p, on peut garantir que toutes les références utilisées dans nos travaux sont de taille au plus $O(\lg m)$ (on peut donc les manipuler en temps $O(1)$).

– appliquer à nouveau à chaque B_i l'algorithme de décomposition cité ci-dessus, avec paramètre $M = \frac{1}{12} \lg m$: cette fois nous avons une sous partition de B en micro arbres, chacun de taille entre $\frac{1}{12} \lg m$ et $\frac{1}{4} \lg m$.

Les micro (respectivement mini) triangulations peuvent s'obtenir en regroupant les triangles qui appartiennent au même micro (respectivement mini) arbre : rappelons que chaque nœud dans l'arbre correspond à un triangle et chaque arête dans l'arbre est la duale d'une arête interne de la triangulation.

Dans la suite, nous allons illustrer la stratégie adoptée pour la construction des bords des micro (resp. mini) triangulations et des relations d'adjacences décrites par les cartes G (resp. F).

Considérons l'ensemble de triangles constituant un arbre de triangles, qui correspond à un micro arbre obtenus avec le procédé ci-dessus.

• marquer comme *internes* toutes les arêtes qui sont le dual d'une certaine arête dans l'arbre ;

• marquer comme *arêtes du bord* celles qui appartiennent à au moins un triangle dans l'arbre (les arêtes qui sont partagées par triangles contenus dans des sous-arbres différents) ;

• s'il existe une extrémité telle que toutes les arêtes incidentes sont internes, à l'exception d'une seule qui n'est pas encore classifiée, alors marquer cette dernière comme interne. Répéter cette étape tant que possible.

Les arêtes restantes non marquées sont classifiées comme appartenant au bord.

De cette manière, une micro triangulation est une carte connexe par faces et toujours simplement connexe, bien que son bord puisse ne pas être simple, en présentant des auto-intersections. Dans ce cas, la triangulation est incidente à elle même, ce qui se traduit par une boucle dans la carte G (respectivement F).

Construction des cartes G et F

Une fois obtenue la décomposition des mini triangulations en micro triangulations, il nous reste à construire la carte G en associant à chaque micro arbre un nœud de G, et en créant un arc dans G pour chaque côté d'une micro triangulation. La carte G est simplement la carte duale de la carte dont les faces sont bornées par les arêtes du bord des micro triangulations, ayant un arc pour chaque côté de ces bords. Les arêtes multiples, ainsi que les boucles, sont admises dans la carte G : il est important de souligner que les faces ont toutes degré au moins 3.

Si la carte G possédait une face f de degré 2, alors f devrait être bornée par deux arcs multiples incidents à deux nœud n_1 et n_2 de G, correspondant à deux micro triangulations adjacentes $\mathcal{TT}_1$ et $\mathcal{TT}_2$. Puisque, par construction, il existe un seul arc dans G pour chaque côté de micro triangulations, ces deux arcs multiples devrait correspondre à deux côtés (suites d'arêtes du bord) consécutives partagées par $\mathcal{TT}_1$ et $\mathcal{TT}_2$: il est donc possible de fusionner ces deux côtés en un seul, et contracter ainsi la face f en un seul arc simple (n_1, n_2).

Cette dernière propriété, ainsi que la planarité locale de G, est à la base de l'argument utilisé dans la preuve du lemme 50 concernant l'analyse du stockage de notre structure.

La construction de la carte F s'effectue de manière similaire.

5.2.4 Vérification des hypothèses

Nous allons ici brièvement montrer comment la décomposition $[\mathcal{TT}_1, \ldots, \mathcal{TT}_l); G; F]$ de la triangulation initiale satisfait les hypothèses énoncées au chapitre 4.

L'hypothèse 1 est facilement vérifiée, puisque les micro triangulation de la décomposition ne partagent aucune face (le paramètre de taille m comptant ici le nombre de faces), ce qui s'exprime par $\sum_j n_j = m$.

Le nombre de côtés des micro triangulations intervenant dans la décomposition (hypothèse 2) est strictement lié aux propriétés de la carte G. Comme déjà mentionné, la carte G peut présenter des arêtes multiples et même des boucles, le degré de ses faces restant toujours ≥ 3. Ces remarques, ainsi que la planarité de G (d'après le corollaire 8), nous garantissent que le nombre d'arcs de G (et donc le nombre de côtés des micro triangulations) est linéairement borné par le nombre de ses nœuds, ce qui s'exprime par : $\sum_j k_j = O(\frac{m}{\lg m})$. Les hypothèses 3 et 4 concernant la taille des micro et mini triangulations sont simplement garanties par le procédé de construction de la décomposition décrit à la section précédente. Et finalement, à la carte F s'appliquent les mêmes arguments utilisés pour G (planarité, absence de faces de degré moins que 3) qui conduisent à une borne sur le nombre de ses arcs (hypothèse 5).

Le lemme 38 donne presque directement une représentation succincte pour la classe des triangulations planaires à bord. De plus, une telle représentation reste valide et succincte lorsqu'il s'agit de coder une triangulation d'une surface de genre g fixé (une version détaillée du lemme 38, spécialisée au cas des triangulations de genre g à bord sera à la section 5.3.3).

5.3 Description et analyse des structures de données

Nous allons maintenant fournir une description détaillée des structures de données constituant notre représentation : la structure A stockant le catalogue $\mathcal{D}$ des micro triangulations, et les représentation des cartes G et F. Notre intention est de donner une version précise et spécialisée des descriptions esquissées au chapitre 4, avec plusieurs buts : aider d'une part le lecteur dans la compréhension de la stratégie de navigation ; d'autre part établir un évaluation explicite et précise de la complexité en mémoire de notre représentation.

5.3.1 Catalogue exhaustif des micro triangulations

Toutes les différentes triangulations (planaires et avec un bord simple) ayant i triangles (avec $i \leq \frac{1}{4}\lg m$) sont engendrées et stockées dans un tableau A, chacune munie d'une représentation explicite (celle-ci étant une représentation par pointeurs basée sur les triangles comme dans [15]).

Les éléments de A sont regroupés et triés par taille croissante, de manière à pouvoir utiliser des références de taille variable, proportionnelle à la taille de la triangulation correspondante.

Dans la partie restante de cette section nous allons décrire en détail l'organisation de cette structure (voir aussi la figure 5.3) : l'analyse de sa complexité, de l'espace mémoire utilisé et du temps de calcul nécessaire sa construction, se trouvera dans la section 5.4.

Description de la représentation en mémoire

• A est un tableau de dimension 2, dont le élément $A[i,k]$ ($i \leq \frac{1}{4}\lg m$, $k \leq \frac{1}{4}\lg m + 2$) est un pointeur vers un tableau $A_{i,k}$.
• $A_{i,k}$ est un tableau contenant toutes les différentes triangulations ayant exactement i triangles, et dont les arêtes du bord sont partitionnées en k côtés. Son j-ème élément est un pointeur vers une représentation explicite $A_{i,k,j}^{explicite}$ de la triangulation $A_{i,k,j}$.
• $A_{i,k,j}^{explicite}$ contient trois champs :

— $A_{i,k,j}^{explicite}.sommets$ est le tableau des sommets de $A_{i,k,j}$. Chaque sommet contient l'index d'un triangle incident, dans le tableau $A_{i,k,j}^{explicite}.triangles$ ci-dessous. Par convention, les sommets du bord apparaissent en premier dans ce tableau, triés selon l'orientation directe (ccw) du bord. Pour tout sommet v du bord, le triangle incident stocké doit être celui incident à l'arête du bord qui a comme extrémités v et son successeur w (sur le bord).

— $A_{i,k,j}^{explicite}.triangles$ est un tableau contenant une représentation des faces de $A_{i,k,j}$. Chaque triangle contient les indices (dans $A_{i,k,j}^{explicit}.sommets$) des sommets incidents et des triangles voisins (indices dans $A_{i,k,j}^{explicit}.triangles$). Ces indices peuvent être *nul*, lorsqu'un triangle est sur le bord.

— $A_{i,k,j}^{explicite}.rank\_select$ est la représentation implicite du coloriage du bord de la micro triangulation, muni d'une structure de *Rank/Select* : ce coloriage est représenté par un bit-vecteur[3] de poids k et longueur au plus

[3] Ce bit-vecteur décrit la manière dont le bord d'une micro triangulation se décompose en côtés : le n-ème bit est '0' si le n-ème sommet du bord est à l'intérieur d'un côté, ou bien est '1' si ce sommet est à l'extrémité et sépare donc deux côtés.

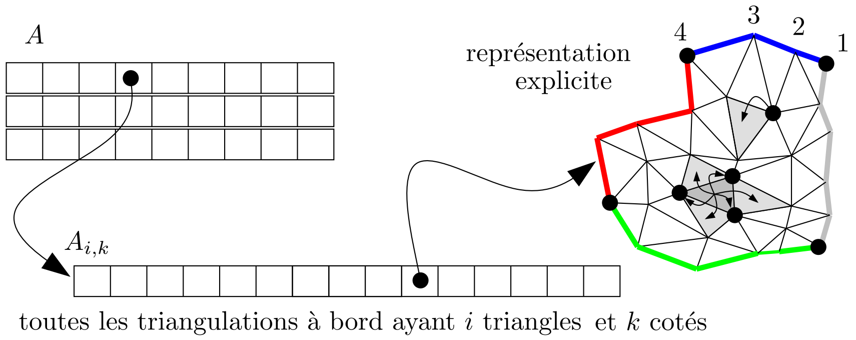


FIG. 5.3 – Stockage de toutes les triangulations de taille au plus $\frac{1}{4}\lg m$.

$i+2$, qui est codé implicitement à l'aide d'un pointeur vers un élément contenu dans la structure B (voir section 6.3).

Analyse de l'espace utilisé

Lemme 40. *Le stockage du tableau A, et de toutes l'information associée aux tableaux $A_{i,k}$ nécessite asymptotiquement $o(m)$ bits.*

Démonstration. • A est un tableau de taille $\frac{1}{4}\lg m \times (\frac{1}{4}\lg m + 2)$, contenant des pointeurs de taille $\lg m$ et donc son coût est de $O(\lg^3 m)$ bits.
• Suivant les résultats d'énumération énoncés dans la section 5.6, $A_{i,k}$ est un tableau contenant au plus $2^{2.175i} \cdot \binom{i+2}{k} \leq 2^{3.175\frac{1}{4}\lg m}$ références, chacune sur $\lg m$ bits, donc le stockage de A_i nécessite au plus $O(2^{3.175\frac{1}{4}\lg m}\lg m)$ bits.
• Par rapport à la représentation explicite d'une triangulation $A^{explicit}_{i,k,j}$:

— $A^{explicit}_{i,k,j}.vertices$ et $A^{explicit}_{i,k,j}.triangles$ sont deux tableaux de taille au plus $i \leq \lg m$. Chaque élément est constitué de plusieurs index, dont la valeur est inférieure à i, donc représentable avec au plus $\lg\lg m$ bits. Enfin, $A^{explicite}_{i,k,j}.rank\_select$ est un pointeur sur $\lg m$ bits.

Donc la taille totale d'une représentation explicite $A^{explicit}_{i,k,j}$ est de $O(\lg m \lg\lg m)$ bits, et le coût total de toutes les $A^{explicit}_{i,k,j}$ associées à un tableau $A_{i,k}$ est au plus $O(2^{3.175\frac{1}{4}\lg m}\lg m \lg\lg m)$ bits.

Pour conclure, la taille totale de l'espace mémoire utilisé pour stocker toutes les micro triangulations s'obtient en faisant la somme sur les différents valeurs de i, ce qui donne un coût final de $O(2^{3.175\frac{1}{4}\lg m}\lg^2 m \lg\lg m) = o(m)$ bits. □

5.3.2 Structures de *Rank/Select*

Comme déjà mentionné, nous utilisons un marquage des sommets du bord des micro triangulations pour tenir compte de leurs adjacences : le coloriage d'une micro triangulation dont le bord est de taille p et partitionné en q côté, se fait naturellement à l'aide d'un bit-vecteur de longueur p et poids q (leur définition a été introduite dans la section 2.6.2). Il peut se se rendre nécessaire de repondre efficacement en temps $O(1)$ à certaines requetes concernant les cellules du bord d'une micro triangulation : par exemple retrouver la k-ième arete sur le j-ème côté ; ou retrouver l'index du côté auquel est incidente une cellule donnée. On peut repondre facilement à ces requetes à l'aide d'une structure de *Rank/Select* sur le bit-vecteur associé au bord. Pour la réalisation d'une implantation efficace de *Rank/Select* nous allons utiliser un catalogue exhaustif de tous les bit-vecteurs de taille p et poids q. La taille des paramètres p et q nous garantit qu'un tel catalogue (ainsi que le tableau B contenant les représentations explicites de chaque bit-vecteur) ne nécessite asymptotiquement que de $o(m)$ bits. Cela nous évite à avoir recours aux structures plus sophistiquées conçues pour représenter de manière succincte cette sorte d'objets (mentionnées dans la section 2.6.2). La partie restante de cette section est dédiée à la description et analyse de notre structure de *Rank/Select*.

Représentation des vecteurs de bits

• B est un tableau de dimension 2 de taille $(\frac{1}{4}\lg m + 2) \times (\frac{1}{4}\lg m + 2)$: chaque entrée $B(p,q)$ est un pointeur vers un tableau B_{pq}.
• B_{pq} est un tableau dont la k-ème entrée est associée au k-ème bit-vecteur de taille p et poids q et contient un pointeur vers une structure B^{RS}_{pqk} (celle-ci permettant d'implanter les opérations de Rank/Select en temps constant).
• B^{RS}_{pqk} est un tableau de taille p dont les entrées sont constituées de deux champs, chacun contenant respectivement les résultats précalculés des opérations $Rank_1$ et $Select_1$:

— $B^{RS}_{pqk}(i).rank$ est le nombre '1' qui précèdent le i-ème bit.

— $B^{RS}_{pqk}(i).select$ est un entier indiquant la position du i-ème '1' dans le bit-vecteur.

Analyse du stockage

Lemme 41. *Le stockage du Tableau B, et de toute l'information associée aux Tableaux B_{pqk} nécessite asymptotiquement $O(m^{\frac{1}{4}} \lg m \lg\lg m)$ bits.*

Démonstration.
- B est un tableau contenant environ $(\frac{1}{4}\lg m)^2$ pointeurs, chacun de taille $\lg m$, et nécessite donc de $O(\lg^3 m)$ bits.
- B_{pq} est un tableau contenant $\binom{p}{q}$ pointeur de taille $O(\lg m)$.
- $B^{RS}_{pqk}(i).rank$, $B^{RS}_{pqk}(i).select$ sont tous des entiers plus petits que $\frac{1}{4}\lg m + 2$, nécessitant au plus $\lg\lg m$ bits. La taille mémoire de la structure B^{RS}_{pqk} est donc au plus $O(\lg m \lg\lg m)$.

Pour conclure, la quantité d'espace demandé pour stocker tous les bit-vecteurs de taille (et poids) au plus $\frac{1}{4}\lg m + 2$ est alors donnée par :

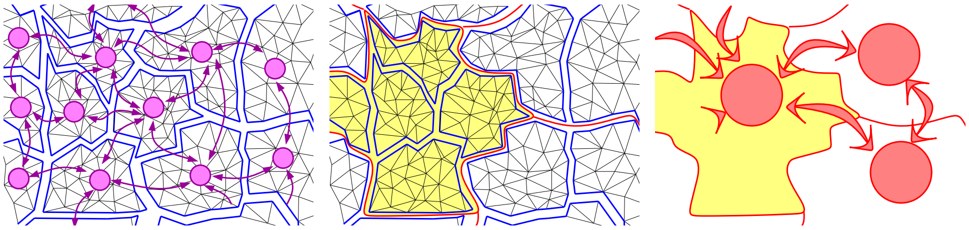


FIG. 5.4 – Décomposition en mini triangulations (triangles dans la région jaune), munie des cartes G (nœuds et arcs violets) et F (nœuds et arcs rouges). Comme pour G, F peut contenir des arêtes multiples et des boucles, ayant des faces de degré au moins 3).

$$\sum_{p,q} \binom{p}{q} O(\lg m \lg \lg m) = \left(\sum_p 2^q \right) O(\lg m \lg \lg m)$$

qui peut se borner par $2^{(\frac{1}{4}\lg m+2)+1} O(\lg m \lg \lg m) = O(m^{\frac{1}{4}} \lg m \lg \lg m))$.
□

5.3.3 Carte des micro triangulations

Nous allons finalement fournir une description détaillée de la carte G décrivant les adjacences entre micro triangulations. L'organisation de l'espace mémoire concernant la représentation de cette carte est basée sur le regroupement des nœuds de G, qui correspondent aux micro triangulations faisant partie d'une même mini triangulation $\mathcal{ST}_i$, dans une sous-carte G_i. Le but principal de cette partition est de garantir l'utilisation de pointeurs locaux de taille "petite" (références entre éléments d'une sous-carte donnée G_i). La phase de construction de G en temps linéaire est donnée dans la section 5.4.3.

Description de la représentation

L'espace mémoire réservé au graphe G est organisé en une séquence de zones de taille variable, chacune dédiée à un sous-graphe G_i. Les ressources mémoires demandée par notre structure, suivent la description fournie ci-dessous :

Dans la zone de mémoire relative à G_i, pour chaque nœud $G_{i,j}$ correspondant à une micro triangulation $\mathcal{TT}_{i,j}$, nous stockons les informations suivantes :

— $G^t_{i,j}$ est le nombre de triangles dans $\mathcal{TT}_{i,j}$.

— $G^b_{i,j}$ est la taille du bord de $\mathcal{TT}_{i,j}$.

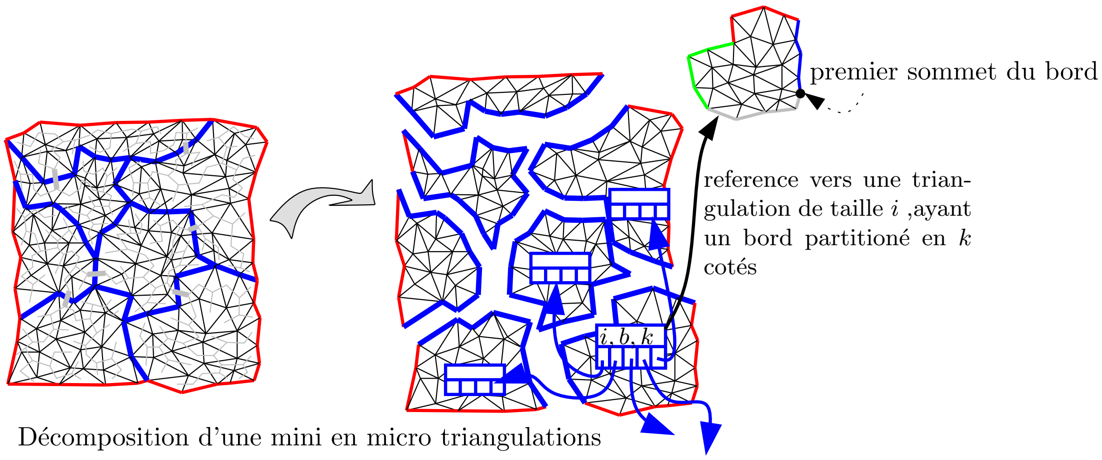


FIG. 5.5 – Sous-carte G_i relative à une mini triangulation

— $G^s_{i,j}$ est le degré du nœud $G_{i,j}$ dans la carte G_i ($G^s_{i,j}$ correspond aussi au nombre de côtés, et donc de voisins, de $\mathcal{TT}_{i,j}$)

— $G^A_{i,j}$ est une référence à la représentation explicite de $\mathcal{TT}_{i,j}$, stockée dans le tableau $A_{G^t_{i,j}, G^s_{i,j}}$.

— Chacun des $G^s_{i,j}$ arcs de G_i qui sont incidents à $G_{i,j}$ est représenté à l'aide d'information supplémentaire (observons que les boucles apparaissent deux fois). En supposant que le $k$ème arc connecte $G_{i,j}$ et un voisin $G_{i',j'}$ dans G, nous stockons les données suivantes :

— $G^{address}_{i,j,k}$ est l'adresse mémoire relative (pointeur local) du premier bit concernant le nœud voisin $G_{i',j'}$, dans la zone de mémoire associée à la mini triangulation $ST_{i'}$ qui le contient.

— $G^{back}_{i,j,k}$ est l'index k' du côté correspondant à l'arc en question, par rapport à la numérotation des côtés de la micro triangulation associée au nœud $G_{i',j'}$.

— $G^{mini}_{i,j,k}$ est l'index de la mini triangulation associée à $G_{i'}$, dans le tableau contenant les voisins de G_i dans la carte F (si $i' = i$ alors cet index est mis à 0).

Analyse de l'espace mémoire utilisé

Le choix de présenter une description détaillée de la représentation nous permet aussi de nous affranchir, au moins dans cette section, de la notation grand-O et de fournir donc des bornes explicites pour toutes les composantes de notre structure de données. Ce qui suit est une version détaillée et spécialisée au cas des triangulations (avec bord et de genre g) du Lemme 38.

Lemme 42. *Le stockage de G nécessite asymptotiquement*

$$2.175m + O(g \lg\lg m) + O\left(m \frac{\lg\lg m}{\lg m}\right) \text{ bits}$$

Démonstration. Pour chaque nœud :

— $G^t_{i,j}$ est plus petit que $\frac{1}{4}\lg m$ et peut donc se stocker sur $\lg\lg m$ bits. $G^b_{i,j}$ et $G^s_{i,j}$) sont au plus $\frac{1}{4}\lg m + 2$ et donc nécessitent $\lg\lg m$ bits (ce qui est valide même pour de très petites valeurs de m).

— $G^A_{i,j}$ est un index dans $A_{G^t_{i,j}}$ qui contient au plus $2^{2.175 G^t_{i,j}} \cdot \binom{G^b_{i,j}}{G^s_{i,j}}$ éléments, en accord avec le résultat d'énumération établi dans la section 5.6. Cette quantité peut donc se stocker sur $2.175G^t_{i,j} + G^s_{i,j}\lg G^b_{i,j}$ bits.

— Le nombre de micro triangulations voisines de $G_{i,j}$ est $G^s_{i,j} < \frac{1}{4}\lg m + 2$. Pour chacune de ces triangulations :

— les références (adressage indirect) $G^{address}_{i,j,k}$ sont stockées sur $K\lg\lg m$ bits (où K est une constante donnée ci-dessous).

— $G^{back}_{i,j,k}$ est au plus $\frac{1}{4}\lg m + 2$ et donc stockable sur $\lg\lg m$ bits.

— $G^{small}_{i,j,k}$ nécessite de $2\lg\lg m$ bits : puisqu'une mini triangulation a au plus $\lg^2 m$ triangles elle contient au plus $\lg^2 m + 2$ arêtes sur le bord, donc le tableau des nœuds adjacents à G_i dans F a au plus $\lg^2 m + 2$ entrées.

Puisque chaque arc est incident au plus à deux nœuds, le coût par arc peut s'évaluer de manière indépendante à $2(K+3)\lg\lg m$ bits. Il reste alors un coût de $3\lg\lg m + 2.175G^t_{i,j} + G^s_{i,j}\lg G^b_{i,j}$ pour chaque nœud $G_{i,j}$ de G_i.

Le nombre de nœuds est au plus $12\lg m$ et le nombre d'arcs (incluant les arcs incidents à un autre nœud $G_{i'}$) est borné par le nombre d'arêtes de la triangulation $\mathcal{T}$ incidentes à des triangles de $\mathcal{ST}_i$, qui est au plus $\lg^2 m$.

Le coût de l'information stockée relative à G_i est donc $C_i \le 2(K+3)\lg^2 m \lg\lg m + 12\lg m(3\lg\lg m + 2.175\frac{1}{4}\lg m + \frac{1}{4}\lg m \lg\lg m)$. En prenant $K = 5$, nous avons $\lg C_i < K\lg\lg m$ pour tout $m \ge 2$, en accord avec l'hypothèse faite ci-dessous concernant le stockage des références $G^{address}_{i,j,k}$.

Le coût global de la carte G entière est obtenu en prenant la somme sur i, j (sur toutes les micro triangulations) :

$$\begin{aligned}
&\sum_i \sum_j \Big(3\lg\lg m + 2.175G^t_{i,j} + G^s_{i,j}(\lg G^b_{i,j}) + G^s_{i,j} \cdot 8\lg\lg m\Big) \\
\le\ & 2.175 \sum_{i,j} G^t_{i,j} + 9\lg\lg m \sum_{i,j} G^s_{i,j} + 3\lg\lg m \cdot 12\frac{m}{\lg m}.
\end{aligned}$$

La somme des termes $G^t_{i,j}$ est le nombre total de triangles de $\mathcal{T}$, *i.e.* m, puisque les micro triangulations ne partagent pas de faces : l'hypothèse 1 (additivité de la décomposition) est donc satisfaite (ici le terme d'erreur $O(\frac{m}{\lg m})$ vaut 0).

La somme des $G^s_{i,j}$ est la somme des degrés des nœuds de la carte G ou, de manière équivalente, deux fois le nombre des arcs : les arguments qui suivent confirment que cette somme fournit un terme $O(\frac{m}{\lg m})$, validant ainsi notre Hypothèse 2 (le nombre de côtés des micro triangulations est sous-linéaire).

Puisque la carte G a des faces de degré ≥ 3, le nombre a de ses arcs est borné linéairement par le nombre f de ses faces : $2a = \sum_f d(f) \geq 3f$.

La formule d'Euler peut donc s'écrire de la manière suivante (où n est le nombre de nœuds et g est le genre de G, qui correspond aussi au genre de la triangulation initiale $\mathcal{T}$) :

$$3(a+2) = 3n + 3f + 6g \qquad \Leftrightarrow \qquad a \leq 3n + 6(g-1).$$

Finalement, le nombre n de nœuds de G est borné par $12m/\lg m$, donc le coût de la représentation de G est

$$C = 2.175m + 9\lg\lg m \cdot 2\left(3 \cdot 12\frac{m}{\lg m} + 6(g-1)\right) + 3\lg\lg m \cdot 12\frac{m}{\lg m},$$

ce qui termine la preuve. Observons aussi que la borne $g \leq \frac{1}{2}m+1$ comporte $\lg C \leq 2\lg m+8$ pour tout m, ce qui sera utilisé dans la section suivante. □

5.3.4 Carte des mini triangulations

La dernière structure de données est utilisée pour représenter la carte F qui décrit les adjacences entre les mini morceaux. F est une carte ayant genre plus petit ou égale au genre de G, et pouvant contenir des faces de degré au moins 3. Nous adoptons ici une représentation explicite utilisant des pointeurs sur $\lg m$ bits. Sa construction nécessite un temps linéaire et peut se trouver à la section 5.4.3.

Description de la représentation

Pour chaque nœud de la carte F nous stockons son degré, une référence vers la partie correspondante de G et la liste de ses voisins. Plus précisément, pour un nœud associé F_i à la mini triangulation $\mathcal{ST}_i$:
- F_i^s est le degré du nœud F_i dans la carte F (correspondant au nombre de mini triangulations adjacentes à $\mathcal{ST}_i$) ;
- F_i^G est un pointeur vers la sous-carte G_i de G, associée à la mini triangulation $\mathcal{ST}_i$.
- un tableau de pointeurs vers les voisins : $F_{i,k}^{address}$ est l'adresse du k-ème voisin de F_i dans F.

Analyse du stockage

Lemme 43. *Le stockage de la carte F nécessite asymptotiquement de $36(g-1)\lg m + O\left(\frac{m}{\lg m}\right)$ bits.*

Démonstration. Observons qu'une mini triangulation contient entre $\frac{1}{3}\lg^2 m$ et $\lg^2 m$ triangles, donc la carte F possède au plus $3m/\lg^2 m$ nœuds.
• F_i^s est au plus $\lg^2 m$ et ainsi représentable sur $2\lg\lg m$ bits.
• l'adresse de G_i est un pointeur dont la taille est bornée par $2\lg m + 8$.
• les pointeurs vers les voisins $F_{i,k}^{address}$ peuvent se stocker chacun sur $K'\lg m$ bits (où K' est une constante pouvant se calculer comme montré ci-dessous).

En prenant la somme sur toutes les mini triangulations, nous avons que la taille mémoire de F est $(2\lg m+8)\cdot 3m/\lg^2 m + K'\lg m \sum_i F_i^s$. La somme des termes F_i^s n'est que la somme des degrés des nœuds de F, qui est aussi deux fois le nombre de ses arcs.

De manière similaire aux arguments utilisés pour la carte G, le nombre d'arcs de F peut être borné exactement par trois fois le nombre de ses nœuds (qui est au plus $3m/\lg^2 m$), plus six fois $g'-1$, g' étant le genre de F qui est borné par le genre g de la triangulation initiale $\mathcal{T}$. Si nous considérons la borne $g < \frac{1}{2}m + 1$ concernant le genre, alors la valeur $K' = 3$ vérifie les contraintes pour $m \geq 5$. Finalement le coût mémoire total de F, en terme de bits, est : $36(g-1)\lg m + O(m/\lg m)$. □

5.4 Construction de la structure de données

5.4.1 Catalogue exhaustif des micro triangulations

Lemme 44. *La construction et stockage du tableau A, et de toute l'information associée aux tableaux $A_{i,k}$ peuvent se faire en temps $o(m)$.*

Démonstration. Ici nous allons utiliser un résultat, déjà introduit à la section 3.3, concernant le codage optimal des triangulations planaires [99] : une triangulation à n sommets peut se coder par un mot binaire de longueur $4n-2$ et poids $n-1$. La collection des tableaux A_i peut être construite à l'aide de l'énumération de tous les mots binaires de taille $4i-2$ et poids $i-1$ selon l'ordre lexicographique : cette phase nécessite un temps $O(\binom{4i}{3i}i)$, si l'on adopte l'algorithme de génération de combinaisons décrit par Knuth [81][4].

Pour chacun de ces mots, nous essayons d'abord de voir si l'algorithme de décodage aboutit en fournissant un résultat correct (c'est le cas, si le mot était effectivement le code pour une certaine triangulation à i sommets). Si l'algorithme échoue le mot est oublié et nous considérons le mot suivant, dans l'ordre lexicographique, de même taille et poids. Autrement, l'algorithme retourne une triangulation à $2i$ triangles : il est maintenant possible d'obtenir une triangulation $A_{k,j}$ à bord, juste en supprimant le sommet racine et ses arêtes incidentes. Si sa taille k est au plus $\frac{1}{4}\lg m$ alors $A_{k,j}$ est une micro-triangulation valide. Dans le cas positif nous construisons la représentation

[4]Nous allons considérer ici la phase de construction de l'information combinatoire des micro triangulations contenues dans le tableau A, sans nous préoccuper du coloriage des arêtes du bord : le calcul des micro triangulations coloriées se fait de manière similaire.

explicite $A_{k,j}^{explicit}$ (suivant la description fournie dans la section 5.3.1), et nous stockons dans A_i un pointeur vers cette représentation. Autrement, $\frac{1}{4}\lg m < k$ et nous ne considérons pas ce mot binaire, car la triangulation correspondante a trop de faces.

Les micro triangulations valides ainsi obtenues sont triées par ordre de taille croissante et stockées dans les tableaux A_i, chacun contenant toutes les triangulations de taille i. Les éléments de ces tableaux sont à leur tour triés selon l'ordre lexicographique du mot binaire correspondant.

Les mots binaires de départ sont ensuite stockés dans un tableau auxiliaire A_i^{code}.[5] Le décodage d'un mot et la construction de la triangulation correspondante nécessite un temps $O(i) \leq O(\lg m)$, ce qui nous dit que le coût du à la construction d'un Tableau A_i est $O(\binom{4i-2}{i-1}\lg m) = O(2^{3.24i}\lg m)$: en sommant sur i, la construction a un coût total de $O(m^{3.24\frac{1}{4}}) = O(m^{0.81})$. Pour conclure, trier les micro-triangulations valides selon leur taille croissante nécessite globalement $O(\|A\|\lg\|A\|) = O(m^{\frac{1}{4}2.175}\lg m)$ opérations élémentaires. □

5.4.2 Structure de Rank/Select

Lemme 45. *La construction du tableau B, ainsi que de toute l'information associée aux tableaux B_{pq} peut se faire en temps $O(m^{\frac{1}{4}}\lg m)$.*

Démonstration. La stratégie adoptée ici est basée sur l'énumération exhaustive de tous les vecteurs de bits de taille et poids donnés. Une fois fixés les paramètres de taille et poids $p, q \leq \frac{1}{4}\lg m$ il nous reste à appliquer à nouveaux l'algorithme précédent pour la génération, selon l'ordre lexicographique, de tous les mots binaires de taille p et poids q. La construction d'une représentation de la structure de *Rank/Select*, telle que celle donnée à la section 6.3, nécessite un temps $O(\lg m)$: puisque il existe au plus $2^{\frac{1}{4}\lg m}$ bit-vecteurs, il nous faut un temps $O(m^{\frac{1}{4}}\lg m)$ pour construire toutes ces représentations, ainsi que pour stocker tous les pointeurs correspondants dans un tableau B_{pq} (encore une fois, les mots binaires originaires sont stockés dans un tableau auxiliaire B_{pq}^{code}[6]). □

5.4.3 Carte des mini et micro triangulations

Lemme 46. *La construction des cartes G et F peut se faire en temps $O(m)$.*

[5] La taille de ce tableau correspond à celle de A_i, et ses éléments est toujours de $O(\lg m)$ bits, ainsi l'analyse faites au cours du lemme 49 s'applique aussi au cas de l'espace mémoire demandé par le tableau A_i^{code}.

[6] Ce tableau a pour éléments des mots de taille $O(\lg n)$ et sa taille est exactement celle de B_{pq}, ce qui permet d'appliquer à nouveau l'argument utilisé dans le lemme 41, pour l'évaluation de sa taille en mémoire.

Démonstration. La triangulation $\mathcal{T}$ est décomposée en mini et micro triangulations en temps $O(m)$ (voir Section 5.2.3), et ensuite les représentations explicites de G et F sont construites (Section 5.2.3). Pour chaque nœud G_{ij}, la référence vers la représentation explicite de la micro triangulation contenue dans le tableau A est calculée et stockée (voir section 5.4.3). Dans un premier temps, un parcours d'une représentation explicite du graphe G (une telle représentation n'est nécessaire que pendant cette phase de construction) permet d'évaluer et allouer la quantité exacte de mémoire nécessaire pour le stockage des nœud de G. Ensuite, avec un deuxième parcours de G il est finalement possible de calculer et stocker effectivement en mémoire l'information concernant ses nœuds (voir section 5.4.3). □

Calcul des références vers A

Un aspect crucial de cette phase concerne la représentation de l'information de connectivité d'une micro triangulation (associée et stockée dans un nœud $G_{i,j}$), qui est codée de manière implicite par le biais d'une référence vers un élément contenu dans le catalogue exhaustif contenant toutes les micro triangulations valides différentes. Considérons une telle micro triangulation t obtenue par l'application de notre stratégie de décomposition : sa taille r, la taille de son bord p et le nombre de ses côtés q sont connues (rappelons que au cours de la phase de décomposition, nous avons accès à une représentation explicite de la triangulation initiale). Pour calculer la représentation implicite d'une micro triangulation associée à un nœud $G_{i,j}$ nous procédons de la façon suivante[7].

Appliquer à t l'algorithme de codage [99] déjà utilisé à la section 5.4.1 : le résultat est un mot binaire de longueur $4r - 2$.

Ce code, de taille $O(\lg m)$, est utilisé pour faire une recherche binaire dans le tableau A_r^{code}, ce qui permet de déduire l'index de la triangulation t dans le tableau A_r (rappelons que A_r et A_r^{code} ont la même taille et leurs éléments sont en correspondance).

Ces tableaux contiennent au plus $2^{\frac{1}{4}2.175r}$ éléments et une recherche binaire nécessite donc $\lg \|A_r^{code}\| = O(r)$ opérations élémentaires : chacune de ces opérations ne prend que temps $O(1)$, puisque il s'agit de comparer deux mots binaires de taille $O(\lg m)$ (ces mots sont les codes de la triangulation t, obtenues à l'aide du codeur [99]). Puisque dans notre cas $r \leq \frac{1}{4} \lg m$ et il existe $\Theta(\frac{m}{\lg m})$ micro triangulations, cette phase a complexité $O(m)$.

Une stratégie similaire peut s'appliquer pour déterminer la partie de la représentation implicite concernant le coloriage (marquage des sommets) du bord d'une micro triangulation : ce codage est réalisé par une référence vers un bit-vecteur stocké dans le tableau B. Étant connus la taille p et le

[7]Rappelons que la représentation implicite (un pointeur vers A) est constituée de deux parties : l'information combinatoire, et l'information relative au coloriage (ces deux informations pouvant se séparer). Ici nous considérons le calcul de la partie combinatoire.

nombre de côtés q du bord, une recherche binaires dans le tableau B_{pq}^{code} permet de retrouver l'index du bit-vecteur correspondant, présent dans le tableau B_{pq}. Compte tenu de la taille des tableaux B_{pq}^{code} et du fait que les bit-vecteurs concernés ont taille $O(\lg m)$, une recherche binaire nécessite de $O(\lg m)$ comparaisons (chacune pouvant se faire en temps $O(1)$, vu la longueur des mots) : encore une fois le coût global de la construction est $O(m)$.

Construction d'une représentation compacte pour G et F

La carte F est une carte classique représentée de manière explicite à l'aide de pointeurs et sa construction ne pose aucun problème particulier. La représentation de G est moins triviale : sa représentation, compacte, est calculée à chaque étape pour chacun des sous cartes, dont les représentations sont écrites en zones de mémoire consécutives. Pour chaque nœud G_{ij} de G_i correspondant à une micro triangulation $\mathcal{TT}_{ij}$, il faut d'abord évaluer (comme décrit à la section 6.4.2) et allouer le nombre exact de bits nécessaire pour coder chaque index et référence constituant l'information associée au nœud G_{ij} (nombre de triangle, taille du bord, nombre de côtés, références vers le tableaux A). Il ne reste alors qu'à allouer la mémoire nécessaire à stocker les références vers les voisins de G_{ij} dans G (l'information codant l'adjacence entre nœuds a taille fixe et bornée, ainsi que le nombre de voisin d'un nœud est connu en avance). Dans une deuxième étape la carte est traversée à nouveau, afin de compléter la représentation, avec l'écriture effective des références (pointeurs locaux) entre nœuds voisins dans G.

5.5 Navigation

Représentation des sommets et des triangles Dans notre structure un triangle t est représenté par un triplet (F_i, a, w), où F_i est un nœud du graphe F (tel que $t \in \mathcal{ST}_i$), a est l'adresse mémoire de G_{ij} (dans la zone de mémoire de G_i telle que $t \in \mathcal{TT}_{ij}$), et w est l'index du triangle correspondant à t dans la représentation explicite $A_{\kappa,\lambda}^{explicit}$ ($A_{\kappa,\lambda}$ indiquant la micro triangulation associé au nœud G_{ij}).

De manière similaire un sommet est représenté par un triplet (F_i, a, v). Il est important de souligner qu'une telle représentation ne permet pas directement de représenter les sommets de façon unique : contrairement aux triangles, les sommets (du bord) peuvent être partagés par un nombre arbitrairement grand de micro triangulations.

Opérations de navigation dans la triangulation Étant donné un triangle (F_i, a, w) ou un sommet (F_i, a, v) les opérations $Triangle$, $Index$ et

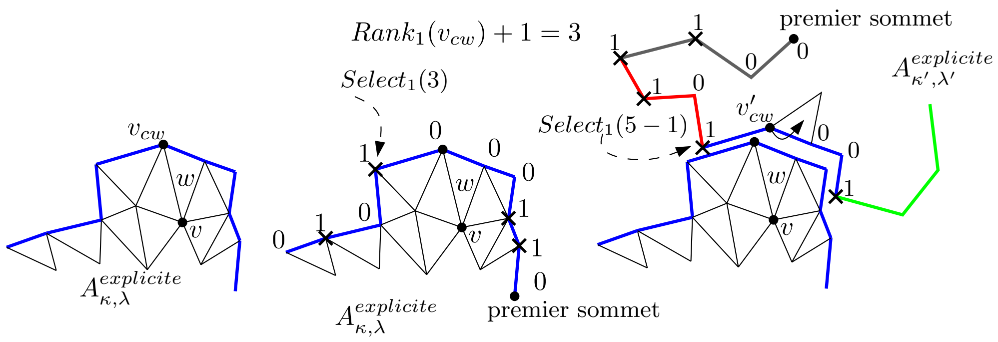


FIG. 5.6 – Navigation entre triangle adjacents.

$Vertex$ introduites dans la section 5.1.2 s'implantent facilement, à l'aide des informations stockées dans la représentation explicite $A^{explicit}_{\kappa,\lambda}$.

L'opération $Voisin((F_i,a,w),(F_i,a,v))$, qui renvoie le voisin d'un triangle opposé au sommet v, demande plus d'attention.

Si le triangle w ne se trouve pas sur le bord de la micro triangulation $A^{explicit}_{\kappa,\lambda}$, il suffit de retrouver son voisin w' à l'aide de l'information associée à $A^{explicit}_{\kappa,\lambda}.triangles$ et $A^{explicit}_{\kappa,\lambda}.sommets$, et de retourner le triplet (F_i,a,w').

Autrement nous procédons de la manière suivante :

— étant donné le triangle w de $A^{explicit}_{\kappa,\lambda}$ et un sommet incident v, trouver le sommet v_{cw} incident à w qui suit v dans le sens anti-trigonométrique ("clockwise").

— calculer $l = Rank_1(v_{cw})+1$ dans le vecteur de bits associé à G_{ij} : cela nous fournit le l-ème côté de la micro triangulation $\mathcal{TT}_{ij}$ ($l=3$, pour l'exemple de la figure 5.6) ;

— calculer $l' = Select_1(l) - v_{cw}$: ce qui nous dit que le sommet v_{cw} est le l'-ème avant la fin du côté correspondant ($l'=1$ dans la figure 5.6) ;

— soit $x = G^{address}_{i,j,l}$, $y = G^{back}_{i,j,l}$ and $z = G^{small}_{i,j,l}$;

— si $z>0$, soit $G_{i'}$ la sous-carte de G pointée par le z-ème voisin $F_{i'}$ de F_i dans le graphe F ; autrement soit $G_{i'}$ égal à G_i ;

— soit $G_{i',j'}$ le nœud de G correspondant à l'adresse x dans la zone de mémoire relative à $G_{i'}$ et $A^{explicit}_{\kappa',\lambda'}$ la micro-triangulation associée ($y=5$ dans la Figure 5.6, et soit y le numéro du côté de $G_{i',j'}$ correspondant au l-ème côté de $G_{i,j}$) ;

— soit $v'_{cw} = Select_1(y-1)+l'$ dans le bit-vecteur associé à $G_{i',j'}$: alors v'_{cw} dans $A^{explicit}_{\kappa',\lambda'}$ correspond à v_{cw} dans $A^{explicit}_{\kappa,\lambda}$;

— soit w' be le triangle pointé par v'_{cw} in $A^{explicit}_{\kappa',\lambda'}$;

• retourner le triangle $(F_{i'},x,w')$.

5.6 Entropie des triangulations planaires avec bord

La représentation décrite dans ce chapitre est basée sur la construction d'un catalogue contenant toutes les triangulations planaires, avec un bord de taille arbitraire, ayant au plus $\frac{1}{4}\lg m$ faces (micro triangulations valides). Un aspect crucial concerne la taille de ce catalogue, liée au nombre de telles triangulations, dont dépendent la longueur des pointeurs vers le tableau A et donc la taille mémoire globale de notre structure. Il est donc naturel de s'intéresser à l'énumération des triangulations d'un polygone avec sommets internes et sans arêtes multiples, pouvant contenir des cordes (arêtes entre sommets du bord).

Soit T^b_p l'ensemble de ces triangulations qui contiennent p triangles et $T^b_{k,p}$ soit le sous-ensemble des triangulations dont le bord est un polygone de taille k. Observons d'abord que $p = 2n + k$, où $n + 1$ est le nombre de sommets internes. En principe il est possible d'établir une borne pour $|T^b_p|$ de manière élémentaire en partant de la formule 2.2 (écrite en terme du nombre p de faces) et en évaluant la quantité :

$$\lg(\sum_{k\geq 3}^{p} |T^b_{k,p}|) = \lg(\sum_{k\geq 3}^{p} \frac{2\cdot(2k-3)!\,(2p-1)!}{(k-1)!\,(k-3)!\,(\frac{p-k}{2}+1)!})$$

Il est néanmoins plus simple de déduire ce résultat en terme de fonctions génératrices. Considérons les séries formelles de puissances $F(u,v)$ et $f_k(v)$ définies par :

$$F(u,v) = \sum_{p,k} |T^b_{k,p}| u^k v^p \ = \ \sum_{k\geq 3} f_k(v)\, u^k.$$

Observons que le nombre $|T^b_p|$ correspond aux coefficients dans le développement de Taylor de $F(1,v)$ pour la variable v.

Une décomposition des triangulations par suppression de l'arête racine ([50, Sec 2.9.5]) conduit à l'équation suivante

$$F(u,v) = \frac{v}{u}(u^2 + F(u,v))^2 + \frac{v}{u}(F(u,v) - u^3 f_3(v)) - vF(u,v)f_3(v). \quad (5.1)$$

L'équation 5.1 peut se résoudre à l'aide de la *méthode quadratique* ([50, Sec 2.9.1]), qui permet d'obtenir les relations suivantes, équivalentes à la formule de Mullin : soit $t \equiv t(v)$ l'unique solution analytique en zéro de l'équation

$$t(v) = 1 + v^2 t(v)^4.$$

Alors $f_3(v) = v\,(2t^2 - t^3)$ et

$$F(1,v) = \frac{1}{2vt^2}(2 - 3t + 3vt^2 + \sqrt{1 + 7t - 9t^2 - 4vt^2 + 6vt^3 + 4vt^4}).$$

Puisque $F(1, v)$ est analytique à l'origine et satisfait une équation algébrique, une méthode standard pour l'étude de ses coefficients (correspondant au nombre de triangulations) est basée sur l'analyse de sa singularité dominante.

D'une part la fonction $t(v)$ possède une singularité en $v = \rho = 3^{3/2}/2^4$, où $t(\rho) = 4/3$. D'autre part, la racine carrée dans l'expression de $F(1, v)$ comporte une autre singularité pour $v = \sigma = 2^6/17^2$, avec $t(\sigma) = 17/16$.

Puisque $\sigma < \rho$, σ est la singularité dominante. Dans le voisinage de σ, $F(1, v)$ a le développement singulier

$$F(1, v) = \frac{7}{8} - c\sqrt{1 - v/\sigma} + O((1 - v/\sigma)^{3/2}).$$

A l'aide de ce développement et du théorème [37, Thm VII.6] on obtient

$$|T_p^b| \;=\; c'\sigma^{-p}p^{-3/2}(1 + O(1/p)),$$

expression dont le logarithme est donné par

$$\lg |T_p^b| \;=\; p\lg(1/\sigma) - \frac{3}{2}\lg p + \lg c' + O(1/p).$$

Pour conclure, il est possible d'établir qu'il existe une constante c'' telle que pour tout $p \geq 1$,

$$\lg |T_p^b| \;\leq\; p\ \lg(1/\sigma) + c'' \;\leq\; 2.175\,p + c''.$$

En utilisant des vérifications numériques pour les premiers termes (voir tableau 5.1) et des arguments de sous-additivité, il est possible de vérifier que la relation précédente est encore vraie lorsque la constante c'' est choisie égale à 0, ce qui permet d'établir le lemme suivant :

Lemme 47. *Le nombre de triangulations ayant p faces est borné par* $2^{2.175p}$.

5.7 D'autres requêtes et données géométriques

Les opérations que nous avons considérées à la section 5.5 ne concerne que la navigation entre triangles : en effet il existe des stratégies permettant d'implanter d'autres opérations locales sur la triangulation, concernant par exemple l'adjacence entre sommets .

Comme déjà mentionné à la section 5.1.2, il est possible d'implanter en temps $O(1)$ les requêtes suivantes :

- $Adjacent(u, v)$: teste si les sommets v et u sont adjacents ;
- $Degre(v)$: retourne le degré du sommet v ;
- $Voisin(v, e, i)$: étant donné un sommet v et une arête incidente e, retourne le sommet u adjacent à v incident à l' arête (incidente à v) que l'on trouve en tournant i fois autour de v en sens direct à partir de e ;

p	$\lvert T_p^b \rvert$	$\lg \lvert T_p^b \rvert$	$2.175p$	p	$\lvert T_p^b \rvert$	$\lg \lvert T_p^b \rvert$	$2.175p$
1	1	1	2.175	16	222991801	29	34.800
2	2	2	4.350	17	924533609	31	36.975
3	6	4	6.525	18	3850615992	33	39.150
4	19	6	8.700	19	16104053458	35	41.325
5	66	8	10.875	20	67600318577	37	43.500
6	236	9	13.050	21	284729605627	40	45.675
7	877	11	15.225	22	1202959079012	42	47.850
8	3321	13	17.400	23	5096769502458	44	50.025
9	12840	15	19.575	24	21650252797852	46	52.200
10	50302	17	21.750	25	92186861004044	48	54.375
11	199657	19	23.925	26	393399791627096	50	56.550
12	800152	21	26.100	27	1682246653890199	52	58.725
13	3235923	23	28.275	28	7207306601612326	54	60.900
14	13182456	25	30.450	29	30933464929153561	56	63.075
15	54063790	27	32.625	30	132985945811160992	58	65.250

Tab. 5.1 – Ce tableau montre le nombre de triangulations pour $p = 1..30$, ainsi que le lg de cette quantité comparé à la valeur $2.175p$.

Il est intéressant de souligner que ces opérations ne peuvent pas s'implanter en temps $O(1)$ d'habitude, même à l'aide de structures de données explicites (le fait de visiter les arêtes autour d'un sommet, nécessite en général un temps qui est proportionnel au degré du sommet). Il existe dans la littérature un certain nombre de travaux, focalisés sur la représentation des adjacence entre sommets (plutôt que le voisinage entre faces), permettant de traiter en temps constant la première des opérations listées ci-dessus. Mais aucune stratégie efficace pour traiter les deux dernières requêtes n'a été proposée, à notre connaissance (de telles opérations ont été l'objet d'études seulement pour de structures de données plus simples, telles que les arbres).

La solution est basée, comme pour le cas des données géométriques associées aux sommets (chapitre 4), sur le précalcul des valeurs retournées par une certaine fonction (par exemple celle qui retourne le degré d'un sommet, ou celle qui donne ses coordonnées) et l'enrichissement de l'information concernant les nœuds du graphe G (et éventuellement de F) : le "petit nombre" de sommets multiples garantit que l'information auxiliaire ne donne qu'une contribution négligeable (cette stratégie a été esquissée à la fin du chapitre 4 : voir aussi la figure 5.7). Plus précisément, dans le cas de l'opération $Degre(v)$, on peut procéder de la manière suivante. Pour les sommets internes à une micro triangulation, ou partagés au plus par deux micro triangulations adjacentes, il suffit de stocker le degré d'un sommet v directement dans la représentation $A_{ij}^{explicite}$. En ce qui concerne les som-

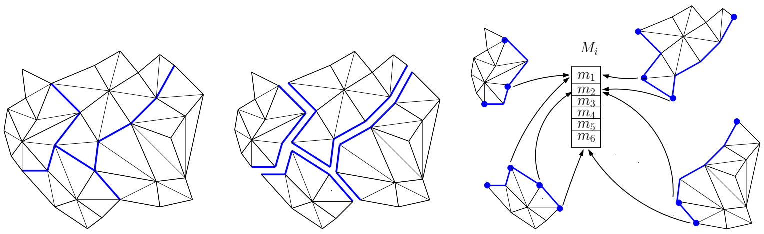


FIG. 5.7 – Cette image illustre le cas de sommets multiples partagés par les micro triangulations appartenant à une mini triangulation. Leur nombre est limité, environ $O(\frac{m}{\lg m})$ globalement, et au plus $O(\lg^2 m)$ dans la mini triangulation. Ainsi il est possible d'ajouter de l'information supplémentaire (à l'aide d'une liste M_i) pour traiter des données géométriques (par exemple les coordonnées des sommets) ou gérer d'autres requêtes locales (par exemple retrouver le degré des sommets).

mets multiples qui n'appartiennent qu'à une seule mini triangulation $\mathcal{ST}_i$, il suffit de stocker un tableau M_i, ayant une case pour chacun de ces sommets, et contenant les résultats de l'opération $Degre(v)$. Dans ce cas leur degré est au plus $O(\lg^2 m)$, donc $O(\lg \lg m)$ bits suffisent : globalement il nous faut une quantité négligeable de mémoire (il existe $O(\frac{m}{\lg m})$ sommets multiples). Finalement, il ne reste qu'à considérer les sommets partagés par plusieurs mini triangulations (au moins 3) : leur nombre est globalement de $O(\frac{m}{\lg^2 m})$, ce qui permet de stocker (dans un tableau M) explicitement leur degré (sur $O(\lg m)$ bits). Des stratégies similaires permettent d'implanter les opérations $Adjacent(u, v)$ et $Voisin(v, e, i)$, ainsi que d'avoir l'accès à des données externes associées aux sommets.

Chapitre 6

Une structure dynamique

6.1 Introduction

S'il existe une vaste littérature concernant la conception de structures de données compactes, ce n'est que récemment que les algorithmiciens se sont intéressés au cas de structures de données succincte dynamiques. La plupart des travaux existants traitent le cas de structures telles que les dictionnaires [106], les bit-vecteurs [105] et les tableaux dynamiques [22, 11]. Certaines de ces résultats se sont avérés utiles pour la conception de représentations succinctes dynamiques de structures plus complexes, pour la majorité des arbres binaires [106, 94], tandis que les graphes n'ont fait l'objet que de très peu de recherches [11]. Dans notre travail nous nous sommes intéressés au cas de structures de données géométriques, et aux maillages triangulaires en particulier : encore une fois, l'information de connectivité d'un maillage est au centre de notre attention. Le résultat principal présenté dans ce chapitre est exprimé par le théorème suivant [1] :

Théorème 48. *Il existe une représentation dynamique d'une triangulation d'une surface de genre g ($g = o(\frac{m}{\lg m})$), ayant un bord simple, qui nécessite asymptotiquement* 2.175 *bits par face, atteignant donc l'entropie de cette classe de triangulations. Plus précisément, étant donnée une triangulation à m triangles, l'espace utilisé est donnée par :*

$$2.175m + O(g \lg m) + o(m) \text{ bits},$$

La représentation admet des opérations standard de navigation nécessitant un temps $O(1)$ (pire cas) et des mises à jour locales de la triangulations, nécessitant

- *un temps $O(1)$ amorti pour l'insertion d'un sommet de degré 3 (s'il n'y a pas d'accès à des données externes[2]),*

[1] Ce résultat a été publié à CCCG 2005 [26]

[2] Cette représentation dynamique permet d'associer des données externes aux sommets

• *un temps* $O(\lg m)$ *amorti pour l'insertion d'un sommet de degré* 3 *(si l'accès aux données associées aux sommets est permis),*

• *un temps* $O(\lg^2 m)$ *amorti pour la suppression d'un sommet de degré* 3 *et la bascule d'arête (*flip*).*

Ce résultat *dynamique* est une extension de la structure *statique* décrite dans le chapitre 5, qui ne permettait pas une mise à jour de la triangulation.

A notre connaissance, ce travail est le premier à présenter une version dynamique de structures de données succinctes pour des triangulations ou graphes : les résultats précédents ne sont que statiques ou compacts (au sens du mot spécifié à la section 4.2), et pour la plupart traitent des structures plus simples telles que les arbres binaires.

6.1.1 Travaux existants

Arbres et graphes. Les seules représentations dynamiques pouvant se dire succinctes (au sens de l'optimalité en terme de ressources mémoire) ont été proposés pour le cas des arbres binaires à n nœuds [106, 94]. L'espace utilisé est encore asymptotiquement optimal (2 bits par nœud) et les opérations de navigation usuelles sont supportées en temps $O(1)$ (cas le pire). La nouveauté concerne la modification de l'arbre, qui peut se réaliser à l'aide de certaines opérations de mise à jour locale agissants sur les nœuds.

En particulier, le premier travail [96] mettait à disposition l'ajout et la suppression de feuilles et l'insertion d'un nœud interne au milieu d'une arête. Ces deux opérations peuvent s'effectuer en temps $O(\lg^2 n)$ amorti, lorsque des données externes de taille $O(\lg n)$ (bits) sont associées aux nœuds de l'arbre (la complexité peut être légèrement inférieure, selon la présence et la taille de données externes éventuelles). Le coût d'une mise à jour a été amélioré dans [106], où les modifications se font en temps $O((\lg\lg n)^{1+\varepsilon})$ amorti (toujours si des données de taille $O(\lg n)$ sont associées aux nœuds).

Comme souvent mentionné dans cette thèse, les représentations (statiques) conçues pour les graphes [74, 96, 31] étaient basées sur des représentations succinctes pour les arbres (et donc des mots de parenthèses) et tiraient profit de certaines propriétés combinatoires caractérisant la planarité des graphes, notamment les ordres canoniques, les réaliseurs et les plongements à 4 pages.

S'il est vrai que de telles structures fournissent un codage compact d'un graphe, tout en permettant une navigation locale en temps $O(1)$, il paraît difficile, voire impossible, de traiter des mises à jour locales dans le graphe : en effet, l'utilisation des propriétés citées ci-dessus impose des fortes contraintes combinatoires, qui se "manifestent" globalement dans le graphe, et qui sont difficiles à décrire lors d'une modification locale.

de la triangulation, telles que leurs coordonnées : dans ce cas l'insertion de sommets ne peut pas s'effectuer en temps constant.

Graphes et petits séparateurs. Une approche tout à fait différente, basée sur les propriétés des graphes ayant des petits séparateurs, avait permis à Blandford, Blelloch et d'autres [14, 11] de concevoir une structure de données compacte, pour laquelle des mises à jour apparaissaient implantables efficacement en pratique. Un résultat récent concernant les dictionnaires dynamiques [11] permet de compléter et rendre un peu plus précise d'un point de vue théorique l'analyse de leur structure de donnée.

Il faut néanmoins souligner la difficulté à fournir des bornes intéressantes explicites concernant le coût d'une mise à jour avec cette approche, encore une fois due au fait que les petits séparateurs tirent profit de certaines propriétés combinatoires difficiles à maintenir après modification locale du graphe.

6.1.2 Notre contribution

La contribution principale, exprimée par le théorème 48, consiste à montrer qu'il est théoriquement possible de mettre à jour, avec une bonne complexité, une représentation succincte d'une triangulation, tout en conservant une navigation efficace et en nécessitant un espace mémoire asymptotiquement optimal. Nous allons aussi esquisser une manière de traiter l'information auxiliaire associée aux cellules élémentaires du graphe (comme les coordonnées des sommets). Bien que d'intérêt plutôt théorique, ce chapitre met en évidence des idées dont nous nous sommes inspiré pour une réalisation pratique de nos structures de données : nous renvoyons pour cela au chapitre 8, qui est un premier pas vers l'étude du compromis entre taille des ressource mémoire et temps d'accès.

L'approche suivie ici, reprend la structure hiérarchique à 3 niveaux introduite dans [94, 105] pour le cas des arbres binaires dynamiques : nous décomposons la triangulation en micro triangulations, qui sont regroupées pour former des mini triangulations. Il existe toutefois plusieurs aspects faisant intervenir des différences entre notre résultat et les précédents.

D'une part, pour le cas des arbres binaires ils existent des ordres totaux "canoniques" (tels que par exemple celui obtenu par un parcours préfixe (ou postfixe) des sommets de l'arbre) qui sont relativement stables par insertion/suppression de feuilles. Cette situation ne se retrouve pas dans le cas des graphes, où des ordres canoniques sur les sommets sont perturbés à une distance arbitraire, lors d'une modification locale du graphe.

D'autre part, certaines opérations de mise à jour d'une triangulation (notamment le flip d'arête et la suppression de sommet), sont susceptibles de provoquer des modifications profondes dans la topologie des micro/mini régions, ce qui ne pouvait pas se produire dans le cas des arbres binaires, par rapport aux opérations de mise à jour proposées. L'implantation de ces opérations, comme illustré par la figure 6.7 (page 122), constitue l'une des difficultés principales que nous aurons à traiter.

6.2 Préliminaires

6.2.1 Vue d'ensemble de la solution

La stratégie adoptée dans ce chapitre est basée encore une fois sur un schéma de décomposition à 3 niveaux, proche de celui adopté dans le chapitre précédent : en particulier, ici nous tirons profit de la structure proposée pour les arbres dynamiques décrite dans [106]. Plus précisément, le niveau le plus bas consiste en une collection de micro-triangulations de taille $\Theta(\lg m)$ formant une partition de l'ensemble des m triangles de la triangulation initiale $\mathcal{T}$: les *micro triangulations* ont taille entre $\frac{1}{12} \lg m$ et $\frac{1}{4} \lg m$, et leurs représentations explicites sont stockées dans une structure appelée[3] A.

L'une des différences avec la représentation statique du chapitre précédent concerne la formation des *mini triangulations*. Les micro triangulations sont regroupées afin de former de plus grosses triangulations (connexes par faces) contenant environ $\Theta(\lg^2 m)$ triangles : ainsi une mini triangulation $\mathcal{ST}_i$, contient entre $\frac{1}{3} \lg m$ et $\lg m$ micro triangulations. Cette construction ne se réalise pas pendant une première phase de précalcul (comme pour le cas statique), mais plutôt de manière incrémentale à l'aide des opérations dynamiques (insertions/suppressions de sommets, flip d'arêtes) à disposition : cela sera rendu possible grâce à des procédures spécialement conçues pour la décomposition/fusion "locale" des mini et micro triangulations (une telle décomposition/fusion locale ne faisant intervenir qu'un nombre limité de micro/mini triangulations).

Le deuxième niveau, décrivant les relations d'adjacence entre micro triangulations, est encore une fois représenté à l'aide des cartes G et F déjà introduites auparavant. Si F est représentée explicitement (avec des pointeurs de taille $\lg m$), G se représente à l'aide de pointeurs locaux sur $O(\lg \lg m)$ bits : cela peut se faire en considérant la sous-carte G_i correspondant aux $O(\lg m)$ micro triangulations contenues dans $\mathcal{ST}_i$.

Un point crucial concerne l'organisation de la mémoire relative aux cartes G et F : leur mise à jour dynamique est rendue possible par l'adoption des tableaux dynamiques décrits à la section 2.6.3 (leur stockage étant toujours asymptotiquement négligeable, en vertu aussi de la planarité locale). Finalement, le terme dominant caractérisant l'espace utilisé par notre structure est donné par la somme des représentations implicites des micro triangulations (références au catalogue A).

6.2.2 Modifications dynamiques de la triangulation

La structure de donnée présentée ici est une extension et amélioration de la représentation fournie au chapitre 5 : toutes les opérations de navigation

[3]Pour les notations et propriétés concernant les micro triangulations (relations d'adjacences, sommets multiples, partition des arêtes du bord, ...) nous renvoyons au chapitre 5.

et adjacence entre triangles sont encore disponibles et implantables en temps $O(1)$, dans le pire cas.

En plus de ces requêtes statiques, nous présentons ici une collection complète d'opérations de base qui permettent de modifier localement la triangulation :

• $Insert(\triangle)$: ajoute un sommet de degré 3 au triangle $\triangle$ (avec les trois faces et arêtes incidentes).
• $Delete(v)$: supprime un sommet de degré 3.
• $Flip(\triangle, v)$: flip l'arête de $\triangle$ opposée au sommet v.

Celles-ci sont des opérations standard pour la mise à jour d'une triangulation sans bord. L'insertion/suppression d'un sommet du bord peut être traitée d'une manière similaire, dont nous omettons les détails[4].

6.3 Catalogue des micro triangulations coloriées

Micro triangulations. Dans le chapitre précédent, nous avons construit et stocké un catalogue exhaustif $\mathcal{D}$ de toutes les micro triangulations coloriées : pour chaque élément de ce catalogue on a stocké (dans une structure A) sa représentation explicite.

Un certain nombre d'informations étaient stockées afin de faciliter la navigation et d'autres requêtes locales d'adjacence entre les cellules de la triangulation. Afin de pouvoir gérer des mises à jour locales de la triangulation nous allons enrichir la description précédente en obtenant une nouvelle représentation explicite qui supporte des modifications locales d'une micro triangulation. Plus précisément, l'implantation en temps $O(1)$ de l'insertion/suppression d'un sommet peut se faire à l'aide des champs auxiliaires suivants, pour une triangulation ayant i triangles (on peut oublier pour l'instant le partitionnement des arêtes du bord) :

— $A_{i,j}^{explicite}.add\_sommet$ est un tableau contenant toutes les triangulations possibles de taille $i+2$ qui peuvent s'obtenir par l'insertion d'un sommet de degré 3 dans la triangulation courante A_{ij}. Le k-ème champ de ce tableau (un pour chaque face de la triangulation) contient un index sur $\Theta(\lg m)$ bits, qui permet l'accès à la nouvelle triangulation in $A_{i+2,j'}$, obtenue par l'insertion d'un sommet dans le k-ème triangle.

— $A_{i,j}^{explicite}.del\_sommet$ est un tableau conçu de manière similaire, pouvant gérer toutes les suppressions possibles de sommets de degré 3 dans la micro triangulation.

[4] Ces opérations constituent des opérations standard pour la mise à jour des triangulations. Plus précisément, toute triangulation d'un ensemble de points peut se transformer dans une autre quelconque de la même taille par une séquence de flips. L'étude des propriétés des distances entre deux triangulations (en terme du nombre de flips) a fait l'objet de nombreuses publications (voir [36, 61] pour une présentation détaillée). Ainsi, les triangulations qu'on peut obtenir en appliquant les opérations mentionnées ci-dessus correspondent exactement aux triangulations à bord considérées au chapitre 5.

— $A^{explicite}_{i,j}.flip\_arete$ est un tableau contenant tous les résultats possibles des flips d'arêtes internes.

Lemme 49. *Le stockage du tableau A nécessite asymptotiquement $o(m)$ bits.*

Démonstration. Il suffit d'observer que tant que la quantité d'information associée à une micro triangulation de taille i reste polynômiale en i, le coût du stockage du catalogue exhaustif $\mathcal{D}$ défini au chapitre **??** est sous-linéaire puisqu'il ne contient que $O(2^{3.17\frac{1}{4}\lg m})$ éléments. □

Il est à souligner que le champ *add_vertex* permet d'implanter l'insertion locale d'un nouveau sommet en temps $O(1)$. Si une micro triangulation devient trop grande elle doit être décomposée en micro triangulation valides.

Si l'on suppose de mettre à jour la triangulation en n'utilisant que l'insertions de sommets, cette éventualité ne peut se présenter que rarement, ce qui permet d'avoir un coût $O(1)$ amorti pour la modification de la structure.

Les champs *del_sommet* and $flip\_arete$ permettent aussi d'implanter en temps constant des modifications locales à l'intérieure d'une micro triangulation, ce qui n'apporte pas néanmoins de solution globale du point de vue théorique (si nous nous plaçons dans le cadre du pire cas possible). Comme il sera montré à la fin du chapitre, il pourrait être nécessaire d'effectuer des opérations de mise à jour non locales de notre structure avec une fréquence arbitrairement grande.

Vecteurs de bits et colorations du bord. Nous reprenons ici le catalogue de tous les bit-vecteurs introduit au chapitre 5 pour décrire la façon dont les arêtes du bord d'une micro triangulation se partagent en côtés. Rappelons que ce catalogue est constitué des tableaux B_{pq}, chacun contenant tous les bit-vecteurs de taille p et poids q, pour $p < \frac{1}{4}\lg m$, ainsi que de l'information nécessaire à répondre à des requêtes statiques de *Rank*/*Select* en temps constant. En particulier il est possible d'enrichir ces tableaux afin d'implanter en temps constant l'insertion/suppression/changement d'un bit à une position donnée, à l'aide de champs auxiliaires comme déjà remarqué pour le tableau A (ces opérations sont utilisées pour la mise à jour efficace de la triangulation après insertion d'un sommet sur le bord). La taille mémoire de ces tableaux reste asymptotiquement sous-linéaire à cause du nombre d'entrées qui est $O(m^{\frac{1}{4}})$.

6.4 Carte des micro triangulations

Nous allons ici donner la description détaillée des structures de données utilisées pour représenter une sous-carte G_i correspondante aux nœuds de G qui appartiennent à une même mini triangulation $\mathcal{ST}_i$.

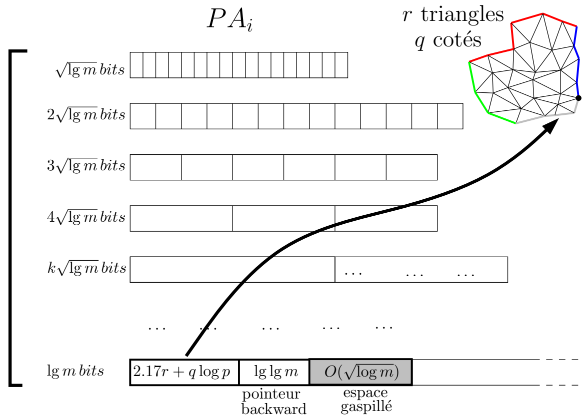


FIG. 6.1 – Ces images donnent un esquisse de l'organisation de la mémoire concernant les tableaux de la collection PA_i.

6.4.1 Description détaillée de l'organization mémoire

La description qui suit s'inspire de celle donnée au chapitre 5 : la différence principale concerne l'utilisation des tableaux dynamiques illustrés à la section 2.6.3, l'un des ingrédients qui nous permis ont de concevoir et analyser une structure compacte dynamique pour les triangulations.

Les 5 collections de tableaux extensibles, associés à G_i, permettent de stocker la connectivité et le coloriage du bord des micro triangulations, ainsi que leurs relations d'adjacences décrites par la carte G (les notations adoptées ici sont cohérentes et étendent celles du chapitre 5).

- Un tableau extensible S_i est utilisé pour stocker les relations d'adjacence entre mini triangulations (représentées par des arcs de la carte F) : ce tableau possède $O(\lg^2 m)$ éléments, chacun de taille $\Theta(\lg m)$, indiquant les voisins de la sous-carte G_i. Bien que la taille d'un seul S_i puisse être de $O(\lg^3 m)$ bits, globalement, sur l'ensemble de toutes les mini triangulations, la taille moyenne d'un tel tableau est bien inférieure (voir lemme 50 pour plus de détails).
- Un tableau extensible T_i est utilisé pour contenir toutes les informations relatives à chaque micro triangulation : l'élément $T_i[j]$, associé à la micro triangulation $\mathcal{TT}_{i,j}$ est constitué d'une suite de $O(\lg\lg m)$ bits, obtenue par la concaténation des champs suivants :
  - $G^t_{i,j}$ est le nombre de triangle de $\mathcal{TT}_{i,j}$.
  - $G^b_{i,j}$ est la taille du bord de $\mathcal{TT}_{i,j}$.
  - $G^s_{i,j}$ est le degré du nœud $G_{i,j}$ dans la carte G_i.
  - $G^{PA}_{i,j}$ est une référence vers un élément contenu dans l'un des tableaux de la collection PA_i (définie ci-dessous), contenant la représentation de $\mathcal{TT}_{ij}$, stockée de manière implicite sous forme d'une référence

vers le tableau A.
- $G^E_{i,j}$ est une référence, dans l'un des tableaux de la collection PE_i, vers la liste des nœuds adjacents à G_{ij}.

Rappelons que chacun de ces champs peut se stocker sur $O(\lg \lg m)$ bits, car la carte G_i a au plus $O(\lg m)$ nœuds et $O(\lg^2 m)$ arcs.
- Une collection de tableaux extensibles PA_i est utilisée pour stocker implicitement (sous forme de références vers le tableau A) la représentation des micro triangulations associées à G_i.
  La collection PA_i consiste de $O(\sqrt{\lg m})$ tableaux extensibles : les éléments du k-ème tableau de cette collection ont taille $k\sqrt{\lg m}$. Chaque élément doit contenir des données de la forme :
  - l'index d'une micro triangulation contenue dans le tableau A (cet index nécessitant de $2.17r + q \lg p$ bits pour une triangulation de taille r, ayant un bord de taille p, dont les arêtes sont partitionnées en q côtés).
  - un index sur $\lg \lg m$ bits utilisé comme pointeur "backward" vers le tableau T_i.

  Des couples de données pourront être ajoutés, supprimés et modifiés dans les tableaux de PA_i. La seule contrainte que nous imposons concerne la taille des cases de ces tableaux : chacun de ces couples sera affecté au tableau de PA_i dont les éléments ont taille $k\sqrt{\lg m}$, où k est le plus petit entier tel que le couple puisse se stocker sur $k\sqrt{\lg m}$.
- Une collection PE_i de tableaux extensibles est utilisée pour stocker la liste des demi-arcs incident aux nœuds $G_{i,j}$ (cette liste est triée et reflète l'ordre cyclique des demi-arcs autour des nœuds). Pour chaque demi-arc, nous stockons les informations suivantes, chacune nécessitant $O(\lg \lg m)$ bits :
  - $G^T_{ij}.source$ est une référence vers le tableau T_i à l'élément associé à G_{ij} (nœud source de l'arc) ;
  - $G^T_{ij}.target$ est une référence vers le tableau T_i à l'élément associé à $G_{i'j'}$, qui est pointé par le demi-arc ($G_{i'j'}$ est le nœud "target" du demi-arc) ;
  - $G_{ij}.back$ est l'index k' du côté de la micro triangulation correspondant au demi-arc courant, selon la numérotation des côtés relative au nœud opposé $G_{i'j'}$.
  - $G_{ij}.small$ est un pointeur vers la mini triangulation $G_{i'}$, dans la liste des voisins de G_i dans la carte F.

  Un nœud $G_{i,j}$ peut avoir degré entre 1 et $\lg m$, ainsi sa représentation nécessite entre $\lg \lg m$ et $\lg m \lg \lg m$ bits.
  La collection PE_i est constituée de $\ell = \sqrt{\lg m \lg \lg m}$ tableaux dont les champs ont taille $k\ell$, pour $k = 1, \ldots, \ell$.
  L'information associée à chaque nœud (suivant la description ci-dessus) est stockée dans une case de l'un des tableaux de PE_i : plus précisément, dans le tableau dont les cases ont taille $k\ell$, où k est le plus petit entier

tel que $k\ell$ bits suffisent pour contenir cette information.
Un tel choix garantit que la quantité de bits gaspillés est au plus $O(\ell)$, ce qui globalement n'a pas d'influence sur la taille de notre structure de données, car le nombre total des arcs de la carte G peut se borner par $O(m/\lg m)$.

6.4.2 Analyse du stockage

Lemme 50. *Le stockage des cartes G et F nécessite asymptotiquement $2.175m + o(m)$ bits.*

Démonstration. Comme dans le cas statique, la décomposition en mini et micro triangulation satisfait les hypothèses du chapitre 4. Ici il suffit de rappeler que pour les cartes G et F (cartes ayant le même genre g que la triangulation $\mathcal{T}$, et des faces de degré au moins 3) on peut établir les bornes suivantes sur le nombre d'arcs :

$$\sum_{ij} G^s_{ij} \;=\; O(\frac{m}{\lg m} + g) \text{ et } \sum_{i} F^s_i \;=\; O(\frac{m}{\lg^2 m} + g)$$

Pour chacune des collections de tableaux extensibles introduites précédemment nous allons calculer sa taille nominale ainsi que l'espace auxiliaire nécessaire, en accord avec le lemme 21.

• Puisque chaque tableau S_i a une taille nominale (et aussi totale) $s(S_i) = O(F^s_i \lg m)$, la quantité de mémoire nécessaire pour stocker les références entre mini triangulations voisines est $\sum_{i=1} S_i = O(\frac{m}{\lg^2 m} \lg m)$.

• Considérons le tableau extensible T_i : chacun de ses éléments (un pour chacune des $O(\lg m)$ micro triangulations contenues dans $\mathcal{ST}_i$) est constitué de $O(\lg lgm)$ bits, ce qui garantit que sa taille nominale (et totale) est $O(\lg m \lg\lg m)$. Globalement, le stockage des tableaux T_i nécessite donc $O(\frac{m}{\lg m} \lg\lg m)$ bits.

• Dans une première étape, calculons la taille nominale de la collection PA_i (ce qui fournit véritablement le terme dominant concernant le stockage de notre représentation). Chaque élément d'un des tableaux de cette collection contient une référence de taille variable vers le tableau A (il est donc utilisé comme représentation implicite d'une micro triangulation), et un pointeur backward au tableau T_i.

En accord avec les remarques concernant la taille du catalogue $\mathcal{D}$, le codage d'une micro triangulation à bord ayant G^t_{ij} triangles (avec un bord taille G^b_{ij} et G^s_{ij} côtés), nécessite $2.175G^t_{ij} + \lg \binom{G^b_{ij}}{G^s_{ij}}$ bits, ce qui implique que la taille nominale de la collection PA_i entière, pour une mini triangulation donnée, est

$$s(PA_i) \leq \sum_j (2.175G^t_{ij} + G^s_{ij} \lg G^b_{ij} + \lg\lg m + \sqrt{\lg m}) = \Theta(\lg^2 m)\,,$$

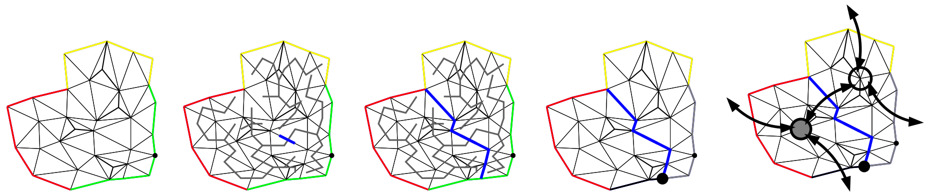


FIG. 6.2 – Décomposition d'une micro triangulation t_r, à l'aide du Lemme 51.

tandis que le stockage auxiliaire, suivant le lemme 21, est de

$$O(\sqrt{\lg m}\lg m + \sqrt{s(PA_i)\sqrt{\lg m}\lg m}) = O(\lg^{3/2} m + \sqrt{\lg^2 m \lg^{3/2} m}) \text{ bits}$$

Ainsi le stockage de toutes les collections PA_i (une pour chaque mini triangulation $\mathcal{ST}_i$), nécessite globalement

$$\sum_{ij}(2.175G^t_{ij} + G^s_{ij}\lg G^b_{ij} + \lg\lg m + \sqrt{\lg m}) + O(\frac{m}{\lg^2 m}\sqrt{\lg m}\lg m) +$$

$$+O(\frac{m}{\lg^2 m}\sqrt{\lg^{7/2} m}) = 2.175m + o(m) \text{ bits}$$

• De manière similaire il est possible d'évaluer la complexité en mémoire de la collection de tableaux extensibles PE_i.

□

6.5 Deux sous-algorithmes auxiliaires

Dans cette section nous allons décrire deux sous-routines importantes dans la conception d'une stratégie efficace de mise à jour de la triangulation. Dans le cas de l'insertion de sommets, elles ne seront appelées que lorsque la taille d'une micro triangulation donnée change de manière dramatique violant les bornes établies : leur coût peut donc être évalué de manière amortie, cet événement ne pouvant se présenter qu'après $\Theta(\lg m)$ insertions. Dans le cas de flip d'une arête ou suppression d'un sommet il n'existe pas la possibilité d'éviter l'utilisation de ces deux procédures, parfois assez coûteuses : cela peut arriver arbitrairement souvent, même si la taille des micro triangulations ne change pas de manière significative.

6.5.1 Décomposition d'une micro triangulation

La première sous-routine que nous considérons est conçue pour la décomposition et mise à jour d'une micro triangulation non valide, dont la taille excède la borne établie de $\frac{1}{4}\lg m$ triangles.

Lemme 51. *La décomposition d'une triangulation ayant entre $\frac{1}{4}\lg m$ et $\frac{3}{8}\lg m$ triangles en deux micro triangulations valides, ainsi que la mise à jour de notre structure, nécessite un temps $O(\lg m)$ amorti.*

Démonstration. Décomposons d'abord $\mathcal{TT}_{ij}$ en deux triangulations $\mathcal{TT}_{ij'}$ et $\mathcal{TT}_{ij''}$ dont les tailles seront bien valides. Il suffit de calculer un arbre couvrant du graphe dual de la triangulation : c'est un arbre binaire auquel peut s'appliquer facilement la procédure de décomposition du lemme 15.

L'ensemble des $\Theta(\lg m)$ triangles initiaux est maintenant partitionné en deux triangulations ayant au plus $\frac{1}{4}\lg m$ triangles chacune. Toutes les arêtes du bord appartiennent encore au bords de $\mathcal{TT}_{ij'}$ ou de $\mathcal{TT}_{ij''}$, tandis qu'une certaine quantité d'arêtes, précédemment internes, apparaissaient maintenant sur le bord qui sépare $\mathcal{TT}_{ij'}$ et $\mathcal{TT}_{ij''}$. De plus, ce procédé garantit qu'au plus deux nouveaux sommets multiples peuvent être créés.

Ainsi, puisqu'il n'y a qu'au plus 4 côtés qui se créent/modifient, le nombre de voisins du nœud associé à $\mathcal{TT}_{ij}$ qui changent leur degré dans la carte G reste constant. En ce qui concerne la mise à jour des informations stockées dans la structure de données, lorsqu'une nouvelle micro triangulation t' de taille r' est créée il faut :

- calculer un code sur $O(\lg m)$ bits de t', par exemple à l'aide du codeur optimal introduit dans [99] (voir section 6.3) ;

- effectuer une recherche binaire dans le tableau A : le résultat de cette procédure est une référence, qui sera utilisée pour représenter de manière implicite la structure combinatoire de t' ;

Il est facile de vérifier que toutes ces étapes nécessitent un temps $O(\lg m)$. Pour conclure, il ne reste qu'à mettre à jour G_i afin de tenir compte des nouvelles relations d'adjacence entre micro triangulations. Par rapport à la mise à jour des structures de données représentant G_i, il faut

- ajouter un élément à T_i, correspondant à la nouvelle micro triangulation créée $\mathcal{TT}_{ij'}$ (voir lemme 57).

- modifier dans la carte G_i la liste des arêtes incidentes à G_{ij} : la mise à jour des tableaux de la collection PE_i nécessite la modification d'au plus $O(\lg m)$ anciennes références (voir lemme 58).

- mettre à jour les étiquettes des arcs de G_i : lorsqu'un nouvel arc $e = (G_{i1}, G_{i2})$ est inséré dans G_i, la liste des étiquettes de tous les demi-arcs incidents aux nœuds G_{i1} et G_{i2} doit être modifiée : cette phase peut se réaliser en temps $O(\lg m)$, puisque le nombre de nœuds dont le degré change est constant.

Ces trois dernières étapes peuvent s'effectuer en temps $O(\lg m)$ amorti, en accord aux lemmes 57 et 58.

□

Si une micro triangulation non valide $\mathcal{TT}_{ij}$ est obtenue par la fusion de 2 ou plusieurs micro triangulations, il est toujours possible de décomposer

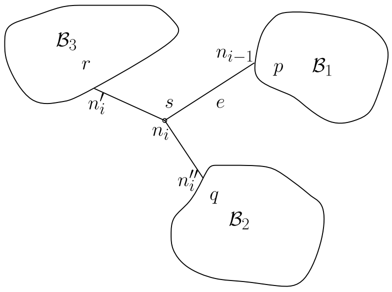


FIG. 6.3 – Décomposition d'un arbre binaire avec poids.

$\mathcal{TT}_{ij}$ en sous micro triangulations valides en temps $O(\lg m)$ amorti : il suffit d'appliquer un nombre constant de fois la procédure de décomposition décrite ci-dessus à $\mathcal{TT}_{ij}$.

6.5.2 Décomposer une mini triangulation

La procédure que nous allons décrire dans cette section est conçue pour la décomposition d'une mini triangulation dont la taille excède les bornes prescrites : rappelons que dans le chapitre présent ces bornes sont exprimées en terme de micro triangulations contenue dans une mini (en non plus en terme de triangles). Comme dans le cas des micro triangulations, notre intérêt est principalement de montrer que le nombre de relations d'adjacence entre mini régions qui changent est négligeable : il est donc important de concevoir une stratégie de décomposition et mise à jour qui soit bien meilleure que la simple reconstruction de la structure de données toute entière. Tout d'abord nous allons établir un résultat concernant les arbres binaires pondérés, qui est inspiré du lemme 15.

Lemme 52. *Considérons un arbre binaire $\mathcal{B}$ à n nœuds (chacun donc de degré au plus 3), auxquels sont associés des poids non négatifs w_i satisfaisant : $\sum_i^n w_i = K$ et $w_i \leq \frac{2}{3}K$ pour tout $1 \leq i \leq n$. Il est alors possible d'obtenir en temps $O(n)$ une décomposition de $\mathcal{B}$ en deux sous arbres $\mathcal{B}'$ et $\mathcal{B}''$ dont les poids satisfont :*

$$\sum_{i' \in I'} w_{i'} \leq \frac{2}{3}K + \max_i w_i \ , \ \sum_{i'' \in I''} w_{i''} \leq \frac{2}{3}K + \max_i w_i$$

où I' (resp. I'') désigne l'ensemble des indices des nœuds appartenant au sous-arbre $\mathcal{B}'$ (resp. $\mathcal{B}''$).

Démonstration. Préalablement, nous allons faire un parcours postfixe de $\mathcal{B}$ qui nous permet de calculer les poids des sous-arbres : ces poids seront utilisés pour étiqueter les demi-arêtes et diriger la stratégie de décomposition.

Le parcours de $\mathcal{B}$ part d'une de ses feuilles et procède par la conquête de ses nœuds en ordre post-fixe. Lorsque nous visitons un nœud n_i nous calculons le poids total du sous-arbre $\mathcal{B}_i$ ayant n_i comme racine : si on note K_i son poids, alors les nœuds restants dans $\mathcal{B}-\mathcal{B}_i$ ont poids total $K-K_i$. Lors du passage du nœud n_i à son ancêtre, nous associons à l'arête e traversée l'étiquette constituée du couple $(K_i, K-K_i)$, qui représente les poids totaux des deux sous-arbres séparés par e : le résultat de cette procédure est un arbre binaire dont les arêtes sont doublement étiquetées. Nous allons maintenant décrire un parcours glouton de $\mathcal{B}$ produisant la décomposition souhaitée. Le point de départ de ce parcours est une feuille ayant poids maximal : à la i-ème étape, après avoir conquis le nœud n_i, nous visitons le nœud adjacent, non traversé, ayant poids maximal parmi ses voisins (n_i a au plus deux voisins à visiter, étant de degré au plus 3) Cette procédure se termine lorsque la demi-arête traversée a pour étiquette $K_j \geq \frac{1}{3}K$.

Il est facile de voir qu'il existe une étape j où la demi-arête traversée a étiquette $\geq \frac{1}{3}K$: lors de notre parcours les valeurs des étiquettes ne peuvent pas diminuer puisqu'elles correspondent aux poids des sous-arbres visités (ces sous-arbres augmentant de taille et les poids des nœuds étant non négatifs). Considérons l'arbre $\mathcal{B}_1$ enraciné en n_{i-1}, visité à l'étape i lors du passage du nœud n_{i-1} au nœud n_i : soit p son poids et s le poids de n_i (voir figure 6.3). De n_i nous pouvons nous déplacer vers l'un de ses voisins n_i' ou n_i'' : désignons par $\mathcal{B}_2$ et $\mathcal{B}_3$ les deux arbres correspondants enracinés en n_i' et n_i'', dont les poids sont notés respectivement r et q(sans perte de généralité nous pouvons supposer $r \geq q$). Il est évident que les poids de ces sous-arbres satisfont $p+r+q+s=K$. De plus, puisque nous supposons que le parcours n'est pas terminé à l'étape $(i-1)$, nous déduisons que $p < \frac{1}{3}K$.

Pour conclure il ne reste qu'à remarquer que notre algorithme termine toujours en calculant une décomposition valide. S'il n'était pas le cas, comme le parcours est censé s'arrêter après avoir visité le nœud n_i, il devrait être $p+q+s > \frac{2}{3}K+s$, qui implique aussi $r > \frac{1}{3}K$: mais cela comporterait une contradiction, puisque $p+q+r+s > K+s$. □

Lemme 53. *Étant donnée une mini triangulation $\mathcal{ST}_i$, il est possible de la décomposer en deux sous-triangulations ayant taille au plus $\frac{2}{3}\|\mathcal{ST}_i\|$ (où $\|\mathcal{ST}_i\|$ exprime la taille de $\mathcal{ST}_i$ en terme de micro triangulations contenues). De plus, cette procédure ne fait intervenir qu'une seule décomposition de micro triangulation.*

Démonstration. Encore une fois la stratégie suivie consiste à calculer un arbre couvrant de la carte G_i (la carte décrivant les adjacences entre micro triangulations de $\mathcal{ST}_i$) : comme résultat nous avons un arbre $\mathcal{B}$, qui n'est pas forcement binaire, ayant $\|\mathcal{ST}_i\|$ nœuds.

L'application du lemme 16 à $\mathcal{B}$, une fois choisi le paramètre M de la bonne manière, permet d'obtenir une décomposition en deux sous-arbres

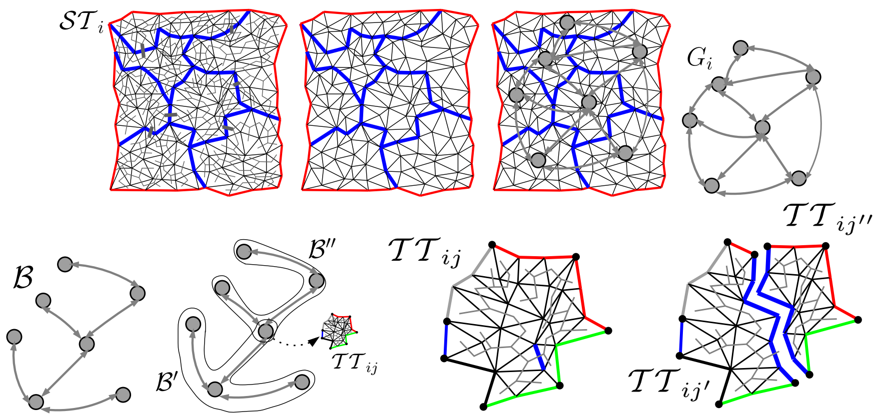


FIG. 6.4 – Ces images illustrent quelques étapes de la stratégie de décomposition du lemme 53 : la décomposition de la micro triangulation $\mathcal{TT}_{ij}$ s'effectue à l'aide du lemme 52.

$\mathcal{B}', \mathcal{B}''$, ayant en commun au plus un nœud n_{ij} et dont les tailles sont au plus $\leq \frac{2}{3}\|\mathcal{B}\|$. Soit n_{ij} le nœud appartenant à $\mathcal{B}'$ et $\mathcal{B}''$, et désignons par $\mathcal{TT}_{ij}$ la micro triangulation correspondante, ainsi que s le nombre de ses côtés. Nous allons construire un arbre binaire $\mathcal{B}_2$ à $\|\mathcal{TT}_{ij}\| + s$ nœuds, auxquels seront associés des poids non négatifs satisfaisant le lemme 52.

Commençons par construire un arbre couvrant $\mathcal{B}_2$ du graphe dual de la triangulation $\mathcal{TT}_{ij}$: ceci est un arbre binaire à $\|\mathcal{TT}_{ij}\|$ nœuds. Pour chaque côté de $\mathcal{TT}_{ij}$ il existe une arête a_k dans $\mathcal{B}$ connectant n_{ij} à l'un de ses nœuds voisins (ceux-ci correspondent aux micro triangulations adjacentes à $\mathcal{TT}_{ij}$ dans $\mathcal{ST}_i$). Étant donnée une telle arête a_k, soit $\mathcal{B}_k$ le sous-arbre obtenu de $\mathcal{B}$ en coupant a_k : cela donne un sous-arbre pour chaque côté de $\mathcal{TT}_{ij}$, contenant au plus $\|\mathcal{B}_k\| \leq \frac{2}{3}\|\mathcal{ST}_i\|$ nœuds. Initialisons à 0 le poids de tous ses nœuds (internes et externes). Pour chaque côté de $\mathcal{TT}_{ij}$, considérons un nœud n_k correspondant à un triangle du bord $\triangle_k$: nous allons attacher une feuille à n_k dans $\mathcal{B}_2$, ayant poids $\|\mathcal{B}_k\|$. Le résultat de cette procédure est un arbre binaire de taille $\|\mathcal{TT}_{ij}\| + s$, ayant poids total $\|\mathcal{ST}_i\|$: tous les nœuds internes ont poids égal à 0 et une sélection des feuilles ont des poids non négatifs satisfaisant le lemme 52. L'applications du lemme 52 à $\mathcal{B}_2$ retourne deux sous-arbres $\mathcal{B}_2', \mathcal{B}_2''$ dont les poids sont au plus $\frac{2}{3}w(\mathcal{B}_2)$. Nous avons ainsi une décomposition de $\mathcal{TT}_{ij}$ en deux nouvelles sous-triangulations $\mathcal{TT}_{ij'}$ et $\mathcal{TT}_{ij''}$: ce qui induit une décomposition de $\mathcal{ST}_i$ en deux mini triangulations $\mathcal{ST}', \mathcal{ST}''$, contenant chacune au plus $\frac{2}{3}\|\mathcal{ST}_i\|$ micro triangulations. Il suffit d'ajouter $\mathcal{TT}_{ij'}$ (resp. $\mathcal{TT}_{ij''}$) à l'union des micro triangulations adjacentes, correspondantes aux nœuds de $\mathcal{B}'$ (resp. aux nœuds de $\mathcal{B}''$). Le résultat des étapes précédentes est la décomposition de $\mathcal{ST}_i$ en deux 2 sous-triangulations $\mathcal{ST}', \mathcal{ST}''$ valides : elles définissent une partition des triangles de $\mathcal{ST}_i$, formant deux sous-régions connexes, étant donnée la manière dont nous avons découpé l'arbre $\mathcal{B}_2$. Pour conclure, il reste à remarquer que si l'une des nouvelles micro triangulations créées, par exemple $\mathcal{TT}_{ij'}$, a une taille ne respectant pas les contraintes établies ($< \frac{1}{12} \lg m$ triangles), il suffit alors de la fusionner avec une micro triangulation voisine dans $\mathcal{ST}'$. Leur union pouvant contenir jusqu'à $(\frac{1}{4} + \frac{1}{12}) \lg m \leq \frac{1}{3} \lg m$ triangles, il suffit de lancer la procédure de décomposition (pour micro triangulations) pour obtenir deux micro triangulations valides ayant chacune entre $\frac{1}{12} \lg m$ et $\frac{2}{9} \lg m$ triangles. □

Corollaire 54. *La décomposition d'une mini triangulation $\mathcal{ST}_i$, ainsi que la mise à jour de la structure de données peut se réaliser en temps $O(\lg^2 m)$ amorti.*

Démonstration. En ce qui concerne la mise à jour des structures de données, il faut créer la représentation en mémoire de deux nouvelles mini triangulations $\mathcal{ST}', \mathcal{ST}''$, donc la construction nécessite un temps $O(\lg^2 m)$ amorti, ainsi que la suppression de l'ancienne mini triangulation $\mathcal{ST}_i$: tous les tableaux concernés par cette opération ont taille $O(\lg^2 m)$. Rappelons que la stratégie de décomposition est telle que le nombre de nouveaux sommets multiples créés est négligeable, ainsi que le nombre de références (entre mini triangulations) à mettre à jour : la modification de tous ces pointeurs peut s'effectuer en temps $O(\lg m)$.

Puisque $O(1)$ côtés de micro triangulations ont été modifiées, au plus $O(\lg m)$ nœuds de la carte G ont changé de degré, ce qui garantit que la mise à jour de tous les références concernant $\mathcal{ST}', \mathcal{ST}''$ et provenant de micro triangulations adjacentes demande un temps $O(\lg^2 m)$ amorti. Pour cette même raison la modification des étiquettes des arcs de la carte G nécessite aussi un temps $O(\lg^2 m)$ amorti. Et finalement, la mise à jour des références stockées dans les tableaux S_i peut se faire aussi en temps $O(\lg^2 m)$ amorti, car le nombre de voisins de $\mathcal{ST}_i$ qui ont été modifiés est constant (lorsqu'il est une reallocation dans un table, le nombre de références à modifier relatives à des nœuds voisins dans G est au plus $O(\lg m)$). □

6.6 Mise à jour des tableaux dynamiques

Pour chacune des structures stockant l'information relative à une mini triangulation nous donnons une description des opérations de mise à jour ainsi qu'une analyse de leur complexité : ces opérations sont largement utilisées lors de la mise à jour de la représentation après décomposition/fusion des mini et micro triangulations.

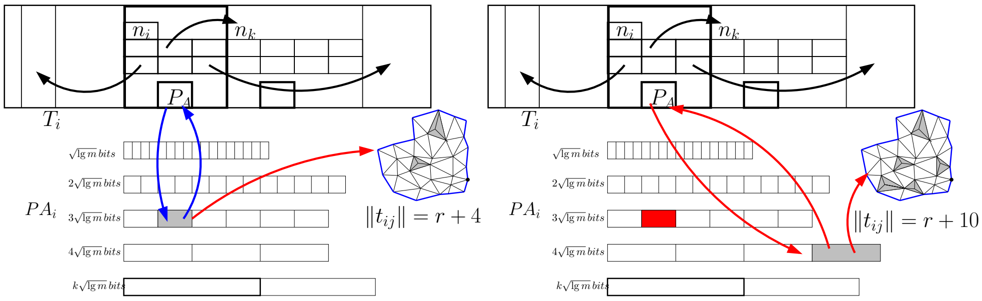


FIG. 6.5 – Insertion de sommets : si l'insertion de sommets n'affecte pas de manière significative la taille r' d'une nouvelle triangulation $\mathcal{TT}'$, qui reste toujours valide, seules les références (flèches rouges) vers le catalogue A sont modifiées (ce cas est esquissé sur la gauche). Parfois il est nécessaire de "rééquilibrer" la collection PA_i, lorsque la longueur $2.17r'$ d'une référence relative à $\mathcal{TT}'$ dépasse un certain seuil, cette phase pouvant s'effectuer en temps $O(1)$ amorti (figure à droite).

6.6.1 Collection PA_i

Les éléments contenus dans les tableaux de la collection PA_i ne nécessitent une mise à jour que lorsque la structure combinatoire (ou le coloriage) d'une micro triangulation a subi une modification (ce cas est illustré par la figure 6.5).

Lemme 55. *La mise à jour de la représentation implicite d'une micro triangulation (modifiée localement) peut s'effectuer en temps $O(1)$ amorti.*

Démonstration. On calcule d'abord la nouvelle représentation et la référence correspondante stockée dans le catalogue A : cela se fait simplement à l'aide de l'information stockée dans la représentation explicite. Si cette référence est environ de la longueur de l'ancienne il suffit de la recopier dans la même case. Sinon, il faut effectuer certaines opérations pour rééquilibrer le tableau dynamique concerné, afin d'éviter le gaspillage d'espace inutilisé :

- on efface d'abord $PA_i[k]$ (tableau dynamique ayant des éléments de taille $k\sqrt{\lg m}$) l'élément contenant l'ancienne référence ;

- il faut créer un nouvel élément dans un tableau $PA_i[k']$ (k' est choisi de telle manière que la nouvelle référence peut se stocker sur $k'\sqrt{\lg m}$ bits) ;

- finalement la case contenant l'ancienne référence (case rouge dans la figure 6.5) qu'on vient d'effacer, doit être rempli avec le dernier élément du tableau $PA_i[k]$: le déplacement de cet élément demande la mise à jour de certaines références contenues dans le tableau T_i, ce qui peut se faire facilement à l'aide du pointeur "backward" vers ce dernier.

Puisque l'accès en écriture/lecture aux tableaux extensibles peut s'effectuer en temps $O(1)$, la complexité est dominée par le coût de la création d'un nouvel élément dans la collection PA_i, qui nécessite $O(1)$ amorti. □

Lemme 56. *La mise à jour des références concernant une nouvelle micro triangulation obtenue par la procédure de décomposition décrite à la section 6.5 nécessite un temps $O(\lg m)$ amorti.*

Il suffit d'observer que dans ce cas l'opération plus coûteuse concerne le calcul de la nouvelle représentation implicite : celle-ci est donnée par une référence contenue dans le tableau A, et comme pour le cas statique elle peut se retrouver par recherche binaire en temps $O(\lg m)$.

6.6.2 Modifications topologiques dans le graphe G_i

Certaines modifications du graphe G_i sont nécessaires lors de la décomposition ou fusion des micro triangulations (après suppression de sommets ou flip d'arêtes).

Modifications dans le tableau T_i

La création (ou suppression) d'une micro triangulation nécessite l'insertion (suppression) de nouveaux éléments dans le tableaux extensible T_i : chaque nouveau élément est ajouté à la dernière position, alors que la suppression d'un élément demande la copie (et donc la suppression) du dernier élément du tableau (comme dans le cas des tableaux de la collection PA_i, cette stratégie est nécessaire afin de minimiser la quantité d'espace, qui sinon serait gaspillé). L'insertion et recopie d'éléments dans T_i implique aussi la mise à jours de toutes les références concernant les éléments qui ont subi une modification : cela inclue les pointeurs "backward" contenus dans les tableaux PA_i, ainsi que tous les pointeurs stockés dans les tableaux de la collection PE_i.

Lemme 57. *L'insertion/suppression d'un élément dans le tableau T_i, ainsi que la mise à jour de toutes les références associées peut se réaliser en temps $O(\lg m)$ amorti.*

En effet, l'insertion/suppression d'un élément dans T_i demande un temps $O(1)$ amorti ; la modification de tous les pointeurs et références stockées dans les tableaux de PA_i et PE_i se fait en temps $O(\lg m)$, car il existe au plus $O(\lg m)$ éléments qui nécessitent une mise à jour, étant donné la taille du bord et le nombre de voisin d'une micro triangulation (rappelons que l'écriture/lecture se fait en temps constant dans le cas des tableaux extensibles).

Mise à jour des arcs de G_i

L'insertion ou mise à jour d'arcs dans la carte G_i produit essentiellement deux sortes de modifications au niveau de la structure de données.

- Premièrement il faut ajouter/supprimer des éléments dans les tableaux de la collection PE_i stockant les références entre micro triangulations voisines. Rappelons que les demi arcs incidents à un nœud donné v de G_i sont stocké de manière consécutive afin de respecter l'arrangement cyclique (structure de carte) autour de v. De plus, ces arcs sont regroupés en un certaine quantité de manière à tenir compte du degré de leur nœud incident v : un tableau donné de la collection PE_i contient les listes des demi arcs relatives à des nœuds ayant environ le même degré. Si le degré de v change dramatiquement, le tableau de PE_i stockant précédemment les arcs incidents à v ne suffit plus : il faudra recopier la liste entière de ces arcs dans un autre tableau de la collection spécialement conçu pour stocker les informations concernant des nœuds ayant un degré un peu plus grand (ces opérations nécessitent naturellement la suppression et le déplacement d'un certain nombre d'éléments contenus dans le premier tableau extensible).

- le changement du degré de v implique aussi la mise à jour des étiquettes des demi arcs de la carte G_i ayant v somme nœud *cible* : dans le cas général il existe $O(\lg m)$ de telles étiquettes à modifier globalement dans une mini triangulation. Ces remarques conduisent au lemme suivant :

Lemme 58. *Les modifications locales dans la carte G_i nécessitent un temps $O(\lg m)$ amorti.*

Du point de vue de la complexité de calcul, l'insertion ou la suppression d'éléments dans la collection PE_i nécessitent jusqu'à $O(\lg m)$ opérations sur des tableaux extensibles, chacune pouvant se réaliser en temps $O(1)$ amorti. La mise à jour des étiquettes nécessite $O(\lg m)$ opérations de lecture/écriture relativement à la collection PE_i. Toutes les références dans le tableau T_i peuvent aussi être mises à jour en temps $O(\lg m)$.

6.7 Mise à jour de la triangulation

Insertion d'un nouveau sommet. L'insertion d'un sommet de degré 3 concerne toujours l'intérieur d'une micro triangulation : ce qui garantit que la modification des relations de voisinage décrite par la carte G est nécessaire seulement lorsque la taille d'une micro triangulation ne respecte pas les contraintes de taille (entre $\frac{1}{12}\lg m$ et $\frac{1}{4}\lg m$).

Ainsi, si la taille d'une micro triangulation n'est pas affectée de manière significative lors de l'insertion d'un sommet (si le nombre de triangles est au plus $\frac{1}{4}\lg m$), alors le calcul et mise à jour de la nouvelle micro triangulation se font simplement à l'aide des résultats précalculés et stockés dans le tableau A (la mise à jour de la représentation implicite de la triangulation nécessitant un temps $O(\lg m)$). Si la taille dépasse la valeur $\frac{1}{4}\lg m$, alors il est nécessaire d'effectuer la décomposition d'une micro triangulation, ce qui demande un temps $O(\lg m)$ amorti, en accord au lemme 51. Lorsque l'insertion d'un nouveau sommet provoque la décomposition d'une micro triangulation, cette dernière impliquant le dépassement des contraintes de taille pour une mini triangulation, alors on peut effectuer une décomposition à l'aide du lemme 53 : dans ce cas la mise à jour de toutes les références entre micro/mini triangulations voisines, ainsi que les représentations de G et F se font en temps $O(\lg^2 m)$ amorti.

Ces deux derniers événements se présentent assez "rarement" (tous les $\Theta(\lg m)$ insertions pour le cas des micro triangulations, et tous les $\Theta(\lg^2 m)$) insertions pour les mini triangulations), ce qui implique le lemme suivant :

Lemme 59. *L'insertion d'un sommet de degré* 3 *peut s'effectuer en temps* $O(1)$ *amorti.*

Suppression d'un sommet de degré 3**.** Si le sommet concerné n'est pas un sommet multiple, la stratégie de mise à jour mime celle adoptée pour le cas d'une insertion : puisque il n'y a pas de modifications topologiques de la décomposition (et donc de la carte G), les procédures de décomposition et mise à jour d'une micro/mini triangulation ne sont nécessaires que si les contraintes de taille sont violées.

Lorsque il s'agit de supprimer un sommet multiple, des modifications dans les relations de voisinage entre micro triangulations peuvent affecter de manière considérable la phase de mise à jour. En général, il n'existe pas de solution meilleure que la simple application des procédures de décomposition (pour mini et micro triangulations) : ainsi, après avoir fusionné les micro triangulations concernées (au plus 3), on peut simplement faire appel au lemme 51, et éventuellement au lemme 53 (l'application de cette procédure est nécessaire un nombre constant de fois).

Lemme 60. *La suppression d'un sommet de degré* 3 *nécessite un temps* $O(\lg^2 m)$ *amorti.*

Flip d'arêtes. Si le flip ne concerne qu'une arête e contenue dans l'intérieure d'une micro triangulation, la mise à jour se fait simplement à l'aide des informations stockées dans le catalogue A : les relations de voisinage restant inchangées, il suffit de modifier la référence G^A_{ij} stockée dans la collection PA_i. Si l'arête e appartient au bord entre deux micro triangulations, le flip peut causer le changement de la topologie des régions concernées, comme illustré dans la figure 6.6). Comme auparavant, lorsque la modification de la topologie de la décomposition ou des tailles des micro/mini triangulations le rendent nécessaire, il faut faire appel aux procédures de décompositions décrites par les lemmes 51 et 53 : ces procédures s'appliquent à la région obtenue en fusionnant les micro/mini triangulations concernées par le flip. L'implantation de la décomposition d'une micro triangulation (s'il n'y a qu'une mini triangulation concernée) peut s'effectuer en temps

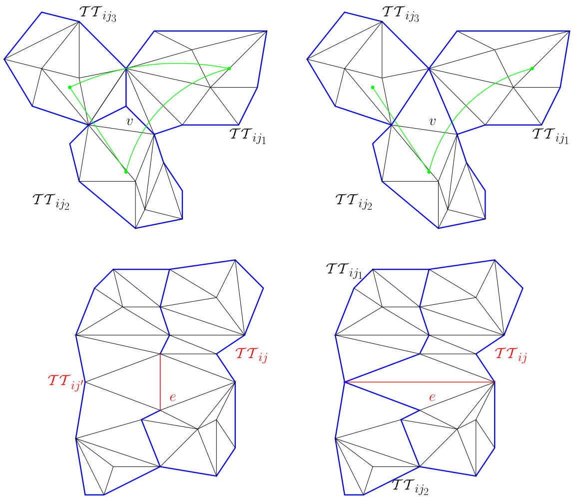


FIG. 6.6 – Changements topologiques dans la décomposition et la carte G.

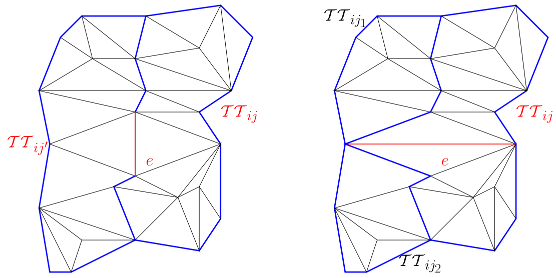


FIG. 6.7 – Flip d'arêtes modifiant la topologie de la décomposition.

$O(\lg m)$ amorti, tandis que l'application de la décomposition d'une mini triangulation (lorsque plusieurs mini triangulations sont touchées) nécessite un temps $O(\lg^2 m)$ amorti, ce qui permet d'établir le lemme suivant :

Lemme 61. *L'implantation d'un flip d'arête nécessite un temps $O(\lg^2 m)$ amorti.*

6.8 Remarques finales

6.8.1 Navigation dans la triangulation

La stratégie de navigation mime exactement la procédure détaillée au chapitre 5 (que nous ne reportons pas ici), la seule différence étant l'utilisation des collections de tableaux dynamiques introduits à la section 6.4. Il suffit juste de rappeler que toutes les références et pointeurs utilisés sont stockés sur $O(\lg m)$ bits, et que tous les tableaux dynamiques décrivant la représentation des cartes G et F permettent l'accès en lecture en temps $O(1)$, pire cas.

6.8.2 Attacher de l'information géométrique

En analogie avec le cas statique, nous pouvons enrichir la structure de données afin de permettre d'associer des données externes (par exemple de type géométrique) à $\mathcal{T}$: ici nous voulons attacher des données de taille $O(\lg m)$ aux cellules élémentaires de $\mathcal{T}$ (sommets ou triangles). La solution proposée consiste à associer cette information directement aux nœuds et arêtes de la carte G_i. Plus précisément, nous allons ajouter une nouvelle collection de tableaux extensibles PD_i, à la description de l'organisation mémoire concernant la mini triangulation $\mathcal{ST}_i$, de la manière suivante :

• tous les éléments dans la collection PD_i sont de taille fixe $O(\lg m)$, chacun contenant l'information géométrique relative à un sommet donné (ses coordonnées). Les données relatives à une même micro triangulation $\mathcal{TT}_{ij}$ sont stockées dans des champs consécutifs et triées de manière à respecter l'ordre choisi sur le sommets de $\mathcal{TT}_{ij}$. Ces champs sont aussi regroupés, comme déjà fait pour les tableaux de la collection PE_i, de façon à que le k-ème tableau de la collection contient des groupes de $k\sqrt{\lg m}$ champs, correspondants aux micro triangulations ayant entre $(k-1)\sqrt{\lg m}$ et $k\sqrt{\lg m}$ sommets.

La collection PD_i est utilisée pour stocker tous les sommets internes (à une micro triangulation), ainsi qu'une sélection des sommets du bord, de telle sorte que les sommets partagés par deux micro triangulations n'apparaissaient qu'une seule fois. Un point crucial concerne les sommets multiples, partagés par plusieurs micro triangulations : pour chaque mini triangulation $\mathcal{ST}_i$, nous stockons la liste des sommets multiples, avec leurs données géométriques, dans un tableaux extensible supplémentaire. Puisqu'il existe au plus $O(\lg^2 m)$ tels sommets dans $\mathcal{ST}_i$, chacun peut être référencé avec des pointeurs locaux sur $O(\lg \lg m)$ bits, qui sont directement stockés dans les demi-arêtes de la carte G_i. Globalement cette stratégie ne fait qu'ajouter une quantité négligeable d'espace. De manière similaire, il faut stocker des données associées aux sommets multiples qui sont partagés par plusieurs mini triangulations (ainsi chaque mini triangulation contient les données relatives à une sélection de ses sommets).

Coût d'une mise à jour

La modification locale de la triangulation implique aussi la mise à jour des tableaux contenant l'information géométrique. En particulier, dans le cas simple d'insertion de sommet la compléxité d'une mise à jour augmente, lorsque on veut maintenir des données externes.

Lemme 62. *Si des données externes (sur $O(\lg m)$ bits) sont associées aux sommets de $\mathcal{T}$, la représentation permet l'insertion de sommets en temps $O(\lg m)$ amorti, la suppression de sommets et le flip d'arêtes pouvant s'effectuer en temps $O(\lg^2 m)$ amorti.*

Le fait de permettre l'accès aux données géométriques stockées dans les tableaux de la collection PD_i présuppose un ordre sur l'ensemble des sommets. Encore une fois le problème de la mise à jour concerne essentiellement les sommets multiples : l'ordre sur ces sommets est implicitement fourni par la façon dont ces sommets sont arrangés dans les tableaux de PD_i, et est soumis à des changements provenant de l'insertion de nouveaux sommets. Contrairement à ce qui arrive lors de la modification des tableaux dans PA_i, la mise à jour des références relatives aux tableaux de la collection PD_i est une opération plus coûteuse.

Chapitre 7

Représentations succinctes optimales de cartes planaires

7.1 Notre contribution

Nous allons illustrer comment le schéma du chapitre 4 nous fournit les première représentations succinctes pour des structures de données géométriques telles que les maillages (triangulaires et polygonaux) de la sphère. Plus précisément, l'optimalité en terme de ressources mémoire est due à deux ingrédients : des nouvelles stratégies de décomposition du graphe initial en micro régions, basées sur des arbres couvrants spéciaux ; la construction d'un catalogue exhaustif des micro régions qui ne contienne pas exactement juste les objets appartenant à la même classe du graphe initial. Cette dernière propriété, absente dans les travaux précédents, joue un rôle crucial dans l'application de notre schéma, et nous fait supposer que notre stratégie est assez générale et pourrait conduire à de nouvelles représentations succinctes ou compactes pour d'autres classes de cartes planaires.

Les résultats principaux de ce chapitre[1] sont exprimés par le théorème suivant :

Théorème 63. *Il existe des représentations succinctes de*

- *triangulations planaires (sans bord) nécessitant asymptotiquement* 1.62 *bits par triangle,*
- *de cartes planaires* 3*-connexes nécessitant asymptotiquement* 2 *bits par arête.*

Ces deux représentations permettent en temps $O(1)$ *la navigation locale en utilisant une quantité de mémoire auxiliaire de taille* $O(\frac{n \lg \lg n}{\lg n})$ *(*n *étant le nombre de sommets de la carte).*

[1] Les deux représentations succinctes présentées dans ce chapitre ont été l'objet d'une publication à SoCG 2006 [28].

7.2 Cartes planaires 3-connexes

Dans cette section nous allons décrire une classe particulière de micro quadrangulations, ainsi qu'un algorithme qui décompose une quadrangulation irréductible quelconque en micro régions appartenant à cette classe.

Cette décomposition satisfait la relation 4.1 et le schéma mini-micro peut ainsi s'appliquer et conduit à une représentation succincte.

Cartes 3-connexes et quadrangulations. Rappelons brièvement la construction décrite à la section 3.3, qui est une légère variante de la notion standard de carte duale : étant donnée une carte planaire 3-connexe M, nous colorons ses sommets en noir et plaçons un sommet blanc au milieu de chaque face. En triangulant à partir des sommets blancs nous obtenons une quadrangulation Q formée des nouvelles arêtes (chaque face est constituée de la fusion de deux triangles incidents à une arête de M).

De manière intuitive, la construction ci-dessus fait de la quadrangulation Q une représentation implicite d'une carte planaire, induite avec une coloration de ses sommets : les sommets blancs (resp. noirs) de Q correspondent aux faces (resp. sommets) de la carte M (voir figure 3.6). Plus précisément, tester des relations d'adjacences entre sommets et faces dans la carte M revient à répondre à des requêtes de voisinage entre sommets qui sont incidents à la même face dans la quadrangulation Q. Il va être bien plus facile de décrire notre construction en terme de quadrangulations. Puisque les quadrangulations (irréductibles) peuvent être interprétées comme une représentation des cartes planaires 3-connexes, le résultat de cette section fournit directement une représentation succincte pour cette classe de cartes.

7.2.1 Application de notre schéma

La stratégie de décomposition que nous allons concevoir tire partie de l'existence d'une correspondance entre les dissections irréductibles et une classe spéciale d'arbres couvrants, décrite à la section 3.3 et introduite dans [42] : plus précisément, le théorème 27 affirme que les arbres binaires à n nœuds internes sont en bijection avec les dissections irréductibles ayant n sommets.

Supposons donc que nous partons d'une dissection irréductible Q d'un hexagone en faces quadrangulaires. Suivant la stratégie détaillée dans [42], il est possible d'effectuer un algorithme d'ouverture sur Q, fournissant un arbre binaire, dont la clôture complète [42, Lemma 2] est exactement la dissection originale (associée à une carte planaire 3-connexe). Plus précisément on obtient un arbre binaire B ayant n nœuds, $n+2$ bourgeons et $(n-1)$ arêtes internes.

S'agissant d'arbres binaires, nous pouvons faire appel à la procédure (décrite au chapitre 2) : le lemme 15 nous garantit de pouvoir calculer en temps linéaire une décomposition d'un tel objet en sous arbres ayant à peu près la même taille. Il est ainsi possible d'obtenir, par l'application réitérée de cette procédure, une partition de $\mathcal{B}$ mini arbres ayant entre $\frac{1}{3}\lg^2 n$ et $\lg^2 n$ nœuds, qui sont ensuite décomposés en micro arbres ayant entre $\frac{1}{30}\lg n$ et $\frac{1}{10}\lg n$ nœuds. Une telle décomposition forme une partition des arêtes de $\mathcal{B}$ (qui sont aussi les arêtes de la quadrangulation initiale). Il est à remarquer que seuls les nœuds qui sont racines des micro (resp. mini) arbres, peuvent être partagés par des micro (resp. mini) arbres différents : chacun de ces nœuds (de degré au plus d) est "décomposé" en un nœud racine de degré $d-1$ (dans le sous-arbre *descendant*) et une feuille dans le sous-arbre ancêtre. La manière dont nous avons partitionné $\mathcal{B}$ garantit que le nombre de micro (resp. mini) arbres créés est $\Theta(\frac{n}{\lg n})$ (resp. $\Theta(\frac{n}{\lg^2 n})$).

7.2.2 Décomposition en micro quadrangulations

Le but de cette section est de décrire une façon canonique d'obtenir une décomposition $\{M_1, \ldots, M_p\}$ de $\mathcal{Q}$, à partir de la décomposition de l'arbre recouvrant de $\mathcal{B}$ mentionnée ci-dessus. Considérons un micro arbre, noté $\mathcal{TB}_j$, ayant $n_j \leq \frac{1}{10}\lg n$ nœuds, obtenu par la décomposition de $\mathcal{B}$: nous allons montrer comment produire une carte quadrangulaire de taille $\Theta(\lg n)$ à partir de $\mathcal{TB}_j$. Un micro arbre $\mathcal{TB}_j$ possède n_j nœuds et n_j+2 bourgeons : nous faisons la distinction suivante concernant deux types de feuilles. D'une part, il peut exister des feuilles apparaissant déjà dans l'arbre original $\mathcal{B}$. D'autre part, il peut y avoir des nouvelles feuilles, correspondant à des nœuds internes dans $\mathcal{B}$ avant l'application de la procédure de décomposition : leur arêtes incidentes seront appelées *faux bourgeons*. Remarquons que ces micro arbres sont enracinés (les racines étant les nœuds partagés par plusieurs micro arbres), et que le nombre de faux bourgeons appartenant à un micro arbre n'est pas fixe et varie entre $0 \ldots n_j + 2$, selon la manière dont $\mathcal{B}$ a été décomposé.

Clôture locale

Il suffit maintenant d'appliquer un algorithme de clôture locale (inspiré de celui introduit dans [42] et déjà détaillé à la section 3.3) dont les étapes principales sont décrites ci-dessus (rappelons que dans sa version originale [42], l'algorithme de clôture ne prenait pas en compte la distinction entre faux et vrais bourgeons). Nous commençons par effectuer un parcours en profondeur préfixe (de gauche à droite) de l'arbre $\mathcal{TB}_j$: en partant de sa racine et en marchant le long de ses arêtes (arêtes internes, vrais bourgeons et faux bourgeons).

– lors de la visite d'un vrai bourgeon, et si ce dernier est précédé par trois nœuds internes (et non par des bourgeons), sa clôture locale consiste à relier son nœud incident au troisième dernier nœud qui le précède sur le bord de

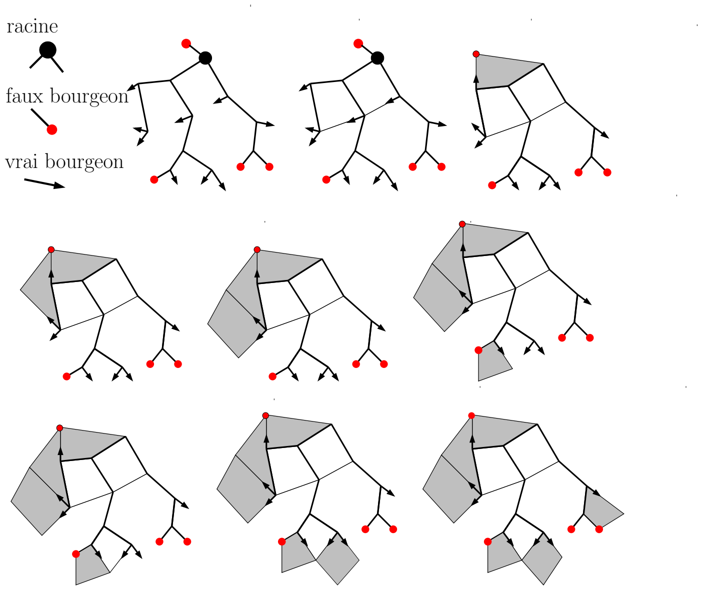


FIG. 7.1 – Ces images illustrent notre clôture locale. Le résultat de cette clôture locale dépend de la distribution des faux bourgeons. Dans cet exemple, l'arbre binaire possède 11 sommets internes, 11 + 2 feuilles (vrais et faux bourgeons, incluant le bourgeon incident à la racine), et peut se coder de manière optimale avec un mot binaire de longueur $2 \cdot 11$: 1101001011001001011000, tandis que la distribution des bourgeons correspondante est définie par le bit-vecteur de taille 13 et poids 4 : 0000100001101.

la face externe, de manière à former une face quadrangulaire (représentée par une face blanche dans la figure 7.1).

– lors de la visite d'un vrai bourgeon s qui n'est pas précédé par 3 vrais nœuds (nœuds originaux existants dans $\mathcal{TB}_j$) sa clôture locale consiste à attacher une *quadrangulaire face fictive* (une face grise dans la figure 7.1) au bord de $\mathcal{TQ}_j$, de telle manière que cette face fictive est incidente à s, ne renfermant aucun faux bourgeon ni arête fictive (arête incidente à une face fictive) déjà créée. De manière similaire, les sommets incidents aux faces fictives, n'existant pas dans le micro arbre, seront appelés *sommets fictifs*.

Et finalement, il reste à souligner que nous n'effectuons pas la fusion des faux bourgeons (indiqués par des petits cercles rouges dans la figure 7.1). De cette manière nous produisons une carte planaire dont toutes les faces internes ont degré 4, et ayant une face externe de taille arbitraire : les arêtes internes à l'arbre peuvent éventuellement apparaître plusieurs fois incidentes à la face externe.

Catalogue des micro quadrangulations

Les cartes ainsi obtenues sont connexes et possède un bord simple, et seront aussi appelées *micro quadrangulations* (le résultat de cette procédure de clôture est illustré par les images de la figure 7.1).

Ici nous considérons le catalogue $\mathcal{D} = \{\mathcal{D}_{p,k,w}\}$ contenant toutes les quadrangulations qu'on peut obtenir avec le procédé ci-dessus, dont les arêtes du bord sont partitionnées en côtés. Plus précisément, un objet M_j de $\mathcal{D}_{p,k,w}$ est une micro quadrangulation $\mathcal{TQ}$ obtenue avec notre clôture locale à partir d'un micro arbre $\mathcal{TB}$ ayant p nœuds, $p+2$ bourgeons and w faux bourgeons (le paramètre de taille étant le nombre p de nœuds dans l'arbre). De plus, $\mathcal{TQ}$ est induite avec une partition des arêtes de son bord en k côtés. Il est facile de vérifier[2] qu'étant donné un micro arbre ayant p nœuds la micro quadrangulation correspondante possède un bord de taille au plus $4p+6$. Alors on peut établir la relation suivante concernant la taille de notre catalogue :

$$\lg |\mathcal{D}_{p,k,w}| \leq \lg \left(2^{2p} \cdot \binom{4p+6}{k} \cdot \binom{p+2}{w} \right) \approx 2p + k \lg p + w \lg p + O(k),$$

(car il existe $\binom{p+2}{w}$ façons de distribuer les faux bourgeons, et $\binom{4p+6}{k}$ façons de partitionner les arêtes du bord en côtés ; on a aussi $w \leq k$). Ceci garantit déjà que le stockage des éléments du catalogue (et de leurs représentations explicites) nécessite une quantité négligeable de ressource mémoire : rappelons que le nombre p de nœuds d'un micro arbre est au plus $\frac{1}{10} \lg n$.

Distribution des bourgeons

Une fois l'arbre recouvrant $\mathcal{B}$ décomposé, il ne reste qu'à indiquer une règle pour assigner les vrais bourgeons incidents aux nœuds racines partagés par les micro arbres. Nous procédons de la façon suivante, en supposant que le nœud v est partagé par deux arbres $\mathcal{TB}_{j'}$ et $\mathcal{TB}_{j''}$, le bourgeon éventuellement incident étant noté s (figure 7.2) :

- s'il existe, dans l'arbre descendant $\mathcal{B}$, une arête interne e incidente à v à la gauche de s, alors nous assignons le bourgeon s au micro arbre $\mathcal{TB}_{j''}$ ayant v pour racine et contenant e ;

- autrement s est le frère le plus à gauche du nœud v dans $\mathcal{B}$, et le bourgeon s est donc assigné à l'arbre ancêtre $\mathcal{TB}_{j'}$.

[2] En ce qui concerne les arêtes internes, éventuellement incidentes à la face externe, leur nombre est $p-1$ et leur contribution à la taille du bord est donc au plus de $2(p-1)$. D'autre part l'apport des faces fictives est maximum lorsque une telle face est produite lors de la fusion d'un (vrai) bourgeon qui est immédiatement précédé par un faux bourgeon : puisque il existe $p+2$ bourgeons, au plus $\lceil \frac{p+2}{2} \rceil$ faces fictives peuvent se présenter, chacune contribuant avec 4 arêtes. En conclusion la taille du bord d'une micro quadrangulation peut être bornée par : $2(p-1) + 4(\frac{p+2}{2} + 1) = 4p + 6$.

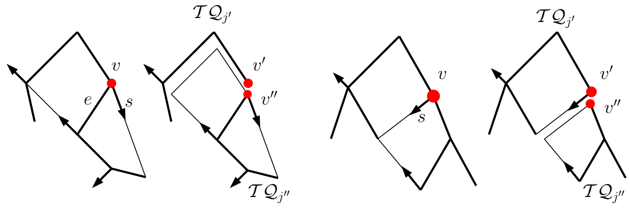


FIG. 7.2 – Ces figures illustrent la distribution des bourgeons et des faces entre deux micro quadrangulations qui partagent un nœud v (un nœud multiple, représenté par un petit cercle).

Il est immédiat d'observer que toutes les faces de la quadrangulation initiale ont été assignées à exactement à une micro quadrangulation (n'étant donc pas partagées), car incidentes chacune à un vrai bourgeon.

Vérification des hypothèses

Une fois définis le catalogue des micro quadrangulations et la décomposition de Q en mini et micro régions, il ne reste qu'à vérifier la validité des hypothèses introduites au chapitre 4, qui conduisent au résultat suivant (version spécialisée au cas des quadrangulations du théorème 38) :

Lemme 64. *Étant donnée une quadrangulation $\mathcal{Q}$ ayant n sommets, notre stratégie de décomposition produit une partition de $\mathcal{Q}$ en micro quadrangulations $\{\mathcal{TQ}_1, \ldots, \mathcal{TQ}_p\}$ satisfaisant les hypothèses 1, 2, 3, 4 et 5, qui conduit ainsi à une représentation succinctes de $\mathcal{Q}$ nécessitant asymptotiquement $2n + o(n)$ bits.*

Démonstration. L'hypothèse d'additivité est vérifiée, puisque les nœuds dans les quadrangulations (nœuds des micro arbres couvrants) ne sont pas partagés par les micro quadrangulations.

La quadrangulation $\mathcal{Q}$ est décomposée en micro quadrangulations $\mathcal{TQ}_j$, chacune contenant entre $\frac{1}{30} \lg n$ and $\frac{1}{10} \lg n$ nœuds (nœuds du micro arbre recouvrant correspondant) : ici la constante c introduite à la section 4.2 est ainsi égale à $\frac{1}{10}$.

Le graphe G utilisé pour décrire les adjacences entre micro triangulations est une carte planaire ayant des faces de degré au moins 3. Ceci se démontre par l'absurde : s'il y avait une face de degré 2, incidente à des arcs multiples, celle-ci pourrait se contracter, puisque les côtés correspondantes sont deux arcs consécutifs et partagés par les même micro quadrangulations adjacentes (rappelons que chaque micro région est une carte planaire connexe, dont les arêtes peuvent être doublement incidentes à la face infinie, ainsi comptant deux fois comme arêtes du bord).

La formule d'Euler nous assure que le nombre d'arcs de G (et donc le nombre de côtés) est $O(\frac{n}{\lg n})$. Les hypothèses 1, 2, 3, 4 et 5 sont vérifiées

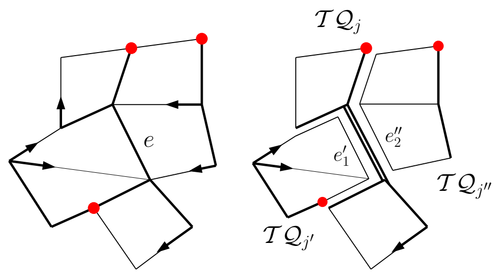


FIG. 7.3 – Ces images montrent les possibles relations d'adjacence entre micro quadrangulations : puisque les arêtes peuvent être doublement incidentes à la face externe, deux micro quadrangulations $\mathcal{TQ}_{j'}$ et $\mathcal{TQ}_{j''}$ peuvent ne pas être voisines (dans la carte G) bien qu'elles "partagent" une arête e dans la quadrangulation initiale $\mathcal{Q}$.

par la décomposition $\{\mathcal{TQ}_1, \ldots, \mathcal{TQ}_p\}$, ainsi le lemme 38 garantit l'existence d'une représentation succincte pour la classe des quadrangulations, atteignant asymptotiquement la borne optimale de 2 bits par arête.

Une micro quadrangulation coloriée est complètement spécifiée par : un mot de Dyck de longueur $2n_j$ (pour coder l'arbre recouvrant), un mot binaire de longueur n_j+2 et poids w_j (indiquant la distribution des faux bourgeons) et un mot binaire de longueur $4n_j+6$ et poids k_j (décrivant la partition des arêtes du bord de $\mathcal{TQ}_j$ en côtés).

Ainsi $\mathcal{TQ}_j$ peut être codée par une référence à un élément du catalogue $\mathcal{D}$, dont le coût est $2n_j+w_j \lg(n_j+2)+k_j \lg(4n_j+6) \leq 2n_j+2k_j(\lg n_j+O(1))$, comme $w_j \leq k_j$ (soit $\alpha = \beta = 2$ dans l'équation 4.1).

La constante c est choisie de telle façon que le tableau A contenant les représentations explicites des éléments de $\mathcal{D}$, nécessite $o(n)$ bits (rappelons que $n_j \leq \frac{1}{10} \lg n$). □

7.2.3 Représentation succincte des cartes 3-connexes

Étant donnée une carte 3-connexe ayant e arêtes, nous allons d'abord construire une représentation succincte de la quadrangulation associée : puisque l'arbre recouvrant correspondant possède $e-5$ nœuds internes (sommets du graphe primal et dual) notre codage nécessite asymptotiquement de $2(e-5)+o(e) = 2e+o(e)$ bits, en accord au lemme 64.

Il suffit d'observer que le fait de vérifier l'adjacence entre deux sommets (resp. deux faces) dans une carte 3-connexe, est équivalent à tester si deux sommets noirs (resp. blancs) sont opposés dans le même quadrangle, dans la carte quadrangulaire associée $\mathcal{Q}$. Cette opération peut s'effectuer efficacement en temps $O(1)$ avec des légères modifications des requêtes d'adjacences

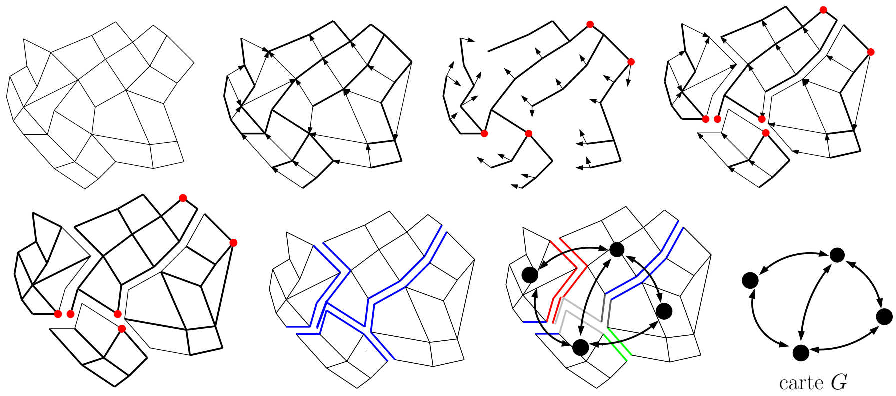


FIG. 7.4 – Ces figures illustrent notre représentation à plusieurs niveaux d'une quadrangulation. La décomposition de l'arbre couvrant $\mathcal{B}$ fournit, via notre clôture locale, une partition de la quadrangulation initiale $\mathcal{Q}$ en micro quadrangulations (les cercles rouges indiquent les nœuds multiples partagés par différents micro arbres). Les relations d'adjacences entre micro triangulations sont décrites pas une carte planaire G.

standard considérées dans le lemme 39 : les détails seront fournis à la section 7.5.

7.2.4 Représentation unique pour les sommets

Un point crucial est de tenir compte de la représentation de certaines cellules élémentaires (ici les sommets), qui peut ne pas être unique : comme observé plusieurs fois auparavant, cet événement peut se présenter lorsque des cellules élémentaires (*cellules multiples*) sont partagées par plusieurs micro régions.

La solution que nous proposons ici consiste à exploiter la correspondance existante entre sommets de la quadrangulation et nœuds de l'arbre couvrant correspondant : il suffit d'associer à chaque sommet dans $\mathcal{Q}$ le nœud correspondant dans l'arbre $\mathcal{B}$.

Les nœuds internes dans le micro arbre $\mathcal{TB}_j$ sont spécifiés de manière unique. Pour les sommets restants (sommets multiples partagées entre micro régions), nous pouvons introduire des *représentants canoniques* qui sont les nœuds correspondants dans une micro région adjacente. En particulier, dans le cas d'un nœud multiple, partagé entre deux micro arbres, le représentant canonique sera le nœud feuille appartenant au micro arbre ancêtre.

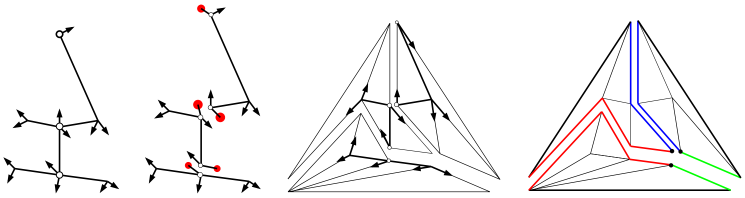


FIG. 7.5 – Ces images illustrent la stratégie adoptée pour construire une décomposition de la triangulation originale $\mathcal{T}$ en micro triangulations. D'abord l'arbre couvrant de $\mathcal{T}$ est décomposé en micro arbres. L'application de notre opération de clôture locale aux micro arbres induit une partition de l'ensemble des triangles de $\mathcal{T}$ en micro triangulations. Chaque micro triangulation induite avec une partition de ses arêtes du bord en côtés (correspondant aux relations de voisinage entre micro régions).

7.3 Triangulations planaires

Triangulations planaires et arbres

Nous allons tirer profit d'une bijection entre triangulations planaires et arbres bourgeonnants (introduite récemment par Poulalhon et Schaeffer [99] et illustrée à la section 3.3). Plus précisément, cette correspondance est décrite par un algorithme de *ouverture/clôture* qui à un arbre recouvrant à n nœuds (ayant exactement deux bourgeons par nœud) associe une triangulation planaire enracinée $\mathcal{T}$ à $n+2$ sommets (induite avec son réaliseur minimal).

7.3.1 Application de notre schéma

Soit $\mathcal{T}$ une triangulation planaire enracinée ayant $n+2$ sommets et notons $\mathcal{B}$ l'arbre recouvrant à n nœuds dont la *clôture complète* donne la triangulation $\mathcal{T}$ (suivant la bijection introduite dans [99]). Puisque il n'y a pas de restriction sur l'arbre $\mathcal{B}$ (qui est juste un arbre ordonné, ayant deux bourgeons par nœud), il n'est pas possible en général d'obtenir une partition de $\mathcal{B}$ en micro arbres (ce qui avait été possible pour les arbres binaires). Notre solution consiste à appliquer une version modifiée et adaptée de la stratégie de décomposition pour arbres ordonnés décrite à la section 2.6.

L'application réitérée du lemme 16 à l'arbre $\mathcal{B}$, fournit une famille de sous arbres de taille $\Theta(\lg^2 n)$, qui sont à leur tour décomposés en micro arbres ayant $\Theta(\lg n)$ nœuds. Cette famille de sous-arbres forme une couverture de l'ensemble des nœuds de $\mathcal{B}$, telle que deux sous-arbres peuvent se couper au plus en leur racine.

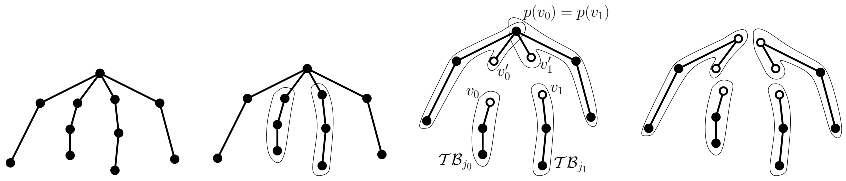


FIG. 7.6 – Ces images montrent le résultat de la stratégie de décomposition (version modifiée de l'algorithme du lemme 16) sur un arbre ordonné, avec paramètre $M = 3$.

Construction des micro arbres

Il est à noter que l'application directe de la stratégie de décomposition mentionnée ci-dessus, conduit à une famille de sous-arbres formant une couverture de l'ensemble des nœuds de $\mathcal{B}$, ne couvrant pas toutes les arêtes : un certain nombre d'arêtes de $\mathcal{B}$ (et donc de $\mathcal{T}$) n'appartiennent à aucun sous-arbre. Afin de distribuer les arêtes entre les sous-arbres, de manière à conserver l'information concernant la position relative des bourgeons autour des nœuds internes, nous allons introduire une version adaptée de l'algorithme de décomposition d'arbres ordonnés (lemme 16), procédant de la manière suivante. Considérons un sous arbre $\mathcal{TB}_{j_0}$, faisant partie de la couverture, qui est retourné à une certaine étape par l'algorithme du lemme 16 : il s'agit d'un arbre "complet", dans le sens qu'il satisfait les contraintes de taille (sa taille minimale est donnée par un paramètre entier $M > 2$). Soit v_0 la racine de $\mathcal{TB}_{j_0}$ et notons $p(v_0)$ son nœud ancêtre dans $\mathcal{B}$. Après avoir retourné $\mathcal{TB}_{j_0}$, nous assignons l'arête $(v_0, p(v_0))$ au sous arbre restant qui contient $p(v_0)$, en attachant ensuite un nœud feuille v'_0, copie de v_0. Ce procédé nous garantit que (voir figure 7.6) :

- chaque sous-arbre satisfait les contraintes de taille (ayant entre $\frac{1}{30}\lg n$ and $\frac{1}{10}\lg n$ nœuds) ;
- chaque arête de $\mathcal{B}$ appartient exactement à un seul sous-arbre ;
- le nombre de nœuds dupliqués (racines des sous-arbres), ainsi que le nombre de sous arbres dans la couverture, est globalement $O(\frac{n}{\lg n})$.

7.3.2 Micro arbres et micro triangulations

Suivant l'approche utilisé pour les quadrangulations, il est possible de définir un *algorithme de clôture* (inspiré de l'opération décrite dans [99]) conduisant à une correspondance entre micro arbres et micro triangulations.

D'abord quelques notations : si un micro arbre $\mathcal{TB}_j$ possède un nœud v de degré 3 qui appartient, en qualité de nœud racine, aussi à un (ou plusieurs) arbre différent $\mathcal{TB}_{j'}$ (voisin de $\mathcal{TB}_j$), nous disons que v est un *faux nœud interne de degré* 3 : dans ce cas le degré de v est > 3 dans $\mathcal{B}$).

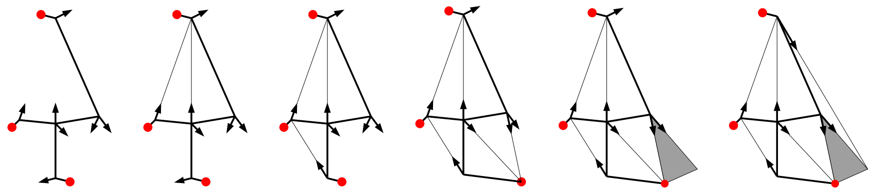


Fig. 7.7 – Ces images montrent une micro triangulation obtenue via notre clôture locale à partir d'un micro arbre $\mathcal{TB}$. Dans un cet exemple, nous partons d'un arbre ayant 5 et 10 feuilles : il possède 3 faux bourgeons (petits cercles rouges) et 3 faux nœuds internes (incluant la racine). La distribution des bourgeons est décrites par le mot binaire de longueur 10 et poids 3 : 0010100001.

Distribution des bourgeons et des faces. Il est à noter que les micro arbres peuvent présenter moins de 2 bourgeons par nœud : afin que ces micro arbres aient la même entropie de la classe à laquelle appartient $\mathcal{B}$, nous allons effectuer certaines modifications.

Les bourgeons originaux dans $\mathcal{B}$ sont dupliqués et distribués entre les micro arbres $\mathcal{TB}_j$ de telle sorte que chaque nœud ait deux feuilles. Le procédé est similaire à celui introduit là section 7.2 pour le cas des quadrangulations : nous omettons les légères modifications qui sont nécessaires pour tenir compte du fait que ici les arbres ne sont pas binaires, et chaque nœud a 2 bourgeons (dans le cas des arbres binaire les nœuds ayant au plus un bourgeon, à l'exception de nœud internes de degré 3). Soulignons seulement que les bourgeons dupliqués, appelés *faux bourgeons*, peuvent être assignés aux micro arbres partageant un même nœud v de manière à respecter l'ordre cyclique des arêtes incidentes à v.

Les faux nœuds internes de degré 3 sont toujours incidents à au moins un faux bourgeon, à l'exception du nœud racine du micro arbre (voir la figure 7.5). De plus, il est facile de vérifier que le nombre de faux nœuds internes, ainsi que le nombre de bourgeons dupliqués, est globalement $O(\frac{n}{\lg n})$.

Clôture d'un micro arbre

Notre clôture d'un micro arbre $\mathcal{TB}_j$ consiste à effectuer un parcours en profondeur le long des arêtes incidentes à la face externe (en sens trigonométrique et en partant de la racine) et à ajouter une face triangulaire (vraie ou fictive) : cela est réalisé par la fusion des bourgeons, suivant la distribution des faux bourgeons et faux nœuds internes de degré 3. Plus précisément nous effectuons la fusion des vrais bourgeons, selon les cas suivants :

- si un bourgeon s est précédé par 2 (vrais) nœuds internes, nous relions son nœud incident au deuxième dernier nœud qui le précède sur le bord de la face externe ;

- autrement, si s est précédé par un nœud v qui n'est pas interne, nous ajoutons une face triangulaire *fictive*, incidente à s et v (une face colorée dans la figure 7.7).

Les cartes planaires $\mathcal{TT}_j$ obtenues avec ce procédé (ayant des faces internes toutes de degré 3, et une seule face externe, simple et de taille arbitraire) sont appelées *micro triangulations*.

7.3.3 Vérification des hypothèses

Lemme 65. *Étant donnée une triangulation planaire $\mathcal{T}$ à $n+2$ sommets, notre stratégie produit une décomposition $\{\mathcal{TT}_1, \ldots, \mathcal{TT}_p\}$ en micro triangulations satisfaisant les hypothèses 1, 2, 3, 4 and 5, conduisant ainsi à une représentation succincte de $\mathcal{T}$ nécessitant asymptotiquement $3.24n + o(n)$ bits.*

Démonstration. La preuve est basée sur les mêmes arguments utilisés dans le lemme 64. L'hypothèse d'additivité est satisfaite, puisque les micro triangulations ne partagent que les nœuds racines (des micro arbres), qui sont globalement en nombre de $O(\frac{n}{\lg n})$.

Ici un objet $M_j = \mathcal{TT}_j$ est une micro triangulation, obtenue via notre clôture locale à partir d'un micro arbre $\mathcal{TB}_j$ ayant n_j nœuds, $2n_j$ bourgeons et w_j et faux bourgeons ; $\mathcal{TT}_j$ est induite avec un partitionnement de ses arêtes du bord en k_j côtés.

Ainsi une micro triangulation peut être représentée implicitement par une référence dans le catalogue exhaustif, celle-ci étant constituée de : un mot binaire de longueur $4n_j - 2$ et poids $n_j - 1$ (il existe au plus $2^{3.24n_j}$ de tels mots, voir section 3.3), un mot binaire de longueur $2n_j$ et poids w_j (pour décrire la distribution des faux bourgeons) et un mot binaire de longueur $6n_j - 2$ et poids k_j (il est facile à vérifier que toute micro triangulation a un bord contenant au plus $6n_j - 2$ arêtes). Encore une fois, la constante c peut être choisie de manière à ce que le catalogue $\mathcal{D}$ nécessite une mémoire auxiliaire de taille négligeable. □

7.4 Remarques finales

Nous avons présenté dans ce chapitre deux applications de notre schéma "micro/mini" conduisant à deux représentations succinctes de cartes planaires : en particulier ces applications ont tiré pleinement profit de la nouvelle définition du catalogue exhaustif des micro morceaux, ce dernier ne nécessitant plus de posséder la même entropie que la classe d'objets à représenter. La relaxation de cette dernière condition, ainsi que des nouvelles stratégies de décomposition d'une carte en mini/micro sous-cartes, nous ont permis d'améliorer les résultats existants connus et d'obtenir ainsi les

Représentation	triangulation	graphe 3-connexe
Jacobson (FOCS 89)	$64n$	$64n$
Munro et Raman (FOCS 97)	$2e + 8n$ ou $7m$	$2e + 8n$
Chuang et al. (ICALP 98)	$2e + n$ ou $3.5m$	$2e + 2n$
Chiang et al. (SODA 01)	$2e + 2n$ ou $4m$	$2e + 2n$
Castelli Aleardi et al. (WADS 05)	$2.175m$	-
Castelli Aleardi et al. (SOCG 06)	$1.62m$	$2e$

TAB. 7.1 – Ce tableau compare les performances de certaines représentations compactes de graphes : à l'exception du premier travail de Jacobson [74], toutes ces représentations permettent de répondre à certaines requêtes locales d'adjacence en temps $O(1)$. Les résultats concernent des graphes planaires simples (3-connexes et triangulés) et sont exprimés en termes du nombre e d'arêtes, m de faces et n de sommets. Les termes d'erreur d'ordre inférieur ne sont pas présentés, étant presque toujours $\Omega(\frac{n \lg \lg n}{\lg n})$.

premières représentations succinctes optimales pour les classes des maillages triangulaires et polygonaux de la sphère (le tableau 7.1 compare notre stratégie avec les représentations compactes, supportant des requêtes locales en temps constant, de graphes planaires existantes).

Possibles généralisations Les arguments utilisés sont assez généraux et suggèrent que notre schéma pourrait s'appliquer aussi à d'autres stratégies de codage, conduisant à des représentations succinctes ou compactes pour d'autres classes de graphes localement planaires (maillages surfaciques). Plus précisément, notre stratégie de décomposition du graphe (à l'aide d'arbres recouvrants) pourrait tirer profit de certaines similarités existantes entre "spanning tree based codings" [77, 125] et "region-growing approaches" [109, 120], comme déjà mis en évidence dans [70] : notamment notre stratégie de décomposition devrait s'étendre facilement au cas de codages de surfaces de genre supérieur.

Intérêt pratique et théorique Les résultats présentés dans ce chapitre sont d'intérêt plutôt théorique. D'une part les arguments mentionnés à la section 2.2.2 s'appliquent de manière générale au schéma mini/micro et donc à nos représentations des chapitres 5- 7 (ainsi qu'aux autres représentations compactes d'arbres, graphes planaires, Rank/Select, ...). D'autre part, les représentations succincte de ce chapitre ne sont valables et optimales que pour des cartes planaires. De plus, notre structure de données n'est que statique, n'admettant pas une version dynamique à cause des difficultés insurmontables pour l'implantation efficace de mises à jour locales : plus précisément, les correspondances bijectives entre cartes et arbres couvrants sont très sensibles aux modifications locales, comme les propriétés combinatoires sous-jacentes (ordres canoniques, réaliseurs).

Toutefois ces résultats révèlent d'un certain intérêt, au moins théorique, plus que ceux des chapitres précédents : en effet nous avons montré que mêmes des codages optimaux, se basant sur des fortes contraintes combinatoire, peuvent être source d'inspiration pour des représentations succinctes conçues pour permettre des opérations de navigation en temps constant, tout en gardant l'optimalité en terme de ressources mémoire.

7.5 Navigation locale dans la quadrangulation

7.5.1 Requêtes d'adjacence entre sommets et faces

Le lemme suivant fournit un outil pour tester l'adjacence entre cellules élémentaires (faces et sommets) dans le graphe. Ici nous supposons que les micro régions sont des cartes planaires "arbitraires" ayant des faces internes de degré borné (elles sont connexes avec un seul bord simple, et peuvent avoir éventuellement des arêtes doublement incidentes à la face externe).

Lemme 66. *Soit $\mathcal{M}$ un objet admettant une décomposition en micro (et mini) régions $[(M_1, \ldots, M_p); G; F]$ (suivant). Alors il est possible de répondre en temps $O(1)$ aux requêtes locales suivantes :*

- $Neighbor(f, e)$: retourne la face f' adjacente à f contenant l'arête e.

- $Adjacent(v, w)$: teste si deux sommets sont adjacents.

Démonstration. La preuve repose sur des arguments similaires à ceux déjà introduits à la section 5.5. □

Représentation unique pour les sommets. Le fait qu'il existe des sommets multiples ayant plusieurs représentation peut se résoudre à l'aide d'un étiquetage local et de la distinction des sommets en 3 catégories : les sommets internes à un micro morceau (ou partagés par au plus deux micro morceaux adjacents), les sommets partagés par au moins 3 micro morceaux, et les sommets appartenant à au moins 3 mini morceaux.

Il est possible d'associer à chaque sommet un représentant unique de manière canonique, exploitant la correspondance entre les sommets dans le graphe original (quadrangulation ou triangulation) et les nœuds de l'arbre bourgeonnant couvrant. Le lemme suivant permet de résoudre les ambiguïtés concernant les sommets multiples :

Lemme 67. *Seulement $o(n)$ bits supplémentaires suffisent pour répondre en temps $O(1)$ aux requêtes suivantes :*

– $Same(v, w)$: teste si deux représentations d'un sommet coïncident ;

– $Canonique(v)$: retourne le représentant canonique du sommet v (un triplet indiquant les micro/mini triangulations auxquelles il appartient).

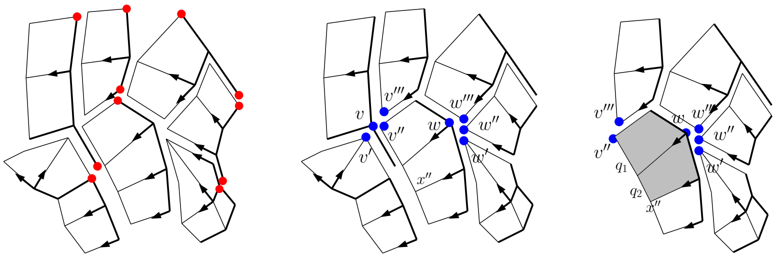


FIG. 7.8 – Dans ces images est montré le cas de deux sommets v et w, qui sont opposés et incidents à la même face quadrangulaire : v et w sont des sommets multiples (partagés par des micro morceaux différents), n'étant pas toutefois nœuds multiples (car ils n'apparaissent que dans un seul micro arbre). Les nœuds multiples, nœuds racines des micro arbres, sont représentés avec des petits cercles rouges.

7.5.2 Requêtes locales dans la quadrangulation

En ce qui concerne la navigation locale dans la quadrangulation, notre représentation supporte en temps $O(1)$ l'opération suivante (conduisant directement à la navigation dans la carte 3-connexe associée, comme déjà mentionné à la section 7.2.3) :

Lemme 68. *Dans la quadrangulation $\mathcal{Q}$ il est possible de répondre en temps $O(1)$ à la requête suivante :*

- Opposite(v, w) : teste si deux sommets v et w sont opposés et incidents à la même face quadrangulaire dans $\mathcal{Q}$.

Démonstration. La validité des arguments utilisés repose essentiellement sur les trois propriétés suivantes, satisfaites par les cartes planaires 3-connexes et leurs dissections correspondantes :

- L'arbre bourgeonnant recouvrant introduit dans [42] est un arbre binaire : le degré des nœuds et le nombre de bourgeons par nœud sont bornés.
- toute face quadrangulaire dans la décomposition de $\mathcal{Q}$ appartient à une micro quadrangulation exactement. De plus, chaque nœud v est incident à au plus deux faces fictives dans la même micro quadrangulation : dans le cas des nœuds ayant deux bourgeons, chaque face fictive est créée par la fusion de l'un de ses bourgeons ; pour les nœuds ayant un seul bourgeon (pouvant donner lieu à une face fictive), au plus une autre face fictive peut être créée, éventuellement incidente au nœud w descendant de v dans l'arbre (voir la figure 7.8).
- pour toute paire de sommet v et w il existe au plus une face quadrangulaire (si elle existe) contenant les deux sommets et telle que v et w sont opposés (la quadrangulation $\mathcal{Q}$ n'ayant pas de 4-cycle séparateur).

Soit q la face quadrangulaire (si elle existe) éventuellement incidente à v et w ; dénotons par $v', v'', \ldots$ (resp. $w', w'', \ldots$) leur copies, et par $\mathcal{TQ}_j$, $\mathcal{TQ}_{j'}$, $\mathcal{TQ}_{j''}$, ... les micro quadrangulations qui les contiennent.

Nous supposons, sans perte de généralité, que le nœud v précède w le long du bord de la face externe de l'arbre couvrant $\mathcal{B}$, selon l'ordre trigonométrique. Supposons d'abord que v et w ne soient pas des nœuds multiples, racines partagées de micro arbres (voir la figure 7.8).

Si v et w sont des sommets (éventuellement des "copies") appartenant à la même micro quadrangulation (lorsque $j = j'$), alors l'information stockée dans la représentation explicite contenue dans le catalogue A suffit pour répondre en temps $O(1)$. En effet, si v et w sont opposés alors ils doivent être à distance 2 dans $\mathcal{TQ}_j$, et tester cette condition nécessite le stockage de $o(n)$ bits supplémentaires.

Si v et w appartiennent à des micro quadrangulations différentes, qui ne sont pas nécessairement adjacentes, nous pouvons d'abord trouver leurs représentants canoniques à l'aide de la fonction $Canonique()$: si $Canonique(v)$ et $Canonique(w)$ appartiennent à la même quadrangulation, on se retrouve dans le cas précédent et il suffit d'effectuer des "table look-up".

Autrement, nous procédons de la manière suivante. S'il existait une face q incidente à v et w (et pour laquelle les deux sommets sont opposés), alors q devrait appartenir à la même micro quadrangulation contenant w et être incident à l'une des arêtes de l'arbre $\mathcal{TB}_{j'}$ (arête interne ou bourgeon) qui est aussi incidente à w (rappelons que v précède w sur la face externe). Puisque le degré des nœuds dans $\mathcal{B}$ est borné, il peut y avoir au plus 2 face quadrangulaires satisfaisant cette dernière condition : notons-les q_1 et q_2 (faces colorées dans la figure 7.8). Étant données q_1 et q_2, il est facile de retrouver les deux sommets v'' et x'' opposés de w, qui sont incidents à ces deux faces. Finalement il suffit de tester si les représentants $Canonique(v'')$ et $Canonique(x'')$ coïncident avec v (représentant canonique des copies de v). Pour conclure, il reste à observer que le cas où v (ou w) est un nœud multiple peut se traiter de manière similaire : il suffit de répéter les étapes ci-dessus un nombre constant de fois, puisque chaque nœud multiple appartient au plus à deux micro arbres différents. $\square$

Chapitre 8

Une piste d'implantation pratique

Bien que les structures asymptotiquement optimales des chapitres précédents soient loin d'être facilement implantables et réellement compétitives avec d'autres stratégies, le schéma adopté a été la source d'inspiration d'une version plus pratique visant à étudier le compromis entre temps de requête et ressources mémoire utilisées[1].

8.0.3 Implantations : travaux existants

S'il est vrai que plusieurs objets algorithmiques (tableaux, bit-vecteurs, dictionnaires, arbres, graphes et maillages,) ont été l'objet d'études visant à obtenir des représentations compactes, il y a eu très peu de tentatives d'en faire des versions utilisables dans la pratique. Une première explication (justifiant la pénurie de telles tentatives) est même donnée dans le travail fondateur [94] introduisant les premières représentations succinctes de mots de parenthèses et arbres. Sans prendre en compte la perte d'efficacité dans l'accès à l'information (en terme de temps d'exécution), il reste la question concernant l'optimalité "asymptotique" (en terme d'espace mémoire).

Une évaluation quantitative précise de toutes les informations supplémentaires, dont le coût est caché dans le terme $o(n)$, porte à penser que le bénéfice provenant d'une représentation succincte (par rapport à une structure classique) pourrait se manifester seulement pour des objets de très grande taille.

Déjà dans le cas des arbres, pour qu'une structure succincte[2] puisse réellement être comparée à une représentation classique (basée sur des pointeurs), il faudrait considérer des objets ayant environ 10 millions de sommets

[1] Les résultats présentés dans cette section sont publiés à CCCG 2006 [25].

[2] En réalité il s'agirait d'une version simplifiée, ne permettant qu'un ensemble restreint d'opérations de navigation locale, de la structure relative au théorème 31.

(et dans ce cas le gain correspondrait à une réduction d'un facteur 3 de la taille mémoire).

Rank/Select et arbres Afin que notre présentation soit complète, il nous reste à mentionner deux travaux plus récents ayant considéré le cas des bit-vecteurs avec Rank/Select [47] et des mots de parenthèses équilibrés [46]. En particulier d'intéressants résultats expérimentaux ont été obtenu avec l'implantation de la représentation succincte (améliorée) des mots de parenthèses, que nous avons mentionnée à la section 3.5.2.

Structures de données géométriques Très peu de travaux ont proposé des solutions pratiques (inspirées des représentations compactes) pour des structures de données plus complexes, telles que les graphes et les maillages.

A part les tentatives d'éliminer les redondances des structures explicites (*star-vertices*, voir section 2.4), à notre connaissance les seuls travaux ayant eu du succès sont ceux se basant sur les propriétés graphes ayant des petits séparateurs.

La structure compacte de Blandford et Blelloch décrite à la section 3.7 a donné lieu à l'implantation d'une structure de données [12] pouvant représenter des maillages triangulaires : les résultats expérimentaux obtenus avec cette implantation montrent qu'un maillage triangulaire en 2D nécessite en pratique (sur les exemples considérés) 5 octets par triangle. Cette représentation admet deux interfaces, l'une basée sur les arêtes et l'autre sur les sommets, mettant à disposition de l'utilisateur des opérations standard de navigation entre triangles et des opérations de mise à jour du maillage : il est à observer que ces opérations peuvent s'effectuer en temps $O(1)$ dans le cas de maillages ayant des sommets de degré borné.

S'il est vrai que les performances montrées rendent cette approche intéressante dans la pratique, d'un point de vue théorique il semble impossible de fournir des garanties précises sur la complexité (en espace et en temps) de la représentation. Plus précisément, la stratégie d'étiquetage des sommets adoptée visant à tirer profit des propriétés des graphes ayant des petits séparateurs, présente quelque points faibles : d'une part les bornes sur la taille mémoire de la structure ne sont valables que dans le cas statique ; d'autre part la nécessité de calculer et modifier les étiquettes des sommets rend difficile une évaluation explicite de la complexité concernant les opérations de mise à jour du graphe.

8.0.4 Termes d'erreurs sous-linéaires

Comme déjà mentionné dans cette thèse (voir section 2.2.2), il existe au moins deux arguments qui rendent les structures compactes ou succinctes basée sur le schéma algorithmique mini/micro de caractère plutôt théorique : nous allons les reprendre brièvement ici avec une approche plus quantitative.

D'une part, il est désormais clair pour le lecteur que l'efficacité et l'optimalité des structures succinctes réside de manière importante dans la capacité de gérer (à tout niveau), des références et des pointeurs locaux de taille arbitraire $O(\lg n)$. Bien que le modèle de calcul adopté soit censé supporter en temps $O(1)$ l'accès en lecture/écriture sur de telles références, l'implantation de tableaux et d'autres structures gérant des cases de taille variable, ainsi que la fréquente utilisation d'opérations bit-à-bit, peut affecter de manière significative les performances pratiques de nos algorithmes.

D'autre part, il reste la question concernant les termes d'erreurs de la forme $O\left(n\frac{\log\log n}{\log n}\right)$: bien que asymptotiquement négligeables en théorie pour des objets de taille assez grande, la pratique nous montre que ces termes d'erreur touchent de manière importante les ressources mémoires utilisées par les représentations succinctes.

8.1 Solution proposée : esquisse

Bien que les représentations succinctes des chapitres précédents puissent difficilement donner lieu à une implantation vraiment efficace et optimale en pratique (pour les raisons mentionnées ci-dessus), notre conviction est que le schéma que nous avons proposé dans cette thèse peut servir d'inspiration pour des versions implantables et rentables (en terme de performances et de ressources mémoire) dans la pratique. En particulier nous allons décrire une version simplifiée et non asymptotiquement optimale, dont la conception est basée sur les remarques suivantes :

– bien que de taille non négligeable en pratique, le graphe d'incidence des micro triangulations est toutefois plus petit que le graphe d'incidence de l'ensemble de triangles original.
– en conséquence, si nous limitons notre structure multi-niveaux à un seul niveau (celui des micro triangulations), il devrait être possible de réduire les ressources mémoires, par rapport aux structures explicites existantes : il est à rappeler que déjà avec un seul niveau, le lemme 37 garantit la compacité de la structure.
– il est facile de se rendre compte que $\frac{\log n}{12}$ est en général très petit, même pour des triangulations de grosse taille : par exemple $\frac{\log 70 milliards}{12} = 4$. En pratique la taille n pourrait être considéré comme une constante plutôt que comme un paramètre. Ceci a motivé une première étude détaillée de catalogues de micro triangulations de petite taille.
– en conséquence, le fait de construire des catalogues "non optimaux" en relâchant les contraintes de taille et en considérant des micro triangulations "combinatoirement simples", permet d'obtenir une version réellement implantable de notre structure.
– pour conclure, la dernière simplification concerne l'organisation mémoire de notre structure et a été motivée par les difficultés pratiques évidentes

à mettre un place une structure succincte suivant de manière rigoureuse les formulations des chapitres 5 et 6 : surtout il serait souhaitable de ne pas avoir recours aux différents tableaux dynamiques décrits à la section 2.6.3. La solution proposée consiste à éviter l'utilisation de pointeurs de taille variable sur $O(\lg\lg n)$ bits, et donc à préférer des vrais pointeurs sur w bits ou des index sur $O(\lg n)$: si l'optimalité asymptotique théorique en est affectée, il est néanmoins vrais que ce choix est presque indispensable, au moins dans cette première étude, pour garantir la faisabilité et l'efficacité de notre structure de donnée.

Dans la suite de cette section nous allons présenter l'étude de 3 catalogues, ainsi que l'étude de leur performances dans la représentation d'une triangulation. Nous terminerons en commentant son implantation et les résultats expérimentaux obtenus.

8.1.1 Définitions

Un *Catalogue* $\mathcal{C}$ est une collection $\{t_1, \ldots, t_p\}$ de triangulation planaires ayant un bord simple (de taille arbitraire), qui sont appelées *patchs* (les t_i correspondent aux micro triangulations du chapitre 5).

Un catalogue est *stable* par rapport à une opération de mise à jour donnée[3] si l'application de l'opération à un élément quelconque du catalogue donne comme résultat soit un autre élément du catalogue, soit l'union d'un nombre borné d'éléments du catalogue.

L'intérêt de définir un tel catalogue réside dans la possibilité de représenter une triangulation $\mathcal{T}$ comme union disjointe de patchs dans $\mathcal{C}$: $\mathcal{T} = \bigcup_{j\in J} t_{j_i}$ $(1 \leq j_i \leq p)$. La figure 8.1 montre des exemples de catalogues stables.

Un catalogue $\mathcal{C}$ est *minimal* si aucun patch ne peut s'obtenir comme union disjointe d'autres patchs dans $\mathcal{C}$.

Calcul d'un catalogue Le problème de déterminer, étant donné un paramètre entier k, un catalogue minimal ayant des patchs de taille au moins k, se pose de manière naturelle. Une solution possible consiste à procéder de la façon suivante :

- le catalogue $\mathcal{C}$ devrait contenir toutes les triangulations (combinatoirement différentes) ayant k triangles ;

- $\mathcal{C}$ devrait aussi inclure toutes les combinaisons ayant entre $k+1$ et $2k-1$ triangles, puisque il n'est pas possible d'obtenir une triangulation de cette taille avec des patchs de k triangles[4] ;

[3] Dans cette section, suivant le choix adopté au chapitre 6, nous considérons les opérations d'insertions/suppression de sommets de degré 3 et de flip d'arête

[4] Rappelons que si on suivait ici la même stratégie adoptée aux chapitres précédents, il faudrait aussi inclure toutes les configurations ayant au plus $3k$ triangles : ceci permet donc d'économiser sur les patchs de taille entre $2k$ et $3k$.

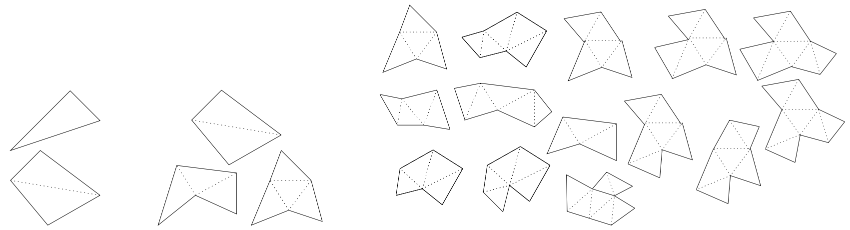


FIG. 8.1 – Le trois catalogues introduits dans la section 8.1.2 sont ici montrés : $\mathcal{C}_1$ est stable mais non minimal, contenant des triangles et des quadrangles. $\mathcal{C}_2$ et $\mathcal{C}_3$ sont minimaux, et leurs patchs contiennent au moins 2 et 3 triangles respectivement.

- enfin, il ne reste qu'à tester toutes les configurations obtenues en appliquant une mise à jour aux patchs du catalogue : lorsque une configuration ne peut pas s'obtenir comme union d'autres éléments de $\mathcal{C}$, celle-ci doit être ajoutée au catalogue.

8.1.2 Catalogues simples

Nous allons maintenant présenter l'étude de trois catalogues, pour lesquels il sera possible de fournir une analyse concernant les ressources mémoires utilisées : encore une fois la relation d'Euler, ainsi que certaines considérations élémentaires de nature combinatoire, nous conduirons à des bornes supérieures garanties.

Triangles-quadrangles. Le premier catalogue $\mathcal{C}_1$ que nous allons considérer n'est composé que de deux patchs : les triangles et les quadrangles. Il est immédiat de vérifier que $\mathcal{C}_1$ est stable pour nos opérations de mise à jour, puisque contient le triangle.

Le schéma adopté pour représenter la triangulation, s'inspire de la représentation "face-oriented" [15] décrite à la section 2.4.

Une fois la triangulation décomposée en deux ensembles de triangles et quadrangles, il suffit de décrire la connectivité en terme de relations de voisinage entre patchs. Un triangle est représenté, de manière usuelle avec 6 références vers ses voisins et sommets incidents ; un quadrangle n'a besoin que de 4 références pour les patchs voisins et 4 références pour ses sommets incidents : ce qui conduit à un gain de 2 références pour chaque triangle converti en quadrangle.

Triangulations et quadrangulations. Une question intéressante qui se pose naturellement, concerne la possibilité de convertir une triangulation en une quadrangulation (notre structure pourrait tirer profit d'une telle situation essentiellement dans le cas statique, où l'absence de mise à jour n'affecterait pas la décomposition en quadrangles).

Plusieurs travaux ont porté sur le problème de convertir une triangulation en une quadrangulation [58, 107], ou de créer une quadrangulation à partir d'un ensemble de points en entrée [121]. Il est à observer qu'il n'est pas évident, en général, de garantir qu'une triangulation arbitraire puisse se convertir en une quadrangulation : c'est même impossible dans plusieurs situations, par exemple lorsque la taille du bord est impaire (pour plus de détails voir [21]).

S'il s'agit d'une triangulation de la sphère alors le théorème de Petersen [98] garantit que tous les triangles peuvent se rassembler en faces quadrangulaires (puisque tout graphe connexe 3-régulier sans isthme admet un couplage parfait).

Représentation dynamique. Vu l'orientation pratique de ce travail, il est toutefois plus intéressant de considérer le cas dynamique où la triangulation peut se construire de manière incrémentale à l'aide des opérations de mise à jour disponibles.

Dans le cas de maillages triangulaires sans bord de la sphère, la relation d'Euler permet d'obtenir le lemme suivant :

Lemme 69. *Soit $\mathcal{T}$ une triangulation planaire à n sommets. Alors il existe une représentation "face-based" n'utilisant que des triangles et des quadrangles qui nécessite entre $9n$ et $10.6n$ références.*

Démonstration. Il est évident que le gain atteignable est proportionnel au nombre de quadrangles construits. Le maximum est atteint lorsque toutes les patchs sont des quadrangles, ce qui correspond à $9n$ références (8 pour chaque quadrangle, une pour chaque sommet).

S'il est vrai qu'en général un certain nombre de triangles restent non couplés, nous pouvons supposer que la décomposition de $\mathcal{T}$ ne contienne aucune paire de triangles adjacents : si c'était le cas, on pourrait les regrouper pour former un quadrangle.

En notant t et q respectivement le nombre de triangles et de quadrangles de la décomposition, nous avons que le nombre total de faces est

$$2n = t + 2q \tag{8.1}$$

Le nombre d'arêtes dans une triangulation est $3n - 6$, qui peut s'écrire aussi $3n = e_{ext} + e_{int}$, où e_{int} représente le nombre d'arêtes internes à un quadrangles (diagonales) et e_{ext} est le nombre d'arêtes partagées par quadrilatères et triangles, et entre quadrilatères.

La première quantité est au moins $3t$, car les arêtes d'un triangle sont toujours partagées avec un quadrilatère (il n'existe pas deux triangles adjacents). Le deuxième terme devrait être positif, ce qui conduit à :

$$3n - 3t - q > 0 \tag{8.2}$$

En utilisant les relations 8.1 et 8.2 il est immédiat d'obtenir :

$$q > \frac{3}{5}n \tag{8.3}$$

Ce qui implique qu'il existe au plus $\frac{4}{5}n$ triangles dans la décomposition. La triangulation peut alors être représentée avec au plus

$$6t + 8q + n \leq [6\frac{4}{5}n + 8\frac{1}{2}(2n - \frac{4}{5}n)] + n = \frac{53}{5}n = 10.6n$$

références (6 et 8 pour les triangles et les quadrangles, une pour les sommets), au lieu des $13n$ de la représentation classique, produisant un gain de 19%.

□

Catalogue avec au moins 2 triangles par patch Nous allons considérer les catalogue $\mathcal{C}_2$, contenant 5 patchs, chacun de taille au moins 2 : $\mathcal{C}_2$ est stable par rapport aux modifications locales (voir la figure 8.1).

Comme auparavant, les patchs sont codées avec une représentation "face-oriented" : les pentagones utilisent 5 références pour les sommets et 5 pour les patchs adjacents (les hexagones en stockent 6, soit pour les sommets soit pour les voisins). De cette manière, le gain obtenu correspond à 2 références pour chaque triangle converti en quadrangle, $\frac{8}{3}$ références pour chaque triangle converti en pentagone, et 3 références dans le cas des hexagones. Des arguments similaires à ceux utilisés dans la preuve du lemme précédent conduisent au résultat suivant :

Lemme 70. *Une représentation d'une triangulation $\mathcal{T}$ à n sommets basée sur le catalogue $\mathcal{C}_2$ utilise au plus $8.37n$ références.*

Démonstration. D'abord nous pouvons supposer qu'il n'existe pas de quadrilatères adjacents : cela est facilement faisable puisque toute séquence de quadrilatères peut être décomposée en hexagones et pentagones pour que configuration formée de 2 ou plusieurs quadrilatères soit évitée.

Le pire des cas est réalisé lorsque il n'existe pas d'hexagones dans la décomposition : il est facile de voir qu'il peut y avoir au plus $\frac{5}{11}n$ quadrilatères. Si q et r désignent respectivement le nombre de quadrilatères et pentagones, le nombre total de faces de $\mathcal{T}$ est :

$$2n = 2q + 3r$$

D'ailleurs, il doit y avoir au moins $\frac{4}{11}n$ pentagones. En rappelant qu'il n'existe pas de quadrangles adjacents et que le nombre total d'arêtes est

$3n = e_{int} + e_{ext} \geq (q + 2r) + 4q$ (où e_{int} est le nombre d'arêtes internes aux patchs, e_{ext} est le nombre d'arêtes partagées par patchs différents), il est immédiat d'obtenir $r \geq \frac{4}{11}$ (et par conséquent $q = n - \frac{3}{2}r \leq \frac{5}{11}n$).

Ainsi, dans le cas le pire, le nombre de références nécessaires pour notre représentation "face-oriented" (8 et 10 pour les quadrangles et les pentagones, une pour les sommets) est au plus

$$8q + 10r + n \leq [8 \cdot \frac{5}{11}n + 10 \cdot \frac{1}{3}(2n - 2\frac{5}{11}n)] + n = \frac{91}{11}n = 8.27n$$

ce qui correspond à un gain de 36% par rapport à la représentation classique basée sur les triangles.

□

Catalogue minimal avec au moins 3 triangles par patch La figure 8.1 montre le dernier catalogue $\mathcal{C}_3$, qui est stable et minimal : $\mathcal{C}_3$ contient des triangulations planaires ayant entre 3 et 7 triangles, dont le bord a une taille entre 6 and 9.

Les polygones constituant les patchs de taille 7, 8 et 9 sont représentés avec 14, 20 et 24 références respectivement. Ce qui conduit à un gain équivalent à $\frac{16}{5}$, $\frac{10}{3}$ et $\frac{24}{7}$ pour tout triangle converti en une de ces triangulations. Le cas le pire correspond à une décomposition ne contenant que des pentagones. Des arguments élémentaires de comptage, similaires à ceux utilisés dans les lemmes précédents, permettent d'évaluer à 41% le gain que l'on peut obtenir avec une représentation basée sur ce dernier catalogue, comme exprimé par le lemme suivant :

Lemme 71. *Une représentation d'une triangulation $\mathcal{T}$ à n sommets utilisant le catalogue $\mathcal{C}_3$ nécessite au plus $\frac{23}{3}n = 7.67n$ références.*

8.2 Implantation et résultats

Nous avons conçu et implanté une version préliminaire de notre structure basée sur le catalogue $\mathcal{C}_1$. Bien que cette représentation soit limitée au cas des triangles et quadrangles, il est utile de mentionner quelques remarques générales qui restent valables pour d'autres catalogues plus riches. Du point de vue de la réalisation pratique, notre représentation d'une triangulation est constituée de plusieurs structures (tableaux) contenant différents types d'information. D'abord il y a un tableau contenant les informations géométriques associées aux sommets. Pour chaque type de patch du catalogue il faut maintenir un tableau contenant un élément pour chaque patch dans la décomposition (ainsi dans le cas du catalogue $\mathcal{C}_1$ trois tableaux suffisent). Le fait d'utiliser des tableaux différents pour des patchs différents implique que les références utilisées dans la description de la triangulations ne sont pas

Représentation adoptée	Nombre triangles	Nombre quadrangles	Mémoire utilisée
triangles/quads	$8 \cdot 10^6$	$16 \cdot 10^6$	1.3Go
triangles	$40 \cdot 10^6$	-	1.7Go

TAB. 8.1 – Ce tableau compare les performances (en terme de ressources mémoire utilisées) de la représentations classique basée sur les triangles et de notre implantation basée sur le catalogue $\mathcal{C}_1$: ces résultats (exprimés en Gigaoctets) correspondent au calcul incrémental de la triangulation de Delaunay d'un ensemble de $20 \cdot 10^6$ de points dans le plan.

équivalentes : il nous reste donc à ajouter une information supplémentaire aux références pour indiquer le type de patch concerné (dans le cas de $\mathcal{C}_1$, constitué de deux éléments, un seul bit suffit). De plus, il est possible de stocker d'autres informations supplémentaires codant l'index de la face à laquelle on fait référence dans un patch donné (d'un point de vue général, cela équivaut au coloriage des arêtes du bord des micro triangulations, dans les représentations des chapitres précédents). Cette information rend beaucoup plus efficace la localisation d'un voisin d'un triangle donné, évitant de tester tous les triangles contenus dans un patch (nous avons adopté cette stratégie dans notre implantation de $\mathcal{C}_1$: un seul bit suffit pour localiser l'un des deux triangles d'un quad).

Performances. Nous avons implanté et testé la version dynamique de notre représentation dans le cas d'un maillage planaire. Les résultats expérimentaux montrés dans le tableau 8.1 ont été obtenus en calculant la triangulation de Delaunay d'un ensemble aléatoire de points (avec distribution uniforme), en adoptant l'algorithme incrémental fourni par la bibliothèque CGAL.Ces résultats confirment notre analyse des ressources mémoire et le gain obtenu d'environ 20%. En ce qui concerne les performances de calcul, nous devons mentionner une perte d'efficacité par rapport aux représentations classiques basées sur les triangles. D'une part cela est naturel si on tient compte du compromis ressources mémoires/temps de requête. D'autres part, nous soulignons que dans la conception de notre structure nous avons voulu garder une pleine compatibilité avec l'interface et les structures adoptés dans CGAL. Ainsi l'utilisateur a plein accès aux fonctionnalités de CGAL : par exemple la manipulation des faces ne nécessite aucune connaissance et interaction avec la représentation interne adoptée (notre représentation triangle/quad peut remplacer directement la représentation d'une triangulation basée sur les triangles adoptée dans CGAL) Il est évident que ce choix (de garantir une intégration avec CGAL) affecte de manière non négligeable le temps de calcul de la triangulation.

Chapitre 9

Conclusion et perspectives

L'objectif de nos travaux a été de présenter une étude générale des représentations compactes d'objets géométriques. Notre contribution principale a été de proposer un schéma algorithmique pour la conception de représentations compactes et succinctes pour différentes classes de maillages surfaciques. Surtout nous avons montré que la propriété de *planarité locale* caractérisant les objets dont nous nous sommes intéressés (maillages surfaciques, triangulés et polygonaux) joue un rôle crucial dans la description de telles structures. La structure hiérarchique à plusieurs niveaux à la base de notre schéma est assez générale et laisse espérer de pouvoir obtenir des représentations compactes pour d'autres classes de graphes admettant une stratégie de décomposition similaire à celles des chapitres 5 ou 7. Et bien que nos travaux révèlent d'un intérêt plutôt théorique, nous avons montré la possibilité de tirer profit de nos descriptions pour obtenir des structures de données réellement implantables et capables de manipuler des maillages de grosses dimensions. En ce qui concerne les prolongations possibles des travaux proposés dans cette thèse, il reste plusieurs questions intéressantes à explorer : certaines plus théoriques (l'étude de représentations compactes pour les maillages surfaciques de genre quelconque et les maillages volumiques en $\mathbb{R}^3$) et d'autres liées aux possibles applications de nos représentations (amélioration de la structure proposée au chapitre 8 et conception de stratégies efficaces pour le traitement et la compression de la géométrie). Dans le reste de cette section nous allons mieux préciser les pistes de recherche que nous envisageons d'explorer dans le futur.

Genre supérieur L'une des premières extensions possibles de nos travaux devrait concerner les maillages surfaciques de genre quelconque. En effet nous avons déjà remarqué comment notre schéma s'applique également bien en genre supérieur (voir chapitre 5) : le fait que les hypothèses du chapitre 4 restent valides dépend principalement de la complexité des graphes G

et F, qui sont encore creux, puisque localement planaires. Il est néanmoins à souligner que l'obtention de représentations succinctes (optimales) n'est plus chose immédiate en genre supérieur : comme notre stratégie est basée sur une décomposition à l'aide d'arbres couvrants, l'optimalité n'est plus atteignable lorsque des codages optimaux ne sont pas connus. En général, il est à rappeler que la plupart des codages et représentations succinctes concernent le cas de graphes planaires : ceci dépend du fait que de nombreux algorithmes tirent profit de fortes propriétés combinatoires (*réaliseurs* et *ordres canoniques*) caractérisant de manière fine la notion de planarité des graphes. Il serait donc très intéressant de vérifier si ces propriétés combinatoires peuvent s'étendre au cas de surface de genre supérieur et de topologie arbitraire (une première étude a été conduite dans [19] pour le cas de surfaces toriques). Les efforts dans cette direction seraient justifiés par le fait qu'il existe des similarités entre les stratégies de compression de maillages (pouvant pour la plupart se généraliser en genre supérieur [109, 120, 54]) et les algorithmes liés aux propriétés combinatoires mentionnées ci-dessus (pour l'instant conçus pour le cas planaire [76, 99, 42, 31]). Notamment ces similarités se manifestent dans l'approche commune consistant à effectuer un parcours incrémental d'un graphe réalisé par conquête de faces ou sommets, comme esquissé à la section 3.1.1 ("growing region approach") : de telles stratégies sont à la base des algorithmes de codage de graphes (par exemple [109, 120, 54]), ainsi que des algorithmes conçus pour calculer les *réaliseurs* et *ordres canoniques* des triangulations et graphes planaires (pour une présentation détaillée de ces sujets voir [20, 76]).

Maillages volumiques Une autre généralisation pourrait concerner le cas de maillages volumiques (3-complexes plongés dans $\mathbb{R}^3$).

En effet le schéma de décomposition qui est à la base de nos représentations pourrait s'appliquer facilement aux cas des maillages tétraédriques. S'il est vrai que notre stratégie de décomposition en mini/micro morceaux à l'aide d'arbres couvrants (cfr. chapitre 5) pourrait s'étendre aux cas 3D, les hypothèses du chapitre 4 ne sont plus forcément vérifiées. En particulier, vient à manquer la notion de planarité locale permettant de décrire les relations de voisinages entre micro morceaux à l'aide d'un graphe globalement creux. La version planaire de la formule d'Euler établit une dépendance linéaire entre le nombre de sommets, arêtes et faces d'un maillage surfacique. Alors que dans le cas de triangulations de dimension 3 on sait que la dépendance entre le nombre de tétraèdres, de sommets, d'arêtes et de faces peut s'exprimer par une relation quadratique[1] : ce qui ne permet pas, dans le cas général, d'obtenir une représentation compacte suivant la stratégie du théorème 38.

[1]On sait qu'il existe des 3-triangulations ayant n sommets et dont le nombre de tétraèdres est $t = \Theta(n^2)$ (voir par exemple [16]).

Le cas de maillages volumiques 3D, a été abordé par un certain nombre de travaux dans le domaine de la simple compression ([53], [64] et [118]) ainsi que de la conception de représentations compactes [12]. Premièrement, nous devons mentionner que dans le cadre des maillages volumiques, même le problème de l'énumération et celui de l'obtention d'un codage de taille linéaire restent ouverts. Pour aucune des stratégies proposées on ne peut fournir des bornes supérieures garanties, concernant la longueur du code produit : de plus, il existe des arguments théoriques et des constatations expérimentales indiquant que la taille des codes et représentations ne croit pas linéairement en la taille du maillage.

En ce qui concerne les méthodes de compression, les travaux existants ([53, 64, 118]) constituent, pour la plupart, des extensions de méthodes conçues pour la compression ou représentation de maillages surfaciques ([109, 120, 54]), se basant encore une fois sur un parcours incrémental du graphe, à l'aide d'*opérations d'extention* et *arbres couvrants*.

Par exemple, l'algorithme *Grow & fold* [118] est une extension de *Edgebreaker* : il effectue un parcours en profondeur d'une triangulation 3D (dont la surface du bord est une variété) en calculant un arbre de tétraèdres. À partir de ce dernier et d'une information supplémentaire associée aux faces du bord, il est possible de reconstituer les relations d'incidence du maillage initial. Cela s'effectue à l'aide de deux opérations, *fold* et *glue*, qui décrivent la manière dont les faces du bord de l'arbre de tétraèdres doivent se recoller. Il est à remarquer qu'il rend nécessaire d'utiliser jusqu'à $O(\lg n)$ bits pour coder de telles opérations, lorsque le recollement ne peut pas se décrire de manière locale : par exemple lorsqu'il faut donner explicitement les index de deux faces, non adjacentes sur le bord, qui doivent se recoller en une seule. Comme mis en évidence dans [118], il n'est pas possible en général d'éviter ces événements, qui peuvent manifester avec une fréquence arbitraire : comme pour les codes *split* dans le cas surfacique, cela empêche ainsi de pouvoir établir une borne linéaire sur la longueur du code. Des arguments similaires valent aussi pour d'autres algorithmes de compression [53, 64]. Intuitivement cela pourrait dépendre du fait que, dans le cas volumique 3D, il n'est pas toujours possible de construire une tétraédrisation (dont le bord est une surface manifold de genre 0) en recollant un à un des tétraèdres de manière à ce que le bord reste à chaque étape manifold (on ne peut pas éviter les auto-intersections du bord). Ces remarques sembleraient ainsi dépendre de certaines propriétés concernant les *ordres d'épluchage* ("*shelling orders*") pour les 3-complexes simpliciaux[2] ([130] fournit une excellente présentation de ces sujets).

En ce qui concerne les représentations compactes de maillages volumiques le seul travail existant, à notre connaissance, est celui de Blandford

[2]Contrairement à ce qui arrive dans le cas surfacique, il existe des tétraédisations qui ne sont pas épluchables.

et al. [12] basé sur les propriétés des séparateurs des graphes : s'il est vrai que leur stratégie s'adapte bien au cas de complexes simpliciaux de dimension 3, ce n'est que pour des classes de maillages particulières que l'on peut fournir des garanties précises. De plus, il semblerait impossible d'obtenir un codage de taille linéaire avec cette méthode dans le cas général. En effet il existe des complexes simpliciaux de dimension 3 en $\mathbb{R}^3$ (ayant $O(n \lg n)$ simplexes) qui n'admettent que des séparateurs de taille au moins $\Omega(n)$ (voir [92]) : alors que la validité du lemme 36 caractérisant cette approche repose sur l'hypothèse de pouvoir calculer des séparateurs de taille $O(n^c)$ (avec $c < 1$).

Toutes ces remarques justifient à notre avis de nouvelles explorations pour le cas des maillages volumiques. Notamment il serait important d'établir des résultats d'énumération et de codage pour des sous-classes de 3-complexes simpliciaux particulièrement intéressantes dans le domaine de la géométrie algorithmique : dans ce sens, nous suggérons que la classe des *3-complexes épluchables* (*"shellable"*) pourraient constituer un très bon candidat pour des recherches futures.

Une implantation compétitive La mise un place d'une première version implantable, non asymptotiquement optimale, de nos représentations (voir chapitre 8), nous laisse espérer que notre schéma pourra être encore amélioré et conduire à des structures performantes et compétitives dans la pratique. Parmi les modifications possibles, la plus simple consiste à adopter des références sur $\lg n$ bits au lieu que de vrais pointeurs sur 32 bits (comme dans notre implantation du chapitre 8). Mais l'idée de base d'une amélioration possible consisterait à utiliser un autre niveau dans la décomposition (mimant la stratégie du chapitre 5). Une fois décomposée la triangulation initiale, on pourrait regrouper les patch pour former des mini régions, où il serait possible d'utiliser des pointeurs locaux, de taille fixe, moins coûteux. Essentiellement cela consiste à adapter la structure du chapitre 5 en relâchant les nombreuses contraintes imposées sur la taille des micro/mini régions et la longueur des références. D'une part on perdrait l'optimalité asymptotique garantie en théorie (l'utilisation de pointeurs de taille variable est cruciale dans l'analyse des ressources mémoire). Mais d'autre part on gagnerait en termes de performances de l'implantation, surtout en ce qui concerne la manipulation de références de taille fixe et non plus variable Et finalement notre méthode de représentation *par patch* peut s'adapter facilement au cas volumique $3D$: comme dans le cas planaire, il suffit de définir des catalogues simples constitués de petits polyèdres (équivalents des patch) obtenus par le recollement de 2 ou plusieurs tétraèdres. Bien que la relation d'Euler ne permette plus d'obtenir de bornes explicites (comme dans le cas planaires), cette stratégie pourrait conduire à une réduction du nombre des références nécessaires pour une représentation explicite.

Compression de la géométrie Tout au long de cette thèse nous avons considéré le problème du codage de l'information combinatoire : cela était motivé par le fait que ce type d'information constitue une partie très importante de l'information globale contenue dans un objet géométrique. Une fois que nous avons montré comment il est possible de "compresser" et de réduire le coût de la connectivité d'un maillages (on est passé de plusieurs centaines de bits par sommet à 2.17 ou 1.62 bits par sommet dans le cas triangulaire), il est temps de se concentrer sur l'information géométrique provenant des coordonnées des points dans l'espace : les quelques dizaines de bits par sommet nécessaires pour coder la géométrie ne sont plus négligeables.

Dans ce cadre on part de l'observation que les coordonnées des sommets d'un maillage ne sont pas complètement indépendantes entre elles, au moins dans la pratique : par exemple lorsqu'on considère des maillages approchant une surface lisse, on constate que les coordonnées d'un sommet sont assez proches de la moyenne des coordonnées de ses sommets voisins (sommets adjacents dans le maillage). Cette remarque a déjà conduit à plusieurs stratégies de compression visant à mieux prédire et coder l'information géométrique à partir de l'information combinatoire d'un maillage. Nous disposons déjà d'un ingrédient crucial, l'information de connectivité, à laquelle nos structures permettent d'accéder de manière efficace, sans la nécessité de décoder globalement la représentation de l'objet. L'idée de base serait de tirer profit de notre représentation hiérarchique d'un maillage afin d'adopter un repère local et une stratégie de *prédiction* pour coder la position de points avec des coordonnées locales. Plus précisément, on pourrait stocker explicitement la géométrie d'un ensemble limité de points du maillage (contenant par exemple les sommets multiples, dont l'information géométrique est partagée par un nombre considérable de micro régions) et utiliser des coordonnées locales pour exprimer la position des autres sommets : ces coordonnées seraient relatives à des repères locaux, chacun associé à une micro région. Il serait aussi possible de *prédire* la géométrie d'un sommet en utilisant la géométrie des sommets dans un voisinage et de se limiter à stocker un vecteur représentant l'*erreur de prédiction*. Cette prédiction peut se faire à l'aide d'une combinaison linéaire des coordonnées d'un nombre restreint de sommets voisins dans le maillage : par exemple de manière similaire à ce qu'on fait lorsque on applique la *règle du parallélogramme*, dans les algorithmes standards de compression [109, 120].

Bibliographie

[1] O. Aichholzer, T. Hackl, H. Krasser, C. Huemer, F. Hurtado, and B. Vogtenhuber. On the number of plane graphs. In *Proc. 17th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA)*, pages 504–513, 2006.

[2] Oswin Aichholzer. The path of a triangulation. In *Proc. 15th Ann. ACM Symp. on Comput. Geom.*, pages 14–23, 1999.

[3] P. Alliez and M. Desbrun. Progressive compression for lossless transmission of triangle meshes. In *SIGGRAPH*, pages 195–202, 2001.

[4] P. Alliez and M. Desbrun. Valence-driven connectivity encoding of 3d meshes. In *Proceedings of Eurographics*, pages 480–489, 2001.

[5] P. Alliez and C. Gotsman. Recent advances in compression of 3d meshes. In N.A. Dodgson, M.S. Floater, and M.A. Sabin, editors, *Advances in Multiresolution for Geometric Modelling*, pages 3–26. Springer-Verlag, 2005.

[6] C. Arndt. *Information measures.* Springer, first edition, 2001.

[7] Robert Ash. *Information Theory.* Dover Publications, 1990.

[8] B. G. Baumgart. Winged-edge polyhedron representation. Technical report, Stanford University, 1972. Standford/CS/320.

[9] David Benoit, Erik D. Demaine, J. Ian Munro, Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Representing trees of higher degree. *Algorithmica*, 43(4) :275–292, 2005.

[10] Daniel K. Blandford. *Compact Data Structures with Fast Queries.* PhD thesis, Carnegie Mellon University, 2006.

[11] Daniel K. Blandford and Guy E. Blelloch. Dictionaries using variable-length keys and data, with applications. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005*, pages 1–10, 2005.

[12] Daniel K. Blandford, Guy E. Blelloch, David E. Cardoze, and Clemens Kadow. Compact representations of simplicial meshes in two and three dimensions. *Int. J. Comput. Geometry Appl.*, 15(1) :3–24, 2005.

[13] Daniel K. Blandford, Guy E. Blelloch, and Ian A. Kash. An experimental analysis of a compact graph representation. In *ALENEX/ANALC*, pages 49–61, 2004.

[14] D. Blanford, G. Blelloch, and I. Kash. Compact representations of separable graphs. In *Proc. of the Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 342–351, 2003.

[15] J.-D. Boissonnat, O. Devillers, S. Pion, M. Teillaud, and M. Yvinec. Triangulations in cgal. *Comput. Geom. Theory Appl.*, 22 :5–19, 2002.

[16] J.-D. Boissonnat and M. Yvinec. *Géométrie algorithmique*. Ediscience Int., first edition, 1995.

[17] N. Bonichon, C. Gavoille, and N. Hanusse. An information-theoretic upper bound of planar graphs using triangulation. In *Proc. of 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2607 of *LNCS*, pages 499–510. Springer, 2003.

[18] N. Bonichon, C. Gavoille, N. Hanusse, D. Poulalhon, and G. Schaeffer. Planar graphs, via well-orderly maps and trees. *Graphs and Combinatorics*, 22(2) :185–202, 2006.

[19] N. Bonichon, C. Gavoille, and A. Labourel. Edge partition of toroidal graphs into forests in linear time. In *7th International Conference on Graph Theory (ICGT)*, volume 22 of *ENCS*, pages 421–425, 2005.

[20] Nicolas Bonichon. *Aspects algorithmiques et combinatoires des réaliseurs des graphes plans maximaux*. PhD thesis, Université Bordeaux I, 2002.

[21] P. Bose and G. Toussaint. Characterizing and efficiently computing quadrangulations of planar point sets. *Comput. Aided Design*, 14(8) :763–785, 1997.

[22] A. Brodnik, S. Carlsson, E. D. Demaine, J. I. Munro, and R. Sedgewick. Resizable arrays in optimal time and space. In *Proc. 6th Workshop on Algorhtms and Data Structures (WADS)*, volume 1663 of *LNCS*, pages 37–48. Springer, 1999.

[23] S. Campagna, L. Kobbelt, and H. P. Seidel. Direct edges - a scalable representation for triangle meshes. *Journal of Graphics tools*, 3(4) :1–12, 1999.

[24] L. Castelli-Aleardi and O. Devillers. Canonical triangulation of a graph, with a coding application. Technical report, INRIA, 2004. Rapport de recherche 5231.

[25] L. Castelli-Aleardi, O. Devillers, and A. Mebarki. 2d triangulation representation using stable catalogs. In *Proc. of 18th Canadian Conference on Computational Geometry (CCCG)*, pages 71–74, 2006.

[26] L. Castelli-Aleardi, O. Devillers, and G. Schaeffer. Dynamic updates of succinct triangulations. In *Proc. of 17th Canadian Conference on Computational Geometry (CCCG)*, pages 135–138, 2005.

[27] L. Castelli-Aleardi, O. Devillers, and G. Schaeffer. Succinct representation of triangulations with a boundary. In *Proc. 9th Workshop on*

Algorhtms and Data Structures (WADS), volume 3608 of *LNCS*, pages 134–145. Springer, 2005.

[28] L. Castelli-Aleardi, O. Devillers, and G. Schaeffer. Optimal succinct representations of planar maps. In *Proc. of 22nd ACM Annual Symposium on Computational Geometry (SoCG)*, pages 309–318, 2006.

[29] Y.-T. Chiang, C.-C. Lin, and H.-I. Lu. Orderly spanning trees with applications to graph encoding and graph drawing. *SODA*, pages 506–515, 2001.

[30] Yi-Ting Chiang, Ching-Chi Lin, and Hsueh-I Lu. Orderly spanning trees with applications. *SIAM J. Comput.*, 34(4) :924–945, 2005.

[31] R.C.-N Chuang, A. Garg, X. He, M.-Y. Kao, and H.-I. Lu. Compact encodings of planar graphs via canonical orderings and multiple parentheses. *Automata, Laguages and Programming*, pages 118–129, 1998.

[32] D. R. Clark and J. I. Raman. Efficient suffix trees on secondary storage. In *Proc. of the 7th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 383–391, 2001.

[33] S. Cook and R. Reckhow. Time bounded random access machines. *Journal of Computer and System Sciences*, 7 :354–375, 1973. (preliminary version in STOC '72).

[34] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry : Algorithms and Applications*. Springer, 1997.

[35] R. Diestel. *Graph Theory*. Springer, second edition, 2000.

[36] F.Hurtado, M. Noy, and J. Urrutia. Flipping edges in triangulations. *Discrete & Computational Geometry*, 22(3) :333–346, 1999.

[37] P. Flajolet and R. Sedgewick. Analytic combinatorics. Preliminary draft, 2004. `http ://algo.inria.fr/flajolet/Publications/books.html`.

[38] H. De Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10 :41–51, 1990.

[39] M. L. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *Proc. of STOC*, pages 345–354, 1989.

[40] Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Syst. Sci.*, 47(3) :424–436, 1993.

[41] Michael L. Fredman and Dan E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. Syst. Sci.*, 48(3) :533–551, 1994.

[42] E. Fusy, D. Poulalhon, and G. Schaeffer. Dissections and trees, with applications to optimal mesh encoding and to random sampling. In

Proc. of the 16th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA), pages 690–699, 2005.

[43] Pierre-Marie Gandoin. *Compression progressive et sans perte de structures géométriques.* PhD thesis, Université de Nice-Sophia Antipolis, 2001.

[44] Pierre-Marie Gandoin and Olivier Devillers. Progressive lossless compression of arbitrary simplicial complexes. *ACM Trans. Graph.*, 21(3) :372–379, 2002.

[45] R. Geary, R. Raman, and V. Raman. Succinct ordinal trees with level-ancestor queries. In *Proc. of the 15th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1–10, 2004.

[46] Richard F. Geary, Naila Rahman, Rajeev Raman, and Venkatesh Raman. A simple optimal representation for balanced parentheses. In *Combinatorial Pattern Matching, 15th Annual Symposium (CPM)*, pages 159–172, 2004.

[47] R. González, S. Grabowski, V. Mäkinen, and G. Navarro. Practical implementation of rank and select queries. In *Poster in Proc. of 4th Intern. Workshop on Efficient and Experimental Algorithms (WEA)*, 2005.

[48] C. Gotsman. On the optimality of valence-based connectivity coding. *Computer Graphics Forum*, 22 :99–102, 2003.

[49] C. Gotsman, S. Gumhold, and L. Kobbelt. Simplification and compression of 3d meshes. In *Proc. of the European Summer School on Principles of Multiresolution in Geometric Modelling*, 2001.

[50] I. P. Goulden and D. M. Jackson. *Combinatorial enumeration.* Dover Publications Inc., Mineola, NY, 2004. Reprint of the 1983 original.

[51] Stefan Gumhold. *Mesh Compression.* PhD thesis, University of Tübingen, 2000.

[52] Stefan Gumhold. Optimizing markov models with applications to triangular connectivity coding. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 331–338, 2005.

[53] Stefan Gumhold, Stefan Guthe, and Wolfgang Straßer. Tetrahedral mesh compression with the cut-border machine. In *IEEE Visualization*, pages 51–58, 1999.

[54] Stefan Gumhold and Wolfgang Straßer. Real time compression of triangle mesh connectivity. In *SIGGRAPH*, pages 133–140, 1998.

[55] T. Hagerup, P. B. Miltersen, and R.Pagh. Deterministic dictionaries. *J. Algorithms*, 41(1) :69–85, 2001.

[56] X. He, M.-Y. Kao, and H.-I Lu. Linear-time succint encodings of planar graphs via canonical orderings. *SIAM J. on Discrete Mathematics*, 12 :317–325, 1999.

[57] X. He, M.-Y. Kao, and H.-I Lu. A fast general methodology for information theoretically optimal encodings of graphs. *SIAM J. on Computing*, 30 :838–846, 2000.

[58] E. Heighway. A mesh generator for automatically subdividing irregular polygons into quadrilaterals. *IEEE Trans. on Magnetics*, 19(6) :2535–2538, 1983.

[59] P.G. Howard and J.S. Vitter. Analysis of arithmetic coding for data compression. *Information processing and Management*, 28(6) :749–763, 1992.

[60] D. A. Huffman. A method for the construction of minimum-redudancy codes. In *Proc. Inst. Radio Eng.*, volume 40, pages 1098–1101, 1952.

[61] F. Hurtado and M. Noy. Graph of triangulations of a convex polygon and tree of triangulations. *Comput. Geom.*, 13(3) :179–188, 1999.

[62] M. Isenburg. Compressing polygon mesh connectivity with degree duality prediction. In *Proceeding of Graphics Interface*, 2002.

[63] Martin Isenburg. *Compression and Streaming of Polygon Meshes*. PhD thesis, UNC at Chapell Hill, 2005.

[64] Martin Isenburg and Pierre Alliez. Compressing hexahedral volume meshes. *Graphical Models*, 65(4) :239–257, 2003.

[65] Martin Isenburg and Peter Lindstrom. Streaming meshes. In *16th IEEE Visualization Conference (VIS 2005)*, page 30, 2005.

[66] Martin Isenburg, Peter Lindstrom, and Jack Snoeyink. Streaming compression of triangle meshes. In *Third Eurographics Symposium on Geometry Processing*, pages 111–118, 2005.

[67] Martin Isenburg and Jack Snoeyink. Mesh collapse compression. In *Symposium on Computational Geometry*, pages 419–420, 1999.

[68] Martin Isenburg and Jack Snoeyink. Face fixer : compressing polygon meshes with properties. In *SIGGRAPH*, pages 263–270, 2000.

[69] Martin Isenburg and Jack Snoeyink. Spirale reversi : Reverse decoding of the edgebreaker encoding. *Comput. Geom.*, 20(1-2) :39–52, 2001.

[70] Martin Isenburg and Jack Snoeyink. Graph coding and connectivity compression, 2004. manuscript.

[71] Martin Isenburg and Jack Snoeyink. On the non-redundancy of split offsets in degree coding, 2005. manuscript.

[72] Martin Isenburg and Jack Snoeyink. Early-split coding of triangle mesh connectivity. In *Graphics Interface 06*, pages 89–97, 2006.

[73] A. Itai and M. Rodeh. Representation of graphs. *Acta Informatica*, 17 :215–219, 1982.

[74] G. Jacobson. Space efficients static trees and graphs. In *Proceedings of the IEEE Symposium on Foundations of Computerb Science (FOCS)*, pages 549–554, 1989.

[75] M. Kallmann and D. Thalmann. Star-vertices : a compact representation for planar meshes with adjacency information. *J. of Graphics Tools*, 6 :7–18, 2002.

[76] Goos Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16(1) :4–32, 1996.

[77] K. Keeler and J. Westbrook. Short encondings of planar graph and maps. *Discrete Appl. Math.*, 58 :239–252, 1995.

[78] L. Kettner. Using generic programming for designing a data structure for polyhedral surfaces. *Comput. Geom. Theory Appl.*, 13 :65–90, 1999.

[79] A. Khodakovsky, P. Alliez, M. Desbrun, and P. Schroder. Near-optimal connectivity encoding of 2-manifold polygon meshes. *J. of the Graph. Models*, 2002.

[80] D. King and J. Rossignac. Guaranteed 3.67v bit encoding of planar triangle graphs. In *11th Canad. Conf. on Computational Geometry*, pages 146–149, 1999.

[81] D. Knuth. The art of computer programming. Preliminary draft, 2004. `http ://www-cs-faculty.stanford.edu/~knuth/taocp.html`.

[82] Boris Kronrod and Craig Gotsman. Efficient coding of nontriangular mesh connectivity. *Graphical Models*, 63(4) :263–275, 2001.

[83] T. Lewiner, M. Craizer, H. Lopes, S. Pesco, L. Velho, and E. Medeiros. Gencode : geometry-driven compression in arbitrary dimension and co-dimension. In *18th Brazilian Symposium on Computer Graphics and Image Processing, (SIBGRAPI 2005)*, pages 249–256, 2005.

[84] T. Lewiner, M. Craizer, H. Lopes, S. Pesco, L. Velho, and E. Medeiros. Gencode : geometry-driven compression for general meshes. In *19th Brazilian Symposium on Computer Graphics and Image Processing, (SIBGRAPI 2006)*, page to appear, 2006.

[85] Thomas Lewiner. *Mesh Compression from Geometry.* PhD thesis, Université Paris 6, 2005.

[86] Thomas Lewiner, Hélio Lopes, Jarek Rossignac, and Antônio Wilson Vieira. Efficient edgebreaker for surfaces of arbitrary topology. In *XVII Brazilian Symposium on Computer Graphics and Image Processing, (SIBGRAPI 2004)*, pages 218–225, 2004.

[87] M. Li and P. Vitányi. *An introduction to Kolmogorov complexity and its applications.* Springer, second edition, 1997.

[88] R.J. Lipton and R.E. Tarjan. A separator theorem for planar graphs. *SIAM Journal of Applied Mathematics*, 36 :2 :177–189, 1979.

[89] Hélio Lopes, Jarek Rossignac, Alla Safonova, Andrzej Szymczak, and Geovan Tavares. Edgebreaker : a simple implementation for surfaces with handles. *Computers & Graphics*, 27(4) :553–567, 2003.

[90] H.-I Lu. Linear-time compression of bounded-genus graphs into information-theoretically optimal number of bits. In *SODA*, pages 223–224, 2002.

[91] Gary L. Miller, Shang-Hua Teng, William P. Thurston, and Stephen A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *J. ACM*, 44(1) :1–29, 1997.

[92] Gary L. Miller and William P. Thurston. Separators in two and three dimensions. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing (STOC)*, pages 300–309, 1990.

[93] R. C. Mullin. On counting rooted triangular maps. *Canad. J. Math*, 17 :373–382, 1965.

[94] J. I. Munro, V. Raman, and A. J. Storm. Representing dynamic binary trees succintly. In *Proc. of the Twelfth ACM-SIAM Annual Symposium on Discrete Algorithms (SODA)*, pages 529–536, 2001.

[95] J. Ian Munro, Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct representations of permutations. In *Automata, Languages and Programming, 30th International Colloquium (ICALP)*, pages 345–356, 2003.

[96] J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses and static trees. *SIAM J. Comput.*, 31(3) :762–776, 2001.

[97] J. Ian Munro, Venkatesh Raman, and S. Srinivasa Rao. Space efficient suffix trees. *J. Algorithms*, 39(2) :205–222, 2001.

[98] J. Petersen. Die theorie des regularen graphs (the theory of regular graphs). *Acta Mathematica*, 15 :193–220, 1891.

[99] D. Poulalhon and G. Schaeffer. Optimal coding and sampling of triangulations. In *Proc. of Automata, Languages and Programming, 30th International Colloquium (ICALP)*, volume 2719 of *LNCS*, pages 1080–1094. Springer, 2003.

[100] Dominique Poulalhon. *Problèmes énumératifs autour des cartes combinatoires et des factorisations dans le groupe symétrique*. PhD thesis, Ecole Polytechnique, 2002.

[101] Sylvain Prat, Patrick Gioia, Yves Bertrand, and Daniel Meneveaux. Connectivity compression in an arbitrary dimension. *The Visual Computer*, 21(8-10) :876–885, 2005.

[102] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C : The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.

[103] R. Raman, V. Raman, and S.S. Rao. Succint indexable dictionaries with application to encoding k-ary trees and multisets. In *SODA*, pages 233–242, 2002.

[104] R. Raman and S.S. Rao. Static dictionaries supporting rank. In *Algorithms and Computation, 10th International Symposium (ISAAC)*, volume 1741 of *LNCS*, pages 18–26. Springer, 1999.

[105] Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct dynamic data structures. In *Algorithms and Data Structures, 7th International Workshop (WADS)*, volume 2125 of *LNCS*, pages 426–437. Springer, 2001.

[106] V. Raman and S.S. Rao. Succint dynamic dictionaries and trees. In *Proc. of Automata, Languages and Programming, 30th International Colloquium (ICALP)*, volume 2719 of *LNCS*, pages 357–366. Springer, 2003.

[107] S. Ramaswami, P. A. Ramos, and G. Toussaint. Converting triangulations to quadrangulations. *Comput. Geom. Theory Appl.*, 9(4) :257–276, 1998.

[108] J. Rissanen. Generalized kraft inequality and arithmetic coding. *IBM Journal of Research and Development*, 20 :198–203, 1976.

[109] J. Rossignac. Edgebreaker : Connectivity compression for triangle meshes. *IEEE Trans. Visual. and Comp. Graph.*, 5 :47–61, 1999.

[110] K. Sadakane and R. Grossi. Squeezing succinct data structures into entropy bounds. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1230–1239, 2006.

[111] D. Salomon. *Data compression : the complete reference*. Springer, 2000.

[112] F. Santos and R Seidel. A better upper bound on the number of triangulations of a planar point set. *Journal of Combinatorial Theory, Ser. A*, 102(1) :186–193, 2003.

[113] Gilles Schaeffer. *Conjugaison d'arbres et cartes combinatoires aléatoires*. PhD thesis, Université Bordeaux I, 1998.

[114] Walter Schnyder. Embedding planar graphs on the grid. In *Proc. of the First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 138–148, 1990.

[115] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27 :379–423, 623–656, 1948.

[116] M. Sharir and E. Welzl. Random triangulations of planar point sets. In *Proc. of SoCG*, page 0, 2006.

[117] Andrzej Szymczak, Davis King, and Jarek Rossignac. An edgebreaker-based efficient compression scheme for regular meshes. *Comput. Geom.*, 20(1-2) :53–68, 2001.

[118] Andrzej Szymczak and Jarek Rossignac. Grow & fold : compressing the connectivity of tetrahedral meshes. *Computer-Aided Design*, 32(8-9) :527–537, 2000.

[119] Gabriel Taubin and Jarek Rossignac. Geometric compression through topological surgery. *ACM Trans. Graph.*, 17(2) :84–115, 1998.

[120] C. Touma and C. Gotsman. Triangle mesh compression. In *Graphics Interface*, pages 26–34, 1998.

[121] G. Toussaint. Quadrangulations of planar sets. In *Proc. 9th Workshop on Algorhtms and Data Structures (WADS)*, pages 218–227, 1995.

[122] W. Trotter and J. Moore. Some theorems on graphs and posets. *Discrete Mathematics*, 15 :79–84, 1976.

[123] W. Trotter and J. Moore. The dimension of planar posets. *Journal of combinatorial theory*, 22 :54–67, 1977.

[124] W.T. Trotter. *Combinatorics and partially ordered sets.* Johns Hopkins Univ. Press, first edition, 1992.

[125] G. Turan. On the succint representation of graphs. *Discrete Applied Mathematics*, 8 :289–294, 1984.

[126] W. Tutte. A census of planar triangulations. *Canadian J. of Mathematics*, 14 :21–38, 1962.

[127] W. Tutte. A census of planar maps. *Canadian J. of Mathematics*, 15 :249–271, 1963.

[128] I.H. Witten, R.M. Neal, and J.G. Cleary. Arithmetic coding for data compression. *Comm. ACM*, 30(6) :520–540, 1987.

[129] M. Yannakakis. Four pages are necessary and sufficient for planar graphs. In *Proc. of the Eighteenth Annual ACM Symposium on Theory of Computing (STOC)*, pages 104–108, 1986.

[130] G. M. Ziegler. *Lectures on Polytopes.* Springer, revised 1998 edition, 1995.

[131] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Information Theory*, IT-23(3) :337–343, 1977.

[132] J. Ziv and A. Lempel. Compression of individual sequences via variable rate coding. *IEEE Trans. Information Theory*, IT-24(5) :530–536, 1978.