Chapitre 4

La carte topologique en dimension 2

Après avoir étudié plusieurs modèles de représentation d'images 2d segmentées en régions et en avoir déduit que les modèles présentant le plus d'avantages et étant le mieux adaptés à nos problématiques étaient le TGF ou son équivalent, les cartes discrètes, nous avons ensuite cherché à les étendre en dimension supérieure. Les principales difficultés que nous avons rencontrées proviennent du fait que les définitions du TGF et des cartes discrètes sont basées sur la notion de frontière, qui est définie en interpixel comme nous verrons section 4.1. La définition de frontière en dimension 2 ne pose pas de problème, car une frontière entre deux régions est simplement une courbe 1d que nous pouvons définir assez simplement.

Le problème est autrement plus complexe en dimension 3, où la frontière entre deux régions est une surface, i.e. un objet de dimension 2. Cela complique les différentes définitions, car il peut y avoir de nombreuses configurations différentes difficiles à prendre en compte. De plus, les cartes combinatoires représentent les objets par leurs bords. Il faut donc définir le bord des surfaces 2d afin d'avoir des courbes en dimension 1, qui peuvent être représentées avec des brins.

Une autre difficulté provient de la définition des différents algorithmes d'extraction. En effet, pour le TGF, Christophe Fiorio a défini un algorithme optimal d'extraction qui calcule le TGF d'une image de dimension 2 en une seule passe de l'image. Mais cet algorithme est complexe et difficilement extensible en dimension supérieure [BFP99]. Il en est de même pour le modèle de Jean-Philippe Domenger, qui est défini à partir des frontières interpixel, à l'aide d'un algorithme de suivi de contour. Ce suivi est facile à effectuer en dimension 2, mais beaucoup plus complexe en dimension 3, étant donné qu'il faut parcourir des surfaces afin de trouver leurs bords.

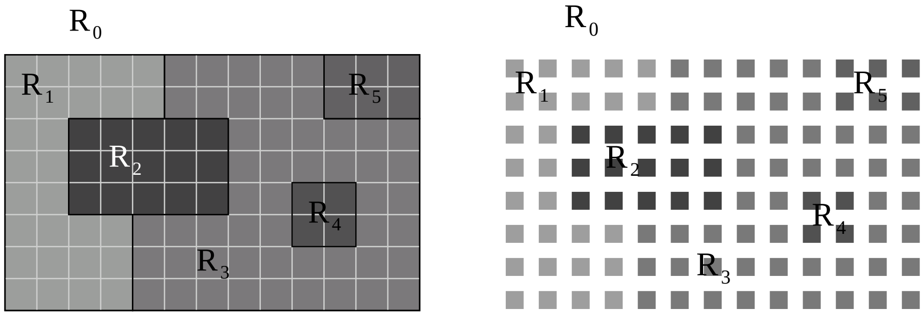
Pour définir la carte topologique en dimension 3, nous avons d'abord défini la carte topologique en dimension 2 de sorte que la définition soit simple et aisément extensible en dimension supérieure. De plus, nous avons conservé l'objectif principal : obtenir une structure minimale, caractéristique des objets qu'elle représente, et invariante par rotation, translation et homothétie. Nous cherchons également à définir un algorithme d'extraction facilement extensible en dimension supérieure.

Cette démarche nous a amené à définir la *carte topologique*, nom que nous donnons à la carte combinatoire représentant les frontières interpixel d'une image segmentée en régions, et donc équivalente au TGF et aux cartes discrètes présentés au chapitre précédent. Pour cela, nous introduisons une notion de niveau de simplification qui permet de définir la carte topologique de manière progressive. Chaque niveau s'obtient à partir du niveau précédent par application d'un type particulier de fusion. Ce type de définition est aisément extensible en dimension n. De plus, cette démarche fournit un premier algorithme d'extraction de notre modèle étroitement lié à la définition de la carte topologique.

Puis nous avons cherché à optimiser cet algorithme « naïf » qui, bien que linéaire en le nombre de brins de la carte, effectue plusieurs passes sur celle-ci avant d'obtenir la carte topologique. De plus, cet algorithme a comme principal inconvénient de créer un grand nombre de brins, avant d'en détruire une bonne partie au cours des différentes passes de simplification. Nous avons pour cela, repris l'idée de l'algorithme optimal de Christophe Fiorio [Fio95, Fio96], qui consiste à balayer l'image de haut en bas et de gauche à droite avec une fenêtre de 2×2 pixels, en exécutant un traitement dépendant de la configuration locale des pixels de cette fenêtre. Nous avons adapté cette idée à nos divers niveaux de carte, et avons étudié les configurations à traiter pour chacun de ces niveaux.

L'idée générale de notre démarche est de calculer la carte de manière incrémentale et directe, au moyen d'un seul balayage de l'image. De plus, les traitements à effectuer dépendent uniquement de la configuration locale des pixels, et créent exactement le nombre de brins nécessaires. Les niveaux de simplification offrent ici un avantage important, car ils permettent de factoriser les cas pouvant se traiter de manière identique. En effet, nous étudions pour chaque niveau, les cas supplémentaires à traiter par rapport au niveau précédent. Au final, nous obtenons le nombre minimal de cas différents à traiter afin d'extraire la carte topologique. Ces regroupements ne sont pas très importants en dimension 2, étant donné qu'il existe seulement 15 cas différents au total. Mais nous verrons au chapitre 5 que ces regroupements sont cruciaux en dimension supérieure et permettent de définir l'algorithme optimal.

Nous commençons section 4.1 par présenter le cadre de ce travail, en rappelant les notions de segmentation en régions, d'interpixel, et celle de frontière interpixel. Section 4.2 nous introduisons la notion de niveau de simplification et définissons formellement la carte topologique au moyen de ces niveaux. Mais les cartes combinatoires ne représentent que la topologie des objets, et il faut leur adjoindre un modèle géométrique afin de coder toutes les informations des objets modélisés. La présentation de divers modèles géométriques et la manière de les associer aux différents niveaux de carte est l'objet de la section 4.3. Nous étudions ensuite section 4.4 le premier algorithme naïf d'extraction de nos diverses structures. Puis section 4.5 nous présentons l'algorithme optimal d'extraction. Section 4.6 nous présentons quelques résultats et effectuons une comparaison des diverses cartes obtenues avant de conclure ce chapitre section 4.7.



a. Plongée dans le plan Euclidien. b. Représentée sous forme discrète.

FIG. 4.1 – Une image 2d segmentée en régions.

4.1 Images, segmentation en régions et interpixel

Nous considérons, pour notre processus d'extraction, que l'image a préalablement fait l'objet d'une segmentation en régions. Nous rappelons section 4.1.1 cette notion qui permet de préciser le cadre de ce travail. Nous rappelons également brièvement la notion d'*interpixel* section 4.1.2, ce qui permet de définir la notion de frontière, centrale dans ce travail.

4.1.1 La segmentation en régions

Nous donnons ici quelques notations utilisées dans la suite de ce travail. Un *pixel* est un point coloré du plan discret $\mathbb{Z}^2$, et une *image* de dimension 2 est un ensemble fini de pixels. Nous utilisons les notions classiques de 4-*voisinage*, de chemin et de 4-*connexité*. Nous travaillons en 4-connexité car les cartes combinatoires ne permettent pas de représenter les non variétés, et donc le 8-voisinage.

La *segmentation en régions* d'une image I de dimension 2 est une partition de I en un ensemble de régions $\{R_1, R_2, \dots R_k\}$ telles que chaque région R_i est une composante 4-connexe.

Rappelons qu'un ensemble $\{R_1, R_2, \dots R_k\}$ est une *partition* d'un ensemble I s'il vérifie :
- $\cup_{i=1}^{k} R_i = I$;
- $\forall i$, $1 \leq i \leq k$, $\forall j$, $1 \leq j \leq k$, $i \neq j \Rightarrow R_i \cap R_j = \emptyset$.

Afin d'éviter un traitement particulier des pixels du bord de l'image, nous considérons que l'image est incluse dans une *région infinie* R_0 contenant l'ensemble des pixels n'appartenant pas à l'image. Ainsi nous avons des propriétés homogènes pour tous les pixels de l'image. La propriété principale, que nous utilisons par la suite, est que chaque pixel possède 4 voisins pour la 4-connexité. Cette région infinie permet également de traiter n'importe quel type d'image, et pas uniquement celles qui sont rectangulaires et sans trous.

La figure 4.1 montre un exemple d'image en dimension 2 segmentée en régions, tout d'abord plongée dans le plan Euclidien (figure 4.1.a), ou sous forme discrète (figure 4.1.b). Elle comporte cinq régions distinctes, plus la région infinie. Nous représentons sur nos figures chaque région par une couleur unique. Une notion importante et utile pour la suite de ce travail est l'inclusion de régions que nous définissons ci-après.

Définition 12 (inclusion) *Une région R_i est* inclusе *dans une région R_j si et seulement si tout chemin 4-connexe allant d'un pixel de R_i vers un pixel de R_0 (la région infinie) possède au moins un pixel appartenant à la région R_j.*

Informellement, cette notion d'inclusion correspond au fait qu'une région en « encercle » complètement une autre, sans aucune contrainte sur la manière dont est effectué cet encerclement. Cette définition, appliquée à l'exemple figure 4.1, indique que la région R_4 est incluse dans deux régions : R_3 et R_0. Toutes les autres régions sont, sans distinction, incluses uniquement dans la région infinie. Remarquons que chaque région est au moins incluse dans la région infinie. Cela est dû au fait que la relation d'inclusion est une relation d'ordre. En effet, elle est :

- **Réflexive** : chaque région r est incluse dans elle-même car tout chemin allant d'un pixel de r à un autre pixel de r possède forcément au moins un pixel de r ;
- **Transitive** : si $R_i \subseteq R_j$ et $R_j \subseteq R_k$, alors $R_i \subseteq R_k$. Cela peut se prouver en étudiant les chemins de R_i à R_0. Ils passent forcément par la région R_j, et les chemins de R_j à R_0 passent forcément par R_k, par définition de l'inclusion. Donc tous les chemins allant de R_i à R_0 traversent bien R_k ;
- **Antisymétrique** : si $R_i \subseteq R_j$, alors nous n'avons pas $R_j \subseteq R_i$ (pour $R_i \neq R_j$). Cela peut se prouver par l'absurde car si c'était le cas, tout chemin de R_i à R_0 passerait par R_j et tout chemin de R_j à R_0 passerait par R_i. Nous pouvons prouver dans ce cas que R_i n'est pas 4-connexe, ce qui contredit la définition de la segmentation en régions.

Cette notion d'inclusion sera utile afin de conserver la relation correspondante sur les régions, comme nous le verrons section 4.4.3. Pour cela, il n'est pas nécessaire de conserver toutes les relations d'inclusion. En effet, il suffit de conserver la réduction transitive de la relation d'ordre, les relations de transitivité pouvant alors aisément se retrouver à partir de cette nouvelle relation. Nous appelons cette relation l'*inclusion directe*.

Définition 13 (inclusion directe) *Une région R_i est* directement incluse *dans une région R_j si et seulement si $R_i \subseteq R_j$, et $\nexists R \neq R_j$ tel que $R_i \subseteq R$ et $R \subseteq R_j$.*

Sur notre exemple figure 4.1, la seule différence par rapport à la notion d'inclusion est pour la région R_4 qui est directement incluse uniquement dans R_3. Les autres régions étant incluses uniquement dans R_0, elles sont donc également directement incluses dans R_0.

Lorsqu'une image a été segmentée en régions, nous nous intéressons aux contours de celles-ci. Ces contours vont permettre de structurer les diverses relations topologiques entre les régions et d'obtenir ainsi le maximum d'informations utilisables pour des processus d'analyse ou de traitement. Ces relations topologiques se calculent en étudiant les contours des régions segmentées, et la manière dont ils se positionnent les uns par rapport aux autres.

Plusieurs travaux ont été réalisés autour de la notion de contours dans une image discrète. Il apparaît qu'utiliser une topologie basée sur la notion d'*interpixel* permet de définir correctement les contours de sorte qu'ils vérifient les propriétés topologiques habituelles (comme par exemple le théorème de Jordan [Ale61]). Nous présentons brièvement cette notion. Pour une définition complète on se réfèrera à [Kov89, KR89, KKM90, KKM91, Fio95].

4.1.2 Interpixel et frontières

La notion d'*interpixel* consiste à considérer une image non pas uniquement comme une matrice de pixels, mais comme une subdivision de l'espace (ici de dimension 2) en un ensemble de cellules de dimension inférieure ou égale à celle de l'espace. Les différentes cellules obtenues sont les *pixels*, cellules de dimension 2, les *lignels*, cellules de dimension 1 « entre » deux pixels, et les *pointels*, cellules de dimension 0 « entre » deux lignels. Nous appelons i-*cellule* une cellule de dimension i. Ces différentes cellules sont présentées figure 4.2.

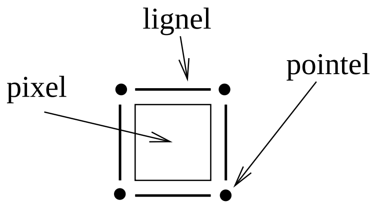
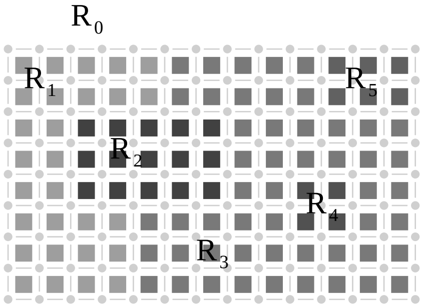


FIG. 4.2 – Les divers éléments interpixel en dimension 2.

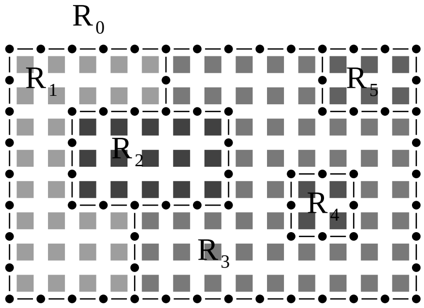
La relation d'*incidence* permet de lier ces diverses cellules. Deux cellules sont dites *incidentes* si l'une appartient au *bord* de l'autre, et *adjacentes* si elles ont même dimension et sont incidentes à la même $(i-1)$-cellule. Remarquons que nous pouvons étendre cette notion d'adjacence en définissant que deux i-cellules sont j-adjacentes ($j < i$) si elles sont incidentes à la même j-cellule. Mais comme nous travaillons uniquement, pour un espace de dimension n, en $2n$ connexité, nous n'utilisons pas cette adjacence « étendue » et nous limitons donc à la première notion d'adjacence. Remarquons que ces notions sont les mêmes que celles présentées au chapitre 2, lors du rappel sur les cartes combinatoires. En effet, les cartes combinatoires représentent les subdivisions de l'espace, ce qui est également le cas d'une topologie interpixel. C'est pour cette raison qu'il est naturel de représenter les contours interpixel d'une image avec une carte combinatoire.

À l'aide de cette notion d'interpixel, nous pouvons définir de manière formelle la notion de courbe puis celle de frontière.

Définition 14 (courbe) *Une* courbe *est une suite de pointels et de lignels* $c_0, \dots, c_{2k}$ *vérifiant :*
- $\forall i \in \{0, \dots, k\}$, c_{2i} *est un pointel ;*
- $\forall i \in \{0, \dots, k-1\}$, c_{2i+1} *est un lignel ;*
- $\forall i \in \{0, \dots, 2k-1\}$, c_i *et* c_{i+1} *sont incidents.*



a. L'image présentée figure 4.1 avec tous les éléments interpixel.



b. Les frontières interpixel des régions de cette image.

FIG. 4.3 – Frontières interpixel.

Une courbe est dite *simple* si toutes ses cellules sont distinctes[1]. Elle est dite *fermée* si $c_0 = c_{2k}$. Une *frontière* interpixel entre deux régions R_i et R_j est un ensemble de courbes simples disjointes deux à deux, tel que chaque lignel de ces courbes est incident à exactement un pixel de R_i et un pixel de R_j. Nous appelons *courbe frontière* une courbe appartenant à l'une des frontières de l'image. Ces courbes sont maximales : tout lignel de l'image incident à un pixel de R_i et un pixel de R_j appartient forcément à une courbe frontière, et deux lignels adjacents appartenant à une frontière appartiennent forcément à la même courbe frontière (ils sont séparés dans cette courbe par le pointel incident à ces deux lignels).

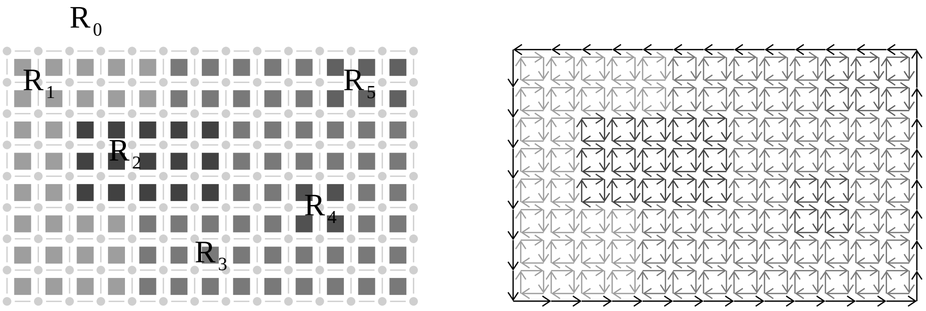
Figure 4.3.b est présenté l'ensemble des frontières interpixel de l'image figure 4.1. Nous y voyons que la frontière entre les régions R_1 et R_3 est composée de deux courbes distinctes, que celle entre R_3 et R_4 est composée d'une courbe fermée, et que celle entre R_1 et R_4 est vide car ces deux régions ne sont pas adjacentes.

Nous notons $frontière(R_i, R_j) = \{c_1, \ldots, c_l\}$ la frontière entre les régions R_i et R_j composée des courbes $c_1 \ldots c_l$. Nous considérons que $frontière(R_i, R_i) = \emptyset$. En effet, une région ne possède pas de frontière avec elle-même. Enfin, nous notons $Frontières(R_i)$ pour l'ensemble des frontières de la région R_i : $\cup_{j=1}^{k} frontière(R_i, R_j)$. Nous appelons respectivement *pointel frontière* et *lignel frontière* un pointel et un lignel appartenant à une frontière de l'image. Remarquons que cette définition des frontières reste valable pour les régions du bord de l'image grâce à la région infinie.

4.2 Les niveaux de simplification

Dans la section 4.1.2, nous avons introduit la notion de frontière interpixel. Il faut maintenant définir une structure de données représentant ces frontières ainsi que les diverses relations d'incidences entre les éléments de celles-ci. Pour cela, nous utilisons les cartes combinatoires qui sont bien adaptées au codage des frontières interpixel, car elles

[1]Une courbe est *simple* quand elle ne s'auto-intersecte pas.



a. L'image et ses éléments interpixel. b. La carte de niveau 0.

Fig. 4.4 – Le niveau 0 d'une image 2d.

représentent les subdivisions de l'espace ainsi que leurs relations. De plus, elles définissent totalement la topologie des objets qu'elles représentent. C'est pour cette raison qu'elles ont été utilisées dans de nombreux travaux. Mais il existe plusieurs cartes combinatoires différentes représentant le même objet, et ces travaux ont utilisé diverses cartes suivant leurs besoins particuliers. De plus, définir formellement une carte combinatoire spécifique représentant les frontières interpixel d'une image est difficile et de ce fait souvent éludé.

Nous introduisons ici une notion de niveau de simplification qui permet de rapprocher ces divers travaux. Ces niveaux peuvent de plus se définir formellement à l'aide de définitions algorithmiques simples et progressives. Nous en présentons dans un premier temps seulement l'aspect topologique sans nous préoccuper du lien avec un modèle géométrique. Le lien avec un modèle géométrique est l'objet de la section 4.3.

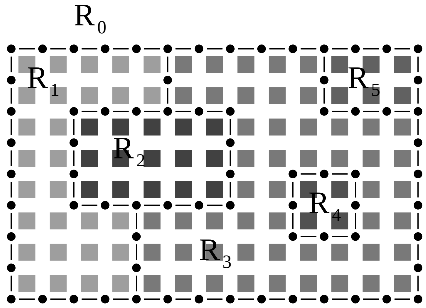
L'idée de notre démarche est de construire la carte combinatoire complète, et de la simplifier petit à petit, tant que cela n'entraîne pas de perte d'information topologique. Ceci permet de définir progressivement la carte représentant les frontières interpixel de l'image, et ainsi d'en simplifier la définition.

4.2.1 Le niveau 0 : la carte complète

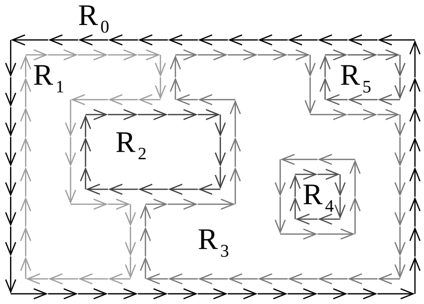
Cette première carte est seulement le point de départ de notre processus. Elle ne représente pas les frontières interpixel de l'image, et c'est pour cette raison que nous l'appelons le niveau 0.

Définition 15 (carte de niveau 0) *La* carte de niveau 0 *correspondant à une image de $n \times m$ pixels, est la carte combinatoire ayant $n \times m$ faces carrées, β_2-cousues entre elles lorsqu'elles sont adjacentes, chacune de ces faces représentant un pixel de l'image, et une autre face « entourant » ces faces représentant la région infinie.*

Cette carte représente tous les éléments interpixel de l'image à traiter. Nous voyons figure 4.4.*b* la carte de niveau 0 de l'image présentée figure 4.4.*a*. Cette carte est composée, pour une image de taille $n \times m$, de $(n \times m) + 1$ faces. $n \times m$ faces carrées composées de



a. L'image et ses frontières interpixel.



b. La carte de niveau 1.

FIG. 4.5 – La carte de niveau 1 d'une image.

4 brins, chacune représentant un pixel de l'image. La face supplémentaire représente la région infinie et est composée de $2 \times (n+m)$ brins. Deux brins sont cousus par β_1 lorsqu'ils représentent deux lignels consécutifs de la même face, et sont cousus par β_2 lorsqu'ils représentent le même lignel de l'image. Remarquons que chaque face est orientée selon le sens trigonométrique inverse, à l'exception de la face infinie. En effet, les brins associés à cette face représentent un bord intérieur qui est donc orienté dans le sens trigonométrique. Nous représentons sur nos figures chaque brin de la couleur de la région auquel il appartient. Notons que cette carte est fermée, chaque brin est cousu par β_1 et β_2.

4.2.2 Le niveau 1 : la carte lignel

Afin d'obtenir une carte combinatoire codant les frontières interpixel de l'image, il faut « supprimer » toutes les arêtes de la carte de niveau 0 représentant des lignels n'appartenant pas à une courbe frontière. Cette opération s'effectue à l'aide de l'opération de fusion, qui modifie deux i-cellules adjacentes pour n'en obtenir plus qu'une. C'est cette opération de fusion qui permet de modifier la carte de niveau 0 afin d'obtenir la carte de niveau 1. Cette opération sera détaillée section 4.4 lors de la présentation de l'algorithme naïf d'extraction de la carte topologique, mais est également l'objet du chapitre 6 permettant de valider notre démarche.

Définition 16 (carte de niveau 1) *La* carte de niveau 1 *est la carte obtenue à partir de la carte de niveau* 0 *en fusionnant chaque couple de faces adjacentes appartenant à la même région.*

Nous voyons figure 4.5.a l'image de notre exemple, représentée avec ses frontières interpixel, et figure 4.5.b la carte de niveau 1 correspondante. Nous pouvons facilement vérifier que cette carte représente bien les frontières interpixel de l'image.

Propriété 1 *Toute arête de la carte de niveau* 1 *correspond exactement à un lignel d'une courbe frontière, et tout lignel d'une courbe frontière est représenté dans la carte de niveau* 1 *par une arête.*

Preuve de la propriété 1

1. *Toute arête de la carte de niveau* 1 *correspond exactement à un lignel d'une courbe frontière :*
   Soit e une arête de la carte de niveau 1*. e est forcément composée de deux brins appartenant à deux régions différentes, sinon les deux faces incidentes auraient été fusionnées et e aurait été détruite lors de cette fusion. Cette arête est de longueur un, car toutes les arêtes de la carte de niveau* 0 *le sont, et nous n'avons pas modifié la longueur des arêtes entre ces deux niveaux.*
2. *Tout lignel d'une courbe frontière est représenté dans la carte de niveau* 1 *par une arête :*
   Soit l un lignel d'une courbe frontière. Par définition des frontières, l est incident à deux pixels appartenant à deux régions différentes. Alors les deux faces correspondant à ces deux pixels ne sont pas fusionnées lors du passage du niveau 0 *au niveau* 1 *et nous avons donc dans la carte de niveau* 1 *une arête représentant ce lignel.*

□

C'est cette propriété qui nous a amené à appeler cette carte de niveau 1 la *carte lignel*. De plus, nous appelons cette carte le niveau 1, car c'est la première carte, et la plus grande en termes de nombre de brins, représentant les frontières interpixel de l'image. En effet, pour ajouter des brins à cette carte, en représentant uniquement les frontières interpixel, il faut représenter un même lignel par plusieurs arêtes. C'est bien entendu réalisable théoriquement (et dans ce cas il n'y a pas de plus grande carte, nous pouvons ajouter un nombre infini d'arêtes) mais peu intéressant en pratique.

Propriété 2 *Soit C une carte lignel codant une image I segmentée en régions. Chaque région (exceptée la région infinie) est représentée dans C par un contour extérieur, et éventuellement par un ou plusieurs contours intérieurs.*

Chaque région possède un contour extérieur, exceptée la région infinie R_0 qui possède uniquement un contour intérieur. Les éventuels contours intérieurs surviennent lorsqu'une région est incluse dans une autre. Nous pouvons voir sur l'exemple figure 4.5.b, que seule la région R_3 possède un contour intérieur (si l'on excepte la région infinie). Nous avons alors déconnecté la carte en plusieurs composantes connexes (ici deux), et nous avons perdu l'information reliant les deux bords maintenant déconnectés. Il faut conserver cette information permettant de relier le bord extérieur d'une région et ses éventuels bords intérieurs, car c'est une information topologique importante, permettant de représenter les « trous » dans une région. Pour cela, nous introduisons un arbre d'inclusion des régions. Cet arbre possède un nœud par région de l'image. Sa racine est la région infinie, et chaque nœud possède comme fils les régions directement incluses dans la région correspondant à ce nœud. À l'aide de cet arbre, nous pouvons retrouver les régions incluses dans n'importe quelle région et donc reconstruire, ou parcourir correctement son bord.

La carte lignel, et plus précisément les contours des régions dans cette carte, vérifie plusieurs propriétés intéressantes :

1. la carte est fermée ;
2. la région infinie est représentée par un unique contour intérieur ;

3. les contours extérieurs sont orientés dans le sens trigonométrique inverse ;
4. les contours intérieurs sont orientés dans le sens trigonométrique ;

Les propriétés 1 à 3 peuvent se démontrer facilement, car elle sont vraies par construction pour le niveau 0, et nous pouvons prouver qu'elles restent vraies après une fusion de faces. Elle sont dont également vraies pour la carte lignel, de par sa construction. La propriété 3 revient à dire que lorsque nous parcourons un contour extérieur dans le sens trigonométrique inverse, la région correspondante se trouve tout le temps à droite du contour. Par contre, c'est l'inverse pour les contours intérieurs : si nous parcourons ces contours dans le sens trigonométrique inverse, la région correspondante se trouve à gauche. C'est pour cette raison que les contours intérieurs sont orientés en sens inverse des contours extérieurs, donc dans le sens trigonométrique. Avec cette orientation, pour n'importe quel contour, la région se trouve tout le temps à droite. Cela explique la propriété 4 et donne une idée de preuve. Remarquons que le sens des contours extérieurs et intérieurs a été choisi par convention, fixé lors de la construction de la carte de niveau 0.

Cette carte lignel dotée d'un arbre d'inclusion contient toutes les informations topologiques de l'image qu'elle représente. Nous venons de voir comment est représentée la relation d'inclusion, au moyen de l'arbre d'inclusion des régions. Cette carte code toutes les 1-cellules et 0-cellules des frontières des régions de l'image. Les 2-cellules ne sont plus représentées, mais nous représentons maintenant les régions (qui sont des ensembles de 2-cellules) au moyen de leurs frontières. Les relations d'adjacences et d'incidences entre n'importe quelles cellules sont toutes représentées dans la carte, de manière explicite ou implicite, de par la définition des cartes combinatoires. Nous nous appuyons sur cette remarque, afin de conserver toutes les informations topologiques au cours de nos différentes simplification. Pour cela, il suffit de montrer, lorsque nous effectuons une simplification, que nous ne perdons pas d'information topologique.

Cette carte lignel est donc la première carte représentant les frontières interpixel d'une image segmentée en régions. Mais cette représentation n'est optimale ni en espace mémoire ni en temps d'exécution. En effet, de par le grand nombre de brins à parcourir, par exemple pour retrouver toutes les régions adjacentes à une région, les opérations sur cette carte sont coûteuses en temps d'exécution. De plus cette carte n'est pas caractéristique de la topologie des images qu'elle représente, n'est pas stable par rotation, translation et homothétie, et n'est pas minimale en nombre de brins. C'est pour ces raisons que nous allons simplifier cette carte et définir la carte de niveau 2.

4.2.3 Le niveau 2 : la carte des frontières

Cette *carte des frontières* a été présentée dans [BFP99]. C'est la carte la plus intuitive représentant les frontières interpixel d'une image. Dans cette carte, chaque arête représente une « portion » rectiligne de frontière. La définition formelle de cette carte s'obtient aisément à partir de la carte lignel qui est le niveau précédent. Nous appelons indifféremment cette carte la carte de niveau 2, car c'est la deuxième carte que nous définissons qui représente les frontières de l'image, ou la carte des frontières qui est le nom original donné dans [BFP99].

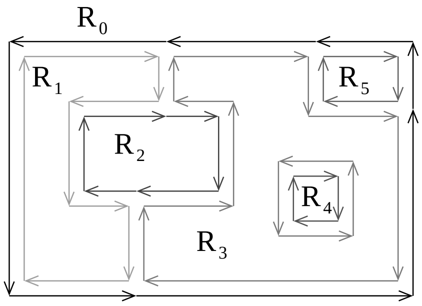


FIG. 4.6 – La carte de niveau 2 de notre exemple.

Définition 17 (carte de niveau 2) *La* carte de niveau 2 *est la carte obtenue à partir de la carte de niveau* 1*, en fusionnant chaque couple d'arêtes adjacentes, alignées et incidentes à un sommet de degré deux.*

La carte des frontières de l'image figure 4.5.a est présentée figure 4.6. La propriété principale de cette carte est que chaque arête représente un segment de droite des frontières interpixel de l'image. Ces arêtes sont désormais de longueur quelconque, contrairement à la carte lignel.

Propriété 3 *Chaque arête d'une carte des frontières correspond à une suite maximale de lignels alignés d'une frontière.*

Cette propriété est évidente, de par la définition de la carte des frontières. En effet, dans la carte de niveau 1, chaque arête correspond à un lignel, et donc lorsque deux lignels sont alignés, et qu'ils sont incidents à un sommet de degré deux, les arêtes correspondantes sont fusionnées pour obtenir la carte de niveau 2. Cette carte ne peut donc pas avoir deux arêtes représentant deux lignels adjacents alignés étant au milieu d'une frontière (ce qui revient à « incident à un sommet de degré deux »). De manière symétrique, si deux arêtes sont adjacentes dans la carte des frontières, alors soit elles sont incidentes à un sommet de degré supérieur à deux, soit elles représentent deux portions de frontières non alignées.

Mais le point le plus important, afin de valider notre démarche, est de garantir l'absence de perte d'information topologiques par rapport à la carte lignel.

Propriété 4 *La carte des frontières représente exactement les mêmes relations topologiques que la carte lignel.*

Cela peut se prouver facilement. La seule modification entre ces deux niveaux, est que nous avons fusionné chaque couple d'arêtes alignées incidentes à des sommets de degré deux. Lorsque deux arêtes sont incidentes à un sommet de degré deux, elles sont alors forcément toutes les deux entre les deux mêmes régions R_i et R_j. Dans ce cas, les deux arêtes représentent la même information topologique d'adjacence entre ces deux régions,

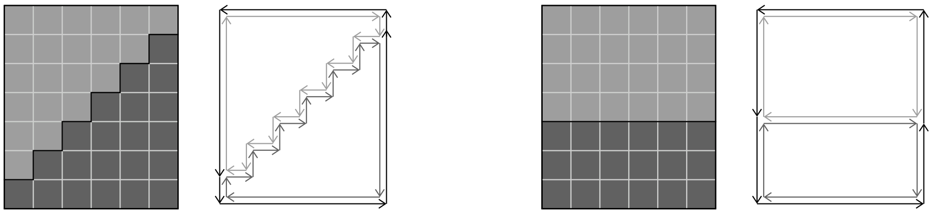


FIG. 4.7 – Deux images topologiquement équivalentes et leurs cartes des frontières.

et le fait de n'en conserver qu'une seule n'entraîne aucune perte d'information. La preuve formelle de la validité des réécritures entre les différents niveaux est l'objet du chapitre 6 dans le cadre plus général de la dimension 3.

En comparant, sur notre exemple, la carte lignel avec la carte des frontières, nous observons que le nombre de brins a considérablement diminué, en passant de 276 brins à 44. Cette baisse importante entraîne évidemment un gain important en espace mémoire, mais également en temps d'exécution, pour les raisons évoquées lors de la présentation de la carte lignel.

Mais ce niveau n'est pas totalement satisfaisant. En effet, le nombre de brins nécessaires pour représenter une frontière entre deux régions dépend de la géométrie de celle-ci. Dans le pire des cas, si une courbe frontière ne possède aucun lignel consécutif aligné (cas de « l'escalier » figure 4.7), le nombre de brins représentant cette frontière sera égal à deux fois le nombre de lignels de celle-ci. Cela a pour conséquence que deux images topologiquement équivalentes peuvent avoir deux cartes des frontières totalement différentes.

Notre but étant d'extraire et de structurer les informations pertinentes des images, il nous semble primordial d'avoir, pour deux images topologiquement équivalentes, deux cartes combinatoires isomorphes. De plus, dans la carte des frontières, il subsiste des arêtes adjacentes incidentes à des sommets de degré deux. Comme pour la carte lignel, ces arêtes représentent la même information topologique, et peuvent donc être simplifiées. La carte des frontières n'est donc pas minimale. Nous définissons donc la carte topologique, qui est la carte de niveau 3, afin d'avoir une structure complètement homogène mais également moins coûteuse en espace mémoire.

4.2.4 Le niveau 3 : la carte topologique

Afin de définir ce dernier niveau de carte, nous conservons notre principe de simplification. Nous partons de la carte des frontières et fusionnons les derniers éléments représentant des informations topologiques de manière redondante : les arêtes adjacentes incidentes à un sommet de degré deux.

Définition 18 (carte de niveau 3) *La* carte de niveau 3 *est la carte obtenue à partir de la carte de niveau 2 en fusionnant chaque couple d'arêtes adjacentes incidentes à un sommet de degré deux.*

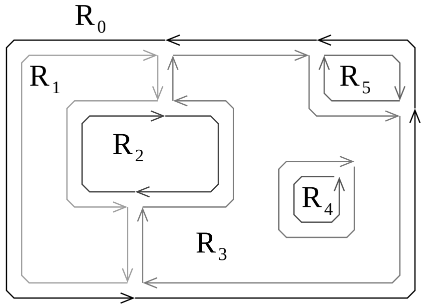


FIG. 4.8 – La carte de niveau 3 de notre exemple.

Nous voyons figure 4.8 la carte de niveau 3 correspondant à notre image d'exemple. La propriété principale de la carte de niveau 3 peut se vérifier sur cet exemple.

Propriété 5 *Chaque arête d'une carte de niveau* 3 *correspond exactement à une courbe frontière de l'image correspondante.*

C'est cette propriété qui nous a amené à appeler cette carte *carte topologique*. En effet, cette carte est minimale, nous ne pouvons plus simplifier quoi que ce soit sans entraîner une perte d'information topologique. C'est bien notre but final : la carte combinatoire minimale représentant les frontières interpixel d'une image. Cette propriété peut se reformuler en : *chaque arête de la carte topologique est incidente à deux sommets de degré strictement supérieur à deux (à l'exception des boucles).* Cela peut se prouver facilement à l'aide de la définition de cette carte. En effet, toutes les arêtes adjacentes incidentes à des sommets de degré deux ont été fusionnées, et il n'en reste donc plus. Cette propriété nous permet de calculer le nombre de brins de cette carte : il est égal à deux fois le nombre de courbes frontière de l'image correspondante. Nous pouvons observer sur la figure 4.8 que la frontière entre les régions R_3 et R_4 est représentée par une boucle. En effet, la région R_4 est incluse dans R_3 et ne possède pas d'autre région adjacente. Le sommet incident à cette boucle est de degré 1, ce qui permet de conserver la boucle dans la carte de niveau 3 étant donné que nous supprimons uniquement les sommets de degré 2.

Pour ce dernier niveau, nous associons à chaque courbe frontière exactement une arête de la carte. La carte obtenue est alors totalement homogène et deux images topologiquement équivalentes seront représentées par deux cartes isomorphes, comme nous pouvons le voir figure 4.9. De plus, cette carte est moins coûteuse en espace mémoire que les précédentes, tout en représentant exactement les mêmes informations topologiques.

Propriété 6 *La carte des frontières représente exactement les mêmes relations topologiques que la carte lignel.*

Pour nous assurer de cette propriété, nous utilisons le même principe que pour la carte des frontières, en montrant que deux arêtes incidentes à un sommet de degré deux représentent la même information topologique. Nous pouvons donc ne conserver qu'une des

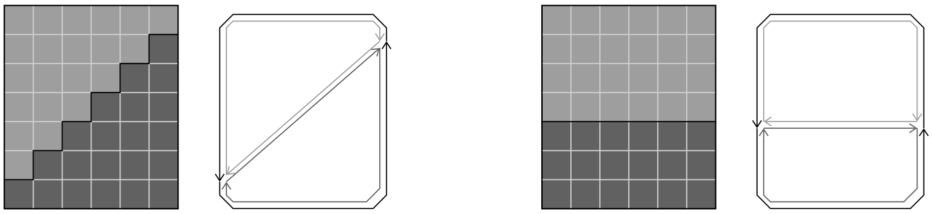


FIG. 4.9 – Deux images topologiquement équivalentes et leurs cartes topologiques qui sont isomorphes.

deux arêtes sans entraîner de perte d'information. Nous voyons sur l'exemple figure 4.8, qu'il ne reste plus que 20 brins qui représentent les 10 courbes frontières de l'image. C'est peu en comparaison des 276 brins de la carte lignel de la même image.

Avec la carte topologique, nous avons codé chaque courbe frontière de manière minimale (une arête) et nous ne pouvons plus « simplifier » la carte obtenue sans perte d'information topologique. Cette carte est donc minimale, stable par rotation, translation et homothétie, et représente bien toutes les informations topologiques de l'image.

Remarquons enfin que la définition de la carte topologique n'utilise pas la notion de frontière. Ce point est très important, car c'est ce qui rend cette définition simple. Le seul lien entre la carte combinatoire et l'image segmentée en régions que nous voulons représenter, est effectué lors de la construction du niveau 1 à partir du niveau 0. En effet, pour construire ce niveau, nous fusionnons les faces adjacentes appartenant à la même région, information que nous trouvons dans l'image segmentée. La construction des autres niveaux n'est plus du tout liée à l'image, mais uniquement à la carte et aux différentes cellules de celle-ci. De ce fait, les notions de frontières interpixel et de courbes frontières ne sont pas nécessaires pour définir la carte topologique, mais uniquement pour prouver qu'elle représente correctement les informations de l'image. Ceci est important pour l'extension de notre modèle en dimension supérieure, et permettra de définir la carte topologique sans se préoccuper de la définition des surfaces frontières dans un espace de dimension 3.

4.3 Modèles de plongement et niveaux de simplification

Les cartes combinatoires codent la topologie des objets qu'elles représentent par subdivision de l'espace qu'ils occupent. Mais dans la plupart des applications, représenter uniquement la topologie ne suffit pas, il faut également représenter la géométrie des objets modélisés. Afin d'associer un modèle géométrique à notre modèle topologique, il est possible d'associer à chaque cellule topologique un objet géométrique de même dimension.

Mais dans la plupart du temps, pour des applications spécifiques, l'existence de liens entre le modèle topologique et le modèle géométrique rend inutile le plongement de chaque cellule. Nous étudions ici quelques modèles de plongements, pour chaque niveau de carte. Ces plongements sont les plus couramment utilisés, mais des applications spécifiques nécessiteront certainement un plongement particulier, qui pourrait alors s'inspirer ou être une composition de certains plongements présentés ici.

4.3.1 La carte lignel

Dans cette carte, chaque brin représente un lignel des frontières interpixel de l'image. Un premier plongement, qui découle immédiatement de cette propriété, consiste à associer à chaque brin le code de Crack[2] [BF70] correspondant. Il suffit alors de conserver les coordonnées du pointel en haut à gauche de chaque composante connexe de la carte pour pouvoir retrouver les coordonnées de chaque pointel, et pouvoir ainsi reconstruire la géométrie des frontières. Le principal avantage de ce plongement est qu'il est très peu coûteux en espace. Il nécessite seulement 8 octets par composante connexe de la carte (pour les coordonnées du pointel en haut à gauche), ce qui est négligeable, plus 2 bits par brin pour le code de Crack. En effet, les lignels étant forcément horizontaux ou verticaux, 2 bits suffisent pour ce codage. L'inconvénient majeur de ce plongement est que pour récupérer les coordonnées d'un pointel particulier, nous devons parcourir, au pire des cas, toute la composante connexe afin de trouver les coordonnées du point de départ. Pour les opérations agissant sur toute la carte (comme l'affichage complet) ce n'est pas coûteux car nous pouvons commencer le parcours par ces points particuliers. Par contre pour des opérations locales (comme le calcul du périmètre d'une région) ce plongement entraîne un surcoût en temps rédhibitoire. En effet, dans ce cas il faut parcourir toute la composante connexe avant de pouvoir calculer ce périmètre.

Un autre plongement, qui contrairement au précédent privilégie le temps d'exécution au détriment de l'espace, consiste à associer à chaque sommet topologique de la carte lignel les coordonnées du pointel correspondant. Ce plongement est beaucoup plus coûteux en espace mémoire que le précédent, mais les opérations pourront s'effectuer de manière minimale en nombre de brins à parcourir. Suivant nos besoins particuliers, nous choisirons l'un ou l'autre plongement. Il est également envisageable d'utiliser un plongement hybride, où certains sommets topologiques seraient associés aux coordonnées des pointels correspondant, et chaque brin serait associé au code de Crack du lignel correspondant. Cette solution permettrait de diminuer la complexité moyenne des opérations, en augmentant légèrement l'espace nécessaire.

4.3.2 La carte des frontières

Pour cette carte, nous utilisons sa principale propriété afin de définir un plongement efficace, comme pour la carte lignel. Nous avons vu à la section précédente que chaque arête de cette carte correspond à la suite maximale de lignels alignés d'une frontière. Cela permet de ne pas plonger les arêtes. En effet, pour chaque arête de la carte, nous pouvons retrouver le plongement de ses deux sommets extrémités, et reconstruire le plongement de cette arête qui est simplement le segment de droite défini par ces deux sommets. Pour ce premier plongement, nous associons simplement à chaque sommet de la carte des frontières les coordonnées du pointel correspondant. Ce plongement est peu coûteux en espace mémoire

[2] Le code de Crack est la restriction du code de Freeman à seulement quatre directions afin de représenter les contours interpixel d'une région.

en comparaison à l'espace utilisé pour le codage de la carte elle-même. En effet le nombre de sommets est en moyenne égal à un tiers du nombre de brins [BF97], et le codage des coordonnées d'un sommet demande moins d'espace que le codage d'un brin. De plus, il est simple à mettre en œuvre étant donné que nous avons un seul type de cellule à plonger. Enfin, retrouver le plongement des arêtes ou des faces se fait sans aucune difficulté ni surcoût en complexité, en parcourant les brins et en récupérant les coordonnées de chaque sommet rencontré.

La simplicité de ce plongement, et le fait qu'il est naturel[3] font qu'il est souvent utilisé. Un autre plongement moins coûteux en espace consisterait à conserver les coordonnées du pointel en haut à gauche de chaque composante connexe, et à associer à chaque brin une direction et une longueur. Mais le gain en espace mémoire n'est pas très important, alors que la complexité des opérations augmente, comme nous l'avons vu pour la carte lignel.

4.3.3 La carte topologique

Pour ce dernier niveau, nous ne pouvons plus retrouver le plongement d'une arête à partir du plongement de ses sommets extrémités. En effet, les arêtes de cette carte topologique peuvent maintenant avoir un plongement quelconque et ne sont plus uniquement des segments de droite.

Un premier plongement consiste à associer à chaque arête la courbe de dimension 1 représentant la frontière interpixel correspondante. Ce plongement a comme principal avantage d'être simple car, comme pour la carte des frontières, un seul type de cellule est plongé. Nous avons représenté figure 4.10 la carte topologique de notre exemple plongée de cette manière. Sur cette figure, nous avons numéroté les brins de 1 à 20. Nous associons à chaque arête une courbe de dimension 1 qui représente le plongement de la frontière correspondante. Nous avons représenté ces courbes par une liste de sommets, qui peuvent être codés de plusieurs manières différentes. Une solution consiste à les représenter par des 1-cartes, la structure obtenue est alors hiérarchique. Dans ce cas, ces courbes de dimension 1 doivent être orientées dans le même sens que les brins qui les désignent, afin de pouvoir, par exemple, reconstruire correctement le plongement d'une face en récupérant la suite des plongements des arêtes qui la compose.

Pour représenter l'image de la figure 4.3, chaque sommet doit être associé aux coordonnées du pointel correspondant dans l'image. Mais plusieurs sommets ont les mêmes coordonnées. Ce sont les sommets géométriques associés au même sommet topologique de la carte. Par exemple, les brins $\{2, 9, 19\}$ représentent un sommet topologique de la carte. Les courbes 1d associées à chaque arête incidente à un de ces brins auront toutes les coordonnées de ce sommet en commun. Cela se vérifie sur notre exemple, figure 4.10, où les sommets s_1, s_{11} et s_{31} ont tous les mêmes coordonnées puisqu'ils représentent le même pointel.

De manière générale, les sommets extrémités de chaque courbe frontière sont dupliqués autant de fois qu'il y a d'arêtes incidentes à ce sommet dans la carte topologique. Cette redondance est non seulement coûteuse en espace, mais également en temps d'exécution

[3] Quelqu'un ne connaissant pas les cartes combinatoires dessinera plus naturellement les arêtes comme des segments de droite que comme des courbes quelconques.

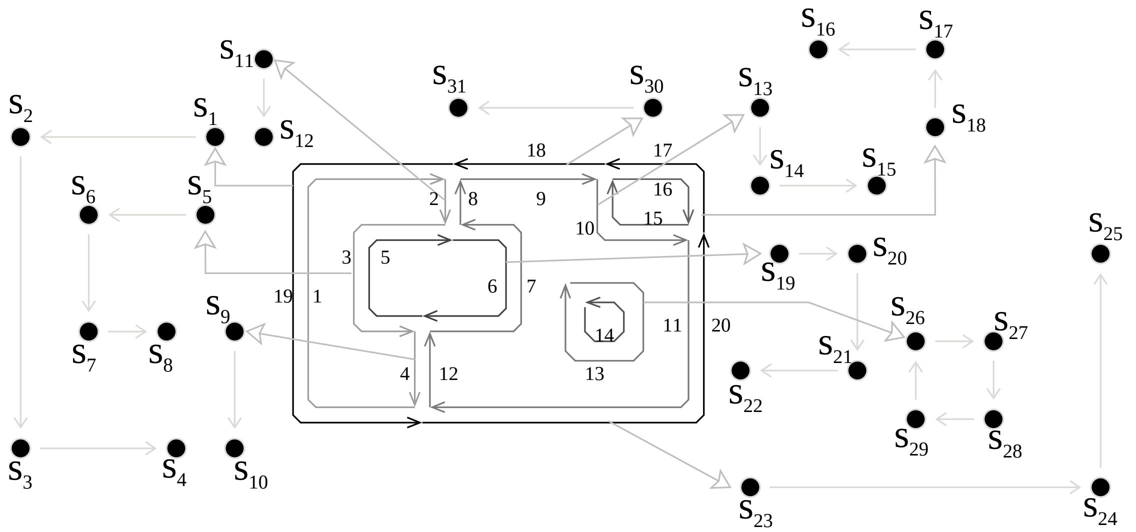


Fig. 4.10 – La carte topologique et son plongement arête.

lors des mises à jour des plongements. En effet, si nous voulons modifier les coordonnées d'un sommet, il faut le faire pour tous les plongements des arêtes incidentes à ce sommet.

Afin d'éviter cette duplication, nous définissons un deuxième type de plongement pour lequel nous plongeons les sommets ainsi que les arêtes de la carte topologique (cf. figure 4.11). Chaque sommet sera plongé par les coordonnées du pointel correspondant, et chaque arête par la courbe de dimension 1 correspondant à la frontière interpixel associée, moins ses extrémités. Nous parlons abusivement de *courbe ouverte* bien que ce ne soit pas exactement une courbe ouverte au sens topologique du terme. L'avantage de ce plongement est de ne pas coder de manière redondante les coordonnées des sommets extrémités des courbes frontières. Son inconvénient est que lorsque nous voulons récupérer le plongement d'une courbe frontière, il faut le reconstruire en utilisant les plongements des sommets incidents à l'arête correspondant à cette courbe frontière et le plongement de l'arête. Mais cette reconstruction peut s'effectuer en temps linéaire et n'est donc pas pénalisante par rapport à la carte des frontières.

Ce plongement est présenté figure 4.11 sur notre image d'exemple. Cette carte a sept sommets topologiques (plongés par les sommets s_1, s_4, s_5, s_8, s_{11}, s_{13} et s_{15}), et dix arêtes topologiques dont trois n'ont pas de plongement. En effet, lorsque le plongement d'une arête est un segment de droite, comme c'est le cas sur notre exemple pour les trois arêtes $\{2, 8\}$, $\{4, 12\}$ et $\{9, 18\}$, la courbe 1d correspondante privée de ses extrémités ne contient plus aucun sommet. Pour reconstruire le plongement d'une arête incidente à un brin b, il suffit de récupérer, dans un premier temps, le plongement du sommet incident à ce brin, puis le plongement de l'arête, et enfin le plongement du sommet incident au brin $\beta_2(b)$. La courbe 1d constituée de la concaténation, dans cet ordre, des sommets obtenus est le plongement de l'arête.

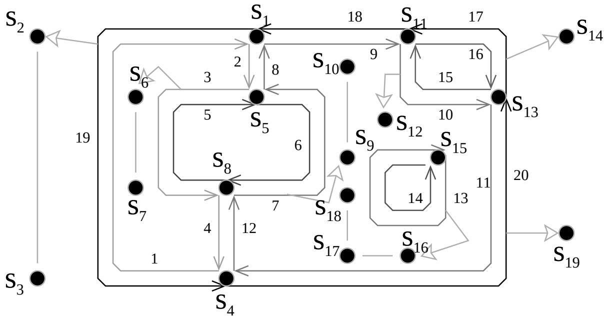


FIG. 4.11 – La carte topologique et son plongement sommet et arête « ouverte ».

Sur notre exemple, afin de récupérer le plongement de l'arête incidente au brin 19, nous récupérons le plongement du sommet incident à ce brin, c'est s_1, puis les sommets du plongement arête, s_2 et s_3, et enfin le plongement du sommet incident au brin 1 (qui est $\beta_2(19)$), c'est s_4. Le plongement de l'arête est donc la courbe 1d constituée des sommets $[s_1, s_2, s_3, s_4]$. Comme pour le plongement précédent, les courbes de dimension 1 doivent être orientées correctement, c'est-à-dire dans le même sens que le brin « pointant » sur cette courbe, afin que la reconstruction soit valide. Ce plongement est moins coûteux en espace mémoire que le précédent, et la complexité en temps des opérations est identique, voire moins coûteuse. Par exemple l'opération de modification des coordonnées d'un sommet de la carte est, pour ce plongement, moins coûteuse que pour le plongement précédent. C'est ce plongement que nous utilisons par la suite, dans les différents algorithmes travaillant sur la carte topologique.

4.4 Un premier algorithme d'extraction

Après avoir défini les trois niveaux de carte combinatoire et les plongements qui leur sont associés, nous allons nous intéresser à la manière de construire ces diverses cartes à partir d'une image segmentée en régions.

Nous présentons ici un premier algorithme « naïf » d'extraction. Cet algorithme a comme principal avantage d'être simple, puisqu'il découle immédiatement des définitions de nos différents niveaux de carte. En effet, ces définitions sont algorithmiques, et définissent chaque niveau de carte à partir du niveau précédent et d'un type particulier de fusion. Pour définir totalement cet algorithme, nous devons donc nous intéresser aux fusions dans les cartes combinatoires.

4.4.1 Les fusions dans les 2-cartes

Une fusion en dimension i consiste à modifier deux objets adjacents de dimension i pour en obtenir un seul constitué de l'union des deux objets. Nous pouvons voir deux exemples de fusion figure 4.12. Pour la dimension n, il existe exactement n fusions possibles qui sont

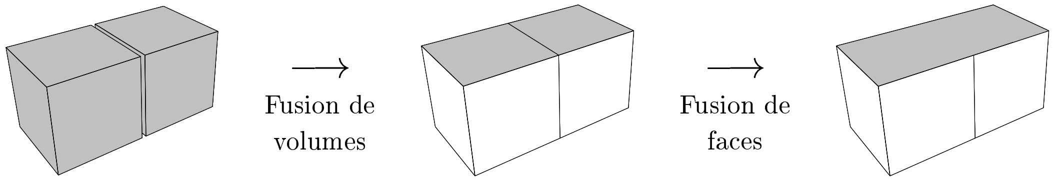
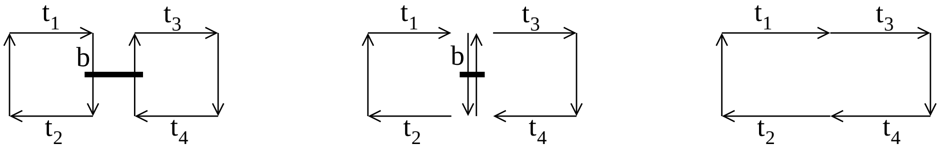


FIG. 4.12 – Deux exemples de fusions.



a. Avant la fusion.

b. β_0 et β_1 décousures des brins centraux.

c. β_1 couture des brins décousus.

FIG. 4.13 – Fusion de faces en 2d.

les fusions des dimensions de $1 \dots n$. Ces opérations de fusions sont simples à écrire dans le cadre des cartes combinatoires, et ce pour n'importe quelle dimension. Afin que cette opération de fusion puisse s'effectuer correctement, la configuration des cellules que nous voulons fusionner est contrainte.

Lorsque nous voulons fusionner deux i-cellules adjacentes, C_1 et C_2, nous parlons de fusion le long de la $(i-1)$-cellule incidente à C_1 et C_2 que nous appelons ici C. Pour que la fusion puisse être appliquée, C doit être de degré 1 ou 2. Intuitivement, la fusion consiste à supprimer cette cellule. Lors de cette suppression, les deux cellules C_1 et C_2 sont modifiées pour n'en faire plus qu'une, mais s'il existe d'autres cellules incidentes à C, leurs relations d'incidences ne seront alors plus valides. Le cas où C est de degré 1, revient en dimension 2 à effectuer la fusion d'une face le long d'un isthme (figure 4.14).

Remarquons que nous considérons ici la fusion de « base », consistant à fusionner deux cellules C_1 et C_2 uniquement le long d'une seule cellule. Il existe des fusions plus évoluées, permettant de fusionner deux cellules qui possèdent des multi-adjacences en une seule opération. Mais cette fusion de haut niveau peut être réalisée par un ensemble de fusions simples, et n'est pas utile ici.

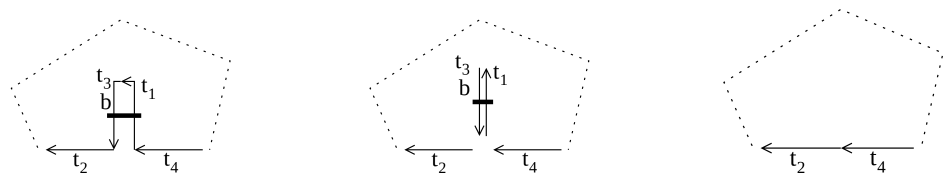
En dimension 2, il existe deux types différents de fusions qui sont la fusion de faces (dimension 2) et la fusion d'arêtes (dimension 1). La figure 4.13 illustre le déroulement de l'algorithme 1 effectuant la fusion de deux faces.

Cet algorithme prend en entrée un brin b, et fusionne la face incidente à ce brin avec la face incidente au brin $\beta_2(b)$ (c'est pour cette raison que b doit nécessairement être cousu par β_2). Il découd l'arête incidente au brin b par β_0 et β_1 (figure 4.13.b), puis coud les brins initialement cousus à cette arête (figure 4.13.c). Remarquons qu'aucun traitement ne met à jour les éventuels plongements car les sommets topologiques conservent les mêmes coordonnées avant et après la fusion, et les éventuels plongements arêtes ne sont pas modifiés.

Algorithme 1 Fusion de faces en 2d

Entrée : Un brin b β_2 cousu
Résultat : Les deux faces incidentes à b et à $\beta_2(b)$ sont fusionnées.

$t_1 \leftarrow \beta_0(b)$; $t_2 \leftarrow \beta_1(b)$;
$t_3 \leftarrow \beta_{21}(b)$; $t_4 \leftarrow \beta_{20}(b)$;
β_0-découdre(b) ; β_1-découdre(b);
β_0-découdre($\beta_2(b)$) ; β_1-découdre($\beta_2(b)$);
β_1-coudre(t_1, t_3) ; β_1-coudre(t_4, t_2);



a. Avant la fusion. b. β_0 et β_1 décousures des brins centraux. c. β_1 couture des brins décousus.

FIG. 4.14 – La fusion de faces en 2d pour un cas dégénéré.

La seule modification éventuelle du plongement est réalisée de manière transparente, lors des coutures et décousures. En effet, chaque sommet topologique doit être associé à exactement un sommet géométrique. L'opération de décousure duplique un sommet géométrique si elle entraîne la déconnexion d'un sommet topologique, et réciproquement l'opération de couture détruit un sommet géométrique si elle regroupe deux sommets topologiques distincts. Dans ce dernier cas, afin de pouvoir contrôler quel est le sommet détruit, nous fixons par convention que c'est toujours le sommet appartenant au sommet topologique du deuxième brin passé en paramètre de la fonction de couture.

Nous pouvons remarquer que cet algorithme effectue correctement la fusion de faces quelles que soient leurs configurations, même dans les cas « dégénérés », par exemple pour la configuration présentée figure 4.14. Cet exemple montre la fusion de faces le long d'une arête de degré un (car $\beta_0(b) = \beta_2(b)$). Nous pouvons vérifier que même dans ce cas la fusion de faces par l'algorithme 1 ne pose aucun problème. En effet, les coutures et décousures s'effectuent uniquement de manière locale, sans aucune contrainte sur la manière dont les brins doivent être cousus, à condition que les brins b et $\beta_2(b)$ soient différents.

Comme pour la fusion de faces, la fusion d'arêtes incidentes à un sommet de degré deux se fait de manière simple comme nous pouvons le voir sur l'algorithme 2.

La figure 4.15.a présente un exemple de carte avant la fusion d'arêtes, et le résultat de cette fusion figure 4.15.b. Comme pour la fusion de faces, l'algorithme de fusion d'arêtes fonctionne quelle que soit la configuration des brins, et quelle que soit la géométrie des arêtes.

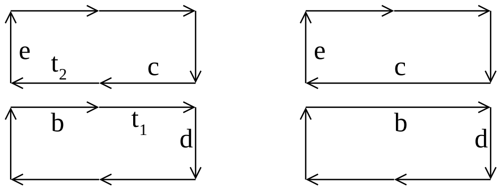
Le fait de conserver le brin b ainsi que le brin $c = \beta_{20}(b)$ pour la nouvelle arête, évite toute modification de plongement sommet. En effet, chaque sommet topologique conserve

Algorithme 2 Fusion d'arêtes en 2d

Entrée : Un brin b β_1 cousu

Résultat : Les deux arêtes incidentes à b et à $\beta_1(b)$ sont fusionnées.

$t_1 \leftarrow \beta_1(b)$; $t_2 \leftarrow \beta_2(b)$;
$c \leftarrow \beta_2(t_1)$; $d \leftarrow \beta_1(t_1)$; $e \leftarrow \beta_1(t_2)$;
β_0-découdre(t_1) ; β_1-découdre(t_1) ; β_2-découdre(t_1);
β_0-découdre(t_2) ; β_1-découdre(t_2) ; β_2-découdre(t_2);
β_1-coudre(b, d) ; β_1-coudre(c, e) ; β_2-coudre(b, c);



a. Avant la fusion. b. Après la fusion.

FIG. 4.15 – Fusion d'arêtes en 2d.

le même plongement avant et après la fusion car nous ne modifions pas l'orbite des deux sommets incidents à ces deux brins, nous supprimons seulement l'orbite du sommet à fusionner. Dans le cas où nous avons un plongement arête, nous devons distinguer deux cas différents. Si les arêtes sont alignées, il n'y a pas de modification à faire, car le plongement de l'arête est simplement le segment de droite reliant les deux plongements sommets extrémités de l'arête. Par contre, si les deux arêtes ne sont pas alignées, nous devons modifier le plongement arête afin de conserver toute la géométrie de la frontière. Dans ce cas, il suffit d'ajouter les coordonnées du sommet supprimé en fin du plongement arête du brin b. De plus, l'opération de couture par β_2 devra gérer le plongement arête, afin de fusionner éventuellement les deux plongements des deux brins cousus. Ces opérations de mise à jour du plongement se définissent simplement, et permettent ensuite d'oublier totalement le plongement qui sera mis à jour de manière transparente par les opérations de base sur la carte.

4.4.2 L'algorithme d'extraction naïf

Les définitions des différents niveaux de carte, présentées section 4.2, nous donnent immédiatement l'algorithme 3 d'extraction. Cet algorithme prend en entrée une image segmentée en région, et le niveau de simplification souhaité (un entier entre 1 et 3), et retourne la carte correspondante.

La première ligne de cet algorithme construit la carte de niveau 0. Pour obtenir la carte de niveau 1, nous devons, d'après la définition 16, fusionner chaque couple de faces adjacentes et appartenant à la même région. Nous utilisons pour cela l'opération de fusion présentée section 4.4.1. Cette fusion de faces s'effectue uniquement le long d'une arête. Nous devons éventuellement fusionner plusieurs fois le même couple de faces, mais le long

Algorithme 3 Extraction « naïve » de la carte de niveau n en 2d

Entrée : Une image I segmentée en régions de $n_1 \times n_2$ pixels
n le niveau de la carte que nous voulons extraire
Sortie : La carte de niveau n de l'image I.

```
1 Construire la carte de niveau 0 correspondant à I;
2 Fusionner tout couple de faces adjacentes appartenant à la même région;
  si n ≥ 2 alors
3     Fusionner tout couple d'arêtes alignées incidentes à un sommet de degré deux;
      si n = 3 alors
4         Fusionner tout couple d'arêtes incidentes à un sommet de degré deux;
5 Calculer l'arbre d'inclusion des régions;
```

d'arêtes différentes. De plus, nous devons également fusionner une face avec elle-même, afin de supprimer les éventuels isthmes. Mais cette opération s'effectue, malgré cela, en complexité linéaire en le nombre de brins de la carte. En effet, nous parcourons l'ensemble des arêtes de la carte, et pour chacune testons si les deux faces qui lui sont incidentes sont de même région. Si c'est le cas nous effectuons leur fusion, et sinon nous savons que cette fusion de faces ne sera jamais réalisée. Cette technique nous permet de fusionner, lorsque cela est nécessaire, plusieurs fois les mêmes faces le long de différentes arêtes, mais également de fusionner une face avec elle-même. Avec cette technique, nous parcourons exactement une fois chaque arête de la carte, d'où la complexité linéaire en le nombre de brins (car le nombre d'arêtes dans une 2-carte fermée est égal à deux fois le nombre de brins).

Ensuite, si le niveau désiré est le niveau 2 ou 3, nous devons d'après la définition 17, fusionner chaque couple d'arêtes alignées incidentes à un sommet de degré deux pour obtenir la carte de niveau 2. Enfin, si la carte demandée est celle de niveau 3, nous fusionnons chaque couple d'arêtes incidentes à un sommet de degré deux d'après la définition 18. Ces deux types de fusions d'arêtes ne peuvent pas être effectués entre deux fois la même arête, contrairement à la fusion de faces. En effet, dans ce cas le sommet est de degré un, et nous ne devons alors pas effectuer de fusion. Afin de réaliser ces deux étapes, nous utilisons le même principe que pour la fusion de faces. Nous parcourons l'ensemble des sommets de la carte, testons pour chacun s'il est de degré deux, et effectuons la fusion si c'est le cas. La complexité est donc à nouveau linéaire en le nombre de brins de la carte.

Nous pouvons remarquer que pour l'extraction de la carte de niveau 3, l'étape 3 pourrait être supprimée, car les arêtes fusionnées lors de cette étape le seraient lors de l'étape 4. Mais conserver cette étape permet d'avoir un parallèle total entre l'algorithme et les divers niveaux de simplification. La carte de niveau 0 est construite au début de l'algorithme, puis chaque nouvelle étape construit le niveau suivant de simplification.

Cet algorithme ne présente pas les opérations concernant les modifications du plongement, car elles sont faites de manière transparente lors de l'opération de fusion, de la manière déjà présentée section précédente. Sa complexité globale est linéaire en le nombre de brins de la carte, car nous venons de voir que c'est le cas pour le calcul de chacun des niveaux, et nous verrons section 4.4.3 que c'est également le cas de l'algorithme de calcul de l'arbre d'inclusion.

4.4.3 Calcul de l'arbre d'inclusion

Le calcul de l'arbre d'inclusion s'effectue en dernière étape de notre algorithme d'extraction, après avoir extrait la carte. Cet arbre est nécessaire afin de conserver les liens entre les diverses composantes connexes de la carte, et pour les positionner les unes par rapport aux autres. Cet algorithme est générique et fonctionne sur les 3 niveaux de carte.

La notion d'inclusion utilisée ici est celle présentée définition 12 section 4.1.1. D'après cette définition, une région est incluse dans une ou plusieurs régions. Mais il n'est pas nécessaire de coder, pour une région, l'ensemble des régions dans laquelle elle est incluse car ces régions se retrouvent sans difficulté grâce à la propriété de transitivité de la relation d'inclusion.

Nous représentons donc uniquement la relation d'inclusion directe (définition 13) au moyen d'un *arbre d'inclusion*, un nœud associé à une région R_i est fils d'un nœud associé à une région R_j si et seulement si R_i est directement incluse dans R_j. Cet arbre aura toujours pour racine la région infinie. Si nous voulons retrouver, pour une région R_i, toutes les régions dans laquelle R_i est incluse, il suffit de parcourir l'arbre d'inclusion en partant du nœud associé à R_i et de remonter jusqu'à la racine. Chaque nœud rencontré lors de ce parcours est associé à une région dans laquelle R_i est incluse.

Pour construire cet arbre d'inclusion, nous parcourons l'ensemble des brins de la carte, composante connexe par composante connexe, ce qui permet de retrouver les diverses inclusions. Mais pour cela, nous avons besoin d'un certain nombre d'informations que nous utilisons lors de cette étape :

- chaque brin doit connaître sa région d'appartenance ;
- chaque région r connaît un des brins de la carte associée à cette région. Nous appelons un tel brin le *représentant* de sa région. Ce brin est soit le brin ayant comme plongement sommet le pointel le plus en haut et à gauche de r, soit si un tel brin n'existe pas, le brin ayant ce même pointel dans son plongement arête ;
- nous connaissons la liste de toutes les régions de l'image, sans la région infinie. Cette liste doit être ordonnée de sorte qu'une région R_i est inférieure à une autre région R_j si et seulement si R_i se rencontre avant R_j lorsque nous parcourons l'image de haut en bas et de gauche à droite.

Ces trois propriétés sont assurées par nos algorithmes d'extraction. La première lors de la construction de la carte de niveau 0, la deuxième également lors de cette construction, puis en nous assurant de la conserver lors de chaque fusion, et enfin la dernière par un parcours préalable de l'image, en classant les régions rencontrées selon cet ordre. Ce sont ces propriétés qui nous assurent que la région contenant la composante connexe en cours de traitement est bien la région *parent*.

L'algorithme 4 calcule l'arbre d'inclusion d'une carte représentant les frontières d'une image segmentée en régions. Il prend comme paramètres d'entrée une carte, qui peut être de n'importe quel niveau, et la liste des régions de l'image, triée suivant l'ordre présenté ci-dessus.

Algorithme 4 Calcul de l'arbre d'inclusion en 2d

Entrée : La carte C d'une image I segmentée en régions
La liste L des régions de l'image I
Sortie : L'arbre d'inclusion $\mathcal{A}$ correspondant.

Démarquer chaque région de L;
Ajouter un nœud associé à R_0 dans $\mathcal{A}$;
1 **pour chaque** *région* R_i *de* L *non marquée* **faire**
  $b_{init} \leftarrow représentant(R_i)$;
  $parent \leftarrow$ le nœud associé à $région(\beta_2(b_{init}))$;
2 **pour chaque** *brin* b *de la composante connexe incidente à* b_{init} **faire**
    si $région(b)$ *non marquée* **alors**
      Ajouter un nœud associé à $région(b)$ dans $\mathcal{A}$;
      Ajouter ce nœud comme fils de *parent* dans $\mathcal{A}$;
      Marquer $région(b)$;
retourner $\mathcal{A}$

Cet algorithme se décompose en deux boucles. La première parcourt l'ensemble des régions de la liste L non marquées. En effet, quand une région est marquée, c'est qu'elle a déjà été traitée, ainsi que toute la composante connexe incidente à cette région. Pour une région non marquée, nous initialisons le brin de départ du parcours de la deuxième boucle comme étant le représentant de cette région. La région dans laquelle la composante connexe incidente à ce brin est incluse est forcément la région du brin cousu par β_2 au brin représentant de sa région. Cela est dû au fait que la liste des régions L est ordonnée de telle manière qu'une région R_i apparaît avant une autre région R_j si et seulement si R_i se rencontre avant R_j lorsque nous parcourons l'image de haut en bas et de gauche à droite, mais également à la contrainte fixée sur les brins représentants.

De par notre ordre de parcours particulier de la liste L, nous savons, lorsque nous atteignons une région non marquée R_i, qu'elle est forcément incluse dans une autre région déjà traitée, et qu'à sa gauche et en-dessus de cette région R_i se trouve cette région dans laquelle elle est incluse. Supposons que R_i ne soit pas incluse dans une région R_j déjà traitée. Cela signifierait que R_j ne possède aucun pixel à gauche et au-dessus de R_i, ce qui est en contradiction avec la définition de l'inclusion. Pour cette même raison, nous sommes certains qu'à gauche et au-dessus de R_i se trouvent uniquement des pixels de R_j. En effet, si nous trouvons à gauche ou en-dessus de R_i des pixels appartenant à une autre région que R_j, cela voudrait dire que cette région serait avant R_i dans la liste. Comme cette région appartiendrait à la même composante connexe que R_i, elle ne serait donc pas traitée, sinon nous aurions également traité R_i. Cela est en contradiction avec le fait que nous parcourons la liste L en respectant son ordre, et en traitant les premières régions rencontrées.

La boucle 2 consiste à parcourir chaque brin de la composante connexe (dans la carte C) incidente à un brin de départ particulier b_{init}, qui est le représentant de la région en cours de traitement. Avant cette boucle, nous avons donc initialisé ce brin de départ, ainsi que le nœud associé à la région dans laquelle la composante connexe incidente à ce brin est incluse : c'est *parent*. Au cours de ce parcours, chaque nouvelle région rencontrée est une région directement incluse dans la région associée à *parent*. Nous associons donc à cette région un nœud dans l'arbre d'inclusion qui est un fils du nœud *parent*. Comme nous parcourons tous les brins de la composante connexe incidente à b_{init}, nous traitons donc toutes les régions incidentes (mais pas incluses) à la région contenant b_{init}.

La complexité de cet algorithme est linéaire en le nombre de brins de la carte et en le nombre de régions de l'image (en pratique, le nombre de régions est très inférieur au nombre de brins). En effet, nous parcourons chaque brin exactement une fois, car nous parcourons les composantes connexes une par une, deux composantes connexes différentes étant forcément disjointes. Nous parcourons également chaque région une seule fois car nous parcourons la liste L. Cet algorithme de calcul fonctionne pour n'importe quel niveau de carte. De plus, son exécution est plus rapide lorsque le niveau augmente, même si la complexité ne change pas, car le nombre de brins diminue de plus en plus.

L'algorithme naïf d'extraction d'une carte à partir d'une image est l'application directe de la définition des différents niveaux de carte. Il est simple à comprendre, et facilement implantable. Mais il a comme inconvénient principal de créer au départ beaucoup de brins, pour en détruire ensuite un grand nombre lors des différentes fusions. Cela pose deux problèmes majeurs. Tout d'abord le temps nécessaire à la création et à la destruction de nombreux brins est important et inutile. Ensuite, l'espace nécessaire à la construction du niveau 0 peut être trop important pour pouvoir construire cette carte, alors que la carte topologique demande un espace mémoire beaucoup moins important et devrait donc pouvoir être calculée aisément. De plus, après avoir créé la carte de niveau 0, il faut parcourir chaque brin n fois pour obtenir la carte de niveau n. Bien que cela ne modifie pas l'ordre de complexité, cela entraîne quand même une perte de temps d'exécution. Nous présentons donc une amélioration de cet algorithme naïf, utilisant la notion de précode déjà utilisée par [Cha95, Fio95, Fio96, BFP99, BDF00]. Nous allons montrer qu'il y a un parallèle entre le niveau de simplification que nous voulons extraire et le nombre de précodes à traiter. De plus, notre approche particulière, basée sur nos différents niveaux de simplification, nous permet de factoriser au maximum les précodes pouvant se traiter de manière identique.

4.5 Les précodes pour définir un algorithme optimal d'extraction

L'idée de Christophe Fiorio [Fio95, Fio96] était de ne pas construire, comme pour notre algorithme d'extraction naïf, toute la carte de niveau 0 pour la simplifier ensuite. Son algorithme d'extraction effectue un balayage de l'image de haut en bas et de gauche à droite, avec une fenêtre de 2×2 pixels, en exécutant un traitement dépendant de la configuration locale des pixels de cette fenêtre.

Nous avons étendu cette idée, pour l'adapter à nos divers niveaux de carte, et nous avons étudié précisément les configurations à traiter pour chacun de ces niveaux. L'idée générale de notre démarche est de calculer la carte de manière incrémentale et directe, au moyen d'un seul balayage de l'image. De plus, les traitements à effectuer vont dépendre uniquement de la configuration locale des pixels, et vont créer exactement le nombre de

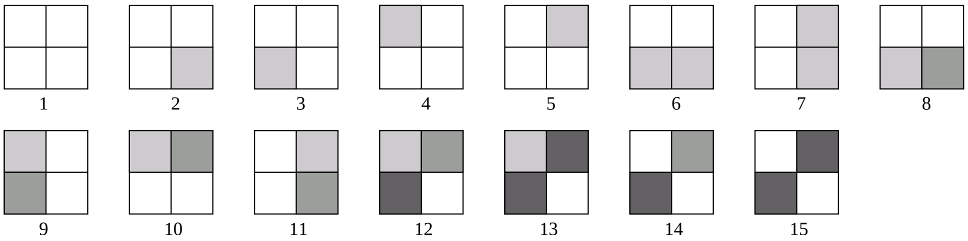


FIG. 4.16 – Les 15 précodes de la dimension 2.

brins nécessaires. L'originalité de ce travail est principalement la factorisation des précodes pouvant se traiter de manière similaire. Cette factorisation est facilitée par l'introduction des niveaux de simplification. Nous obtenons alors des « classes » de précodes en fonction du niveau de simplification souhaité. Mais avant de définir ces classes et de donner notre algorithme d'extraction, nous devons dans un premier temps définir la notion de précode.

4.5.1 Les précodes

Nous balayons l'image de haut en bas et de gauche à droite. Au cours de ce balayage, nous appelons *pixel courant* le pixel en cours de traitement, donc le dernier rencontré au cours de notre balayage, et *précode* une configuration locale de pixels.

Définition 19 (précode) *Un* précode *en dimension n est une partition de l'ensemble des 2^n n-cellules de l'hypercube de longueur* 2.

Un précode est donc une partition $\{p_1, \ldots, p_k\}$ d'un ensemble de cellules. Chaque p_i est l'ensemble des cellules appartenant à la même région. Lorsque $p_i \neq p_j$, alors les cellules de p_i appartiennent à une région différente des cellules de p_j.

Pour la dimension 2, un précode est une fenêtre de 2×2 pixels, le pixel courant étant celui en bas à droite de cette fenêtre. Nous pouvons voir figure 4.16 les précodes de dimension 2. Nous représentons un précode en affectant une couleur différente à chaque élément de la partition. Deux pixels de même couleur indiquent qu'ils appartiennent à la même région, et réciproquement.

Mais il faut effectuer le lien entre les précodes et les images segmentées en régions. Pour cela, étant donné un précode, nous associons une région différente à chaque p_i de la partition de ce précode. Nous obtenons alors une *configuration*, que nous dirons *couverte* par ce précode. Un précode couvre plusieurs configurations. En effet, pour un précode $\{p_1, \ldots, p_k\}$, si nous avons r régions distinctes dans l'image, nous pouvons affecter r régions à p_1, $r-1$ régions à p_2, …, $r-k+1$ à p_k. Le nombre de configurations couvertes par un précode est donc $A_r^k = \frac{r!}{(r-k)!}$.

Un précode en dimension n étant une partition d'un ensemble de 2^n n-cellules, nous pouvons donc calculer le nombre de précodes distincts qui est le nombre de manières de partitionner les 2^n cellules. Soit N_n le nombre de précodes à la dimension n, nous pouvons calculer N_n grâce aux nombres S_l^k, appelés nombres de Stirling de première espèce. S_l^k donne le nombre de surjections d'un ensemble de l éléments dans un ensemble à k éléments. $\frac{S_l^k}{k!}$ donne alors le nombre de partitions en k sous-ensembles d'un ensemble à l éléments. Dans notre cas, $l = 2^n$. Il suffit alors d'additionner les partitions à un ensemble, deux ensembles, ..., à 2^n ensembles pour obtenir le nombre de précodes. Nous avons donc : $N_n = \sum_{k=1}^{2^n} \frac{S_{2^n}^k}{k!}$. Sachant que $B_l = \sum_{k=1}^{l} \frac{S_l^k}{k!}$ où B_l sont les nombres de Bell donnés par la formule de récurrence suivante : $B_l = \sum_{i=0}^{l-1} \frac{B_i.l!}{i!(l-i)!}$ avec $B_0 = B_1 = 1$, nous pouvons alors calculer le nombre de précodes N_n pour la dimension n qui est B_{2^n}. La formule précédente permet de calculer qu'il y a 15 précodes différents pour la dimension 2, déjà présentés figure 4.16.

Définition 20 (précode non variété) *On dit qu'un précode de dimension n est* non variété *si et seulement s'il existe dans ce précode une partition n'étant pas $2n$-connexe. Nous appelons précode* variété *un précode n'étant pas non variété.*

Pour la dimension 2, nous pouvons voir sur la figure 4.16 que les trois derniers précodes sont non variétés. La notion de précode est utile afin de définir les actions à effectuer lors de notre balayage de l'image. Mais nous avons besoin d'une autre notion permettant de regrouper plusieurs précodes : c'est la notion de *précode partiel.*

Définition 21 (précode partiel) *Un* précode partiel *en dimension n est une partition d'un sous-ensemble des cellules d'un précode, chaque ensemble de la partition devant être un ensemble $2n$-connexe.*

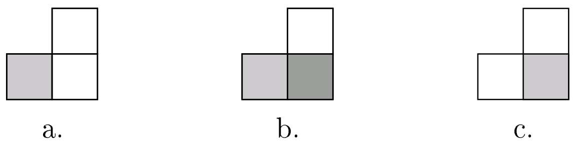


FIG. 4.17 – Deux exemples de précodes partiels (a et b) et un contre-exemple (c).

Nous pouvons voir figure 4.17 deux exemples de précodes partiels en dimension 2 (a et b) ainsi qu'un contre-exemple de précode partiel (c). Les pixels non représentés peuvent appartenir à n'importe quelle région. La partition associée au précode présenté figure 4.17.c n'est pas un précode partiel car un ensemble de cellules de la même région n'est pas 4-connexe.

L'interprétation d'un précode partiel n'est pas totalement identique à celle d'un précode. En effet, si nous appelons $\{p_1, \ldots, p_k\}$ la partition du précode partiel, chaque p_i contient les cellules étant de la même région. Mais contrairement aux précodes, deux p_i différents ne sont pas obligatoirement de deux régions différentes. En fait il y a deux cas distincts. Étant donné $p_i \neq p_j$:

- $p_i \cup p_j$ est $2n$-connexe. Alors la région associée à p_i est forcément différente de la région associée à p_j ;

– $p_i \cup p_j$ n'est pas $2n$-connexe. Alors la région associée à p_i peut être égale ou différente de la région associée à p_j.

De manière intuitive, cela veut dire que le précode partiel fixe l'appartenance ou non à la même région uniquement pour les régions voisines. Sur notre exemple, figure 4.17, le précode partiel b indique uniquement que le pixel inférieur droit n'est pas de la même région que son voisin de gauche, et pas de la même région que son voisin du haut. Mais il ne fixe aucune contrainte sur l'appartenance à la même région des pixels en bas à gauche et en haut à droite, car leur union n'est pas un ensemble 4-connexe. Par contre, les deux ensembles de pixels du précode partiel a ne peuvent pas être de la même région car leur union est un ensemble 4-connexe.

Les cellules n'étant dans aucun élément de la partition d'un précode partiel peuvent être de n'importe quelle région, sans aucune contrainte. Nous les appelons les *cellules libres*. Un précode partiel *couvre* plusieurs précodes : ce sont les précodes que l'on peut obtenir à partir du précode partiel en affectant les cellules libres dans des autres partitions ou dans des nouvelles partitions, et en faisant éventuellement des fusions respectant les contraintes déjà présentées.

Sur la figure 4.17, le précode partiel a couvre les précodes 3, 7 et 9 de la figure 4.16. Le précode partiel b couvre les précodes 2, 8, 11, 12, 13 et 14. Nous pouvons remarquer que ce précode couvre par exemple le précode 2 où les pixels en bas à gauche et en haut à droite sont de la même région, et couvre également le précode 12 où ces deux pixels sont dans des régions différentes.

4.5.2 Algorithme optimal et général d'extraction

L'algorithme général d'extraction d'une carte en une seule passe de l'image est simple puisque, comme nous pouvons voir sur l'algorithme 5, il se résume par un balayage de l'image en exécutant le code associé à chaque précode rencontré. Cet algorithme fonctionne pour tout type d'image : segmentées ou non, étiquetage fort ou faible[4]. En effet, la seule information utilisée ici est la notion de précode basée uniquement sur les configurations locales de pixels.

Algorithme 5 Extraction optimale de la carte de niveau n en 2d

Entrée : Une image I segmentée en régions de $n_1 \times n_2$ pixels
n le niveau de la carte que l'on veut extraire

Sortie : La carte de niveau n de l'image I.

```
last ← Construire le bord supérieur de la carte;
pour j = 1 à n2 + 1 faire
    pour i = 1 à n1 + 1 faire
        last ← Exécuter le code associé au précode (i, j) du niveau n;
Calculer l'arbre d'inclusion des régions;
```

[4]On parle d'étiquetage fort si chaque région possède une étiquette unique, et d'étiquetage faible si deux régions non connexes peuvent avoir une même étiquette.

Nous considérons ici que l'image est constituée des pixels dont les coordonnées vont de $1 \dots n_1$ en x, et de $1 \dots n_2$ en y, et qu'elle est entourée d'une région infinie. Le précode (i, j) est le précode ayant comme pixel courant (c'est le pixel en bas à droite de la fenêtre 2×2) le pixel (i, j).

La première étape de cet algorithme consiste à construire le bord supérieur de la carte. En effet au cours de cet algorithme, l'invariant est que la carte correspondant aux pixels plus anciens (pour notre ordre de parcours) a déjà été construite. Cet invariant garantit, à tout moment, qu'il existe un brin à gauche et au-dessus du pixel courant que l'on appelle respectivement *last* et *up* (cf. figure 4.18).

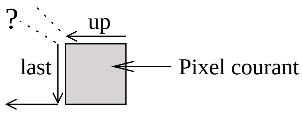


FIG. 4.18 – Les brins *last* et *up* par rapport au pixel courant.

Avant de commencer le parcours de l'image, le pixel courant est le pixel $(1, 1)$ et pour que cet invariant soit vérifié, nous devons avoir construit la carte pour les pixels de la région infinie à gauche ou au-dessus de ce pixel. C'est cette carte qui va être créée au cours de cette première étape détaillée plus précisément section 4.5.6.

Au cours de notre algorithme d'extraction, nous conservons le brin *last* qui permet de « raccrocher » le bout de la nouvelle carte à l'ancienne. Il nous suffit de conserver ce brin, car le brin *up* peut se calculer facilement à partir de ce brin, à l'aide de l'algorithme 6.

Algorithme 6 Calcul du brin *up* en 2d

Entrée : *last* le brin à gauche du pixel courant
Sortie : *up* le brin au-dessus du pixel courant.

$up \leftarrow \beta_0(last)$;
tant que up *est* β_2*-cousu* **faire**
  $up \leftarrow \beta_{20}(up)$)
retourner up;

Pour cet algorithme, nous partons du brin *last* et nous « tournons » autour du sommet incident à ce brin jusqu'à trouver un brin qui n'est pas β_2-cousu. Ce brin est alors le brin *up*, car tous les autres brins incidents à ce sommet sont forcément β_2-cousus. Nous pouvons voir figure 4.19 les quatre configurations possibles des brins *last* et *up*. Sur cette figure, nous avons représenté le pixel courant, et les différentes possibilités des cartes déjà construites. Ce sont les seules car la carte que nous construisons est associée aux frontières interpixel, ce qui limite les configurations possibles. Nous voyons sur cette figure que l'algorithme de calcul du brin *up* à partir du brin *last* effectue, dans le cas d qui est le pire des cas, deux tours dans la boucle « tant que ». Nous notons, dans les prochaines figures, l le brin *last* et u le brin *up* afin de condenser la notation.

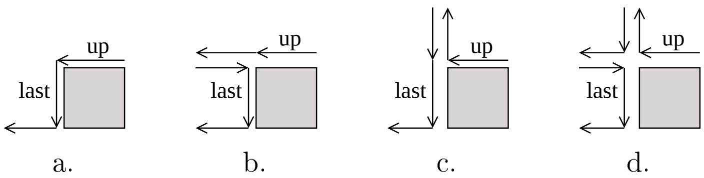


FIG. 4.19 – Les quatre configurations possible des brins *last* et *up*.

Après avoir créé le bord supérieur de l'image dans l'algorithme 5, le brin *last* est initialisé comme étant le brin vertical en haut à gauche de ce bord. L'algorithme d'extraction effectue ensuite un parcours de l'image de haut en bas et de gauche à droite, en exécutant chaque traitement associé au précode courant. Chacun de ces précodes modifie la carte localement, et retourne le brin vertical en bas à droite de ce précode, qui est alors affecté à *last* pour le prochain précode. Étant donné que l'image est plaquée sur un cylindre, ce traitement est valable pour tous les précodes, même ceux du bord. Nous consacrons la section 4.5.6 à la gestion du bord de l'image et à la manière dont il est utilisé lors de l'algorithme d'extraction.

Enfin, après avoir extrait la carte, il faut, comme pour l'algorithme naïf, construire l'arbre d'inclusion qui permet de faire le lien entre les éventuelles composantes connexes de la carte. Cet algorithme est l'algorithme 4 présenté section 4.4.3. Pour ce second algorithme d'extraction, le parcours préalable de l'image afin d'obtenir la liste des régions triées selon leur ordre d'apparitions n'est plus nécessaire. En effet, cette liste peut maintenant être créée durant le même parcours que celui construisant la carte.

Cet algorithme optimal d'extraction est générique. En effet, pour extraire un niveau particulier de carte, il faut définir quels précodes nous devons traiter, et comment nous allons les traiter. C'est simplement pour ces définitions qu'il y aura une distinction entre les différents niveaux. Nous étudions, dans un premier temps, les précodes nécessaires à l'extraction de la carte lignel.

4.5.3 Les précodes pour la carte lignel

Pour trouver les précodes à traiter afin d'extraire une carte spécifique, nous devons regarder quelles sont les fusions à effectuer afin d'obtenir cette carte, et étudier l'implication de ces fusions sur le traitement du précode local. En effet, étant donné que nous avons déjà construit la carte des pixels plus anciens, nous devons uniquement étudier le traitement à réaliser pour le pixel courant.

Pour la carte lignel, la seule fusion à effectuer est celle de faces adjacentes appartenant à la même région (cf. définition 16). Nous considérons donc uniquement les deux voisins à gauche et au-dessus du pixel courant. En effet, ce sont les deux seules fusions de faces possibles avec le pixel courant. Il faut étudier toutes les combinaisons de fusion ou non du pixel courant avec ces deux voisins. Nous obtenons alors 4 possibilités qui sont les 4 précodes présentés figure 4.20.

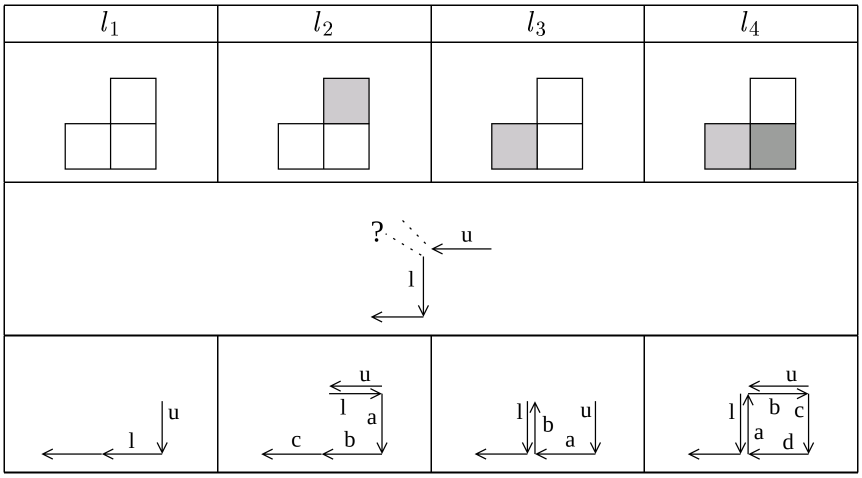


FIG. 4.20 – Les quatre précodes pour obtenir la carte lignel.

Sur cette figure, nous avons représenté sur la première ligne les précodes (que nous appelons par la suite *précodes lignels*), sur la deuxième ligne l'état de la carte correspondante avant de traiter le pixel courant, et sur la dernière ligne la carte à obtenir après avoir traité ce précode. Nous ne connaissons pas l'ensemble des brins présents dans la carte étant donné que nous ne savons pas la région d'appartenance du pixel libre dans les précodes partiels. Mais il est suffisant de connaître les brins *last* et *up* car les modifications à effectuer porteront uniquement sur ces brins. De même, les cartes obtenues après traitement des précodes sont des représentations partielles où nous montrons uniquement les brins modifiés.

Pour trouver le code associé à chaque précode, il suffit de regarder la carte avant le traitement de ce précode, de la comparer à la carte que nous voulons obtenir, et d'écrire la suite d'opérations permettant de transformer la première carte en la seconde. Nous pouvons voir sur les algorithmes 7, 8, 9 et 10 que ce code est simple à écrire. De plus, il est facile de vérifier sur la figure 4.20 que ces quatre algorithmes effectuent correctement les modifications des cartes de départ pour arriver aux cartes finales.

Algorithme 7 Code associé au précode l_1 en 2d

Entrée : *last* et *up*
(x, y) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$a \leftarrow \beta_0(last)$; $b \leftarrow \beta_1(up)$;
β_0-découdre$(last)$; β_1-découdre(up);
β_1-coudre$(up, last)$; β_1-coudre(a, b);
Le plongement du sommet incident à $last \leftarrow (x, y)$;
retourner up;

Pour ces algorithmes, ainsi que pour tous les suivants, nous considérons qu'à chaque sommet de la carte sont associées les coordonnées du pointel correspondant. Les fonctions de coutures et de décousures dupliquent ou fusionnent donc les plongements lorsque c'est nécessaire, comme nous l'avons présenté section 4.4. Le traitement d'un précode peut

Algorithme 8 Code associé au précode l_2 en 2d

Entrée : *last* et *up*
(x, y) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$a \leftarrow$ un nouveau brin ; $b \leftarrow$ un nouveau brin;
$c \leftarrow \beta_1(last)$; β_1-découdre$(last)$;
β_1-coudre(a, b) ; β_1-coudre(b, c);
β_1-coudre$(last, a)$; β_2-coudre$(up, last)$;
Le plongement du sommet incident à $b \leftarrow (x, y)$;
retourner a;

Algorithme 9 Code associé au précode l_3 en 2d

Entrée : *last* et *up*
(x, y) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$a \leftarrow$ un nouveau brin ; $b \leftarrow$ un nouveau brin;
$c \leftarrow \beta_1(up)$; β_1-découdre(up);
β_1-coudre(a, b) ; β_1-coudre(b, c);
β_1-coudre(up, a) ; β_2-coudre$(last, b)$;
Le plongement du sommet incident à $a \leftarrow (x, y)$;
retourner up;

s'écrire de plusieurs manières. Nous avons essayé d'être minimal en nombre d'opérations afin d'être optimal en temps d'exécution. Nous avons également organisé les coutures et décousures de façon à minimiser les modifications de plongements.

Nous pouvons vérifier que les quatre précodes lignels couvrent bien toutes les configurations que nous pouvons rencontrer dans l'image. En effet, les 15 précodes les couvrent bien toutes par définition, et les quatre précodes lignels couvrent bien ces 15 précodes comme nous pouvons le voir figure 4.21. Sur cette figure, un précode p_1 est fils d'un précode partiel p_2 lorsque p_1 est couvert par p_2. À l'aide de ces arbres, nous pouvons également vérifier que les traitements à effectuer pour deux précodes couverts par le même précode partiel sont identiques. C'est pour cette raison que nous n'avons pas besoin de définir 15 traitements différents mais simplement 4.

Algorithme 10 Code associé au précode l_4 en 2d

Entrée : *last* et *up*
(x, y) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$a \leftarrow$ un nouveau brin ; $b \leftarrow$ un nouveau brin;
$c \leftarrow$ un nouveau brin ; $d \leftarrow$ un nouveau brin;
β_1-coudre(a, b) ; β_1-coudre(b, c);
β_1-coudre(c, d) ; β_1-coudre(d, a);
β_2-coudre$(last, a)$; β_2-coudre(up, b);
Le plongement du sommet incident à $d \leftarrow (x, y)$;
retourner c;

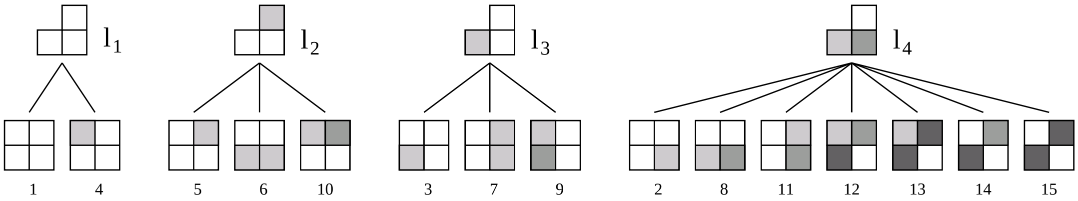


FIG. 4.21 – Les arbres de couverture des quatre précodes lignels.

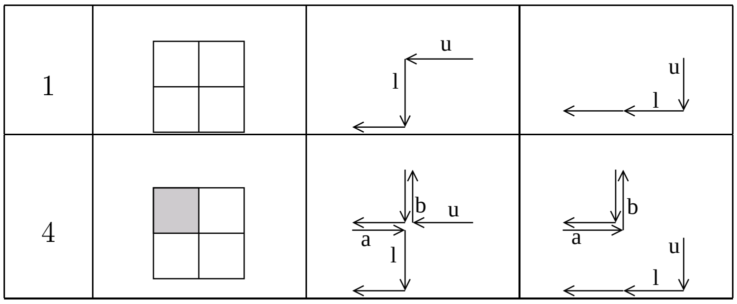


FIG. 4.22 – Les deux précodes couverts par le précode l_1 en 2d.

Il est intéressant d'étudier plus précisément le précode l_1. En effet, il couvre les précodes 1 et 4, qui semblent assez différents, mais que nous traitons pourtant avec le même algorithme. Nous pouvons voir ces deux précodes figure 4.22, ainsi que la carte avant et après leurs traitements. La seule différence entre ces deux précodes, porte sur les brins appelés a et b pour le précode 4. En effet, dans ce cas, il faut β_1-coudre a à b, alors que cette opération n'est pas utile pour le précode 1. Mais cette opération, bien qu'inutile, n'est pas source d'erreur. En effet, pour ce précode 1, nous avons alors $a = u$ et $b = l$, et β_1-coudre a à b revient à « recoudre » u à l, qui étaient déjà β_1-cousus. Nous pouvons donc effectuer cette opération de couture tant pour le précode 1 que pour le précode 4, ce qui permet d'obtenir un seul traitement et évite d'avoir à tester précisément quel est le précode courant.

La définition des quatre précodes lignels et de leur traitement définit complètement l'algorithme optimal d'extraction de la carte lignel.

4.5.4 Les précodes pour la carte des frontières

L'extraction de la carte des frontières nécessite de fusionner chaque couple d'arêtes alignées incidentes à un sommet de degré deux (cf. définition 17). Étant donné que nous effectuons la construction de cette carte en une seule passe, nous devons fusionner également les faces adjacentes de même région (fusion nécessaire pour obtenir la carte de niveau 1). Il est facile de voir qu'il existe seulement deux cas pour lesquels deux arêtes alignées sont incidentes à un sommet de degré deux : ce sont les précodes présentés figure 4.23. C'est donc seulement ces deux précodes qu'il va falloir traiter (en plus des quatre de la carte

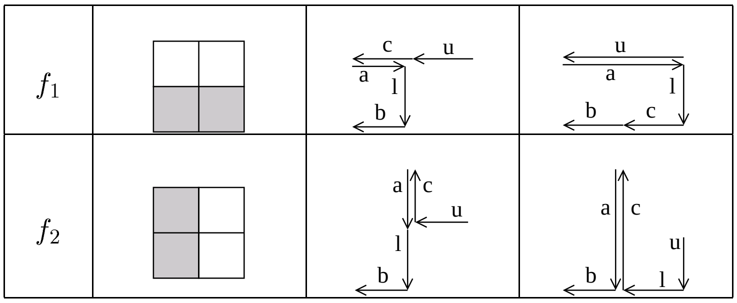


FIG. 4.23 – Les deux précodes pour obtenir la carte des frontières en 2d.

lignel), et c'est uniquement dans ces deux cas que nous allons fusionner deux arêtes. Nous devons conserver les quatre précodes lignel, car nous devons, lorsque nous ne sommes pas dans un de ces deux précodes frontières, traiter la configuration courante par l'un des quatre précodes lignels afin de réaliser les éventuelles fusions de faces nécessaires.

Sur chaque ligne de la figure 4.23, nous avons représenté le précode, puis la carte avant son traitement et enfin celle obtenue après traitement. Contrairement aux quatre précodes lignels, nous connaissons ici tous les brins présents dans la carte avant le traitement du précode. En effet, ces deux précodes ne sont pas partiels, et nous connaissons donc les régions des quatre pixels. Comme pour la carte lignel, l'écriture des algorithmes de traitement de ces deux précodes se fait de manière simple comme le montrent les algorithmes 11 et 12.

Algorithme 11 Code associé au précode f_1 en 2d

Entrée : *last* et *up*
(x, y) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$a \leftarrow \beta_0(last)$; $b \leftarrow \beta_1(last)$;
$c \leftarrow \beta_1(up)$; $d \leftarrow \beta_1(c)$;
β_1-découdre$(last)$; β_0-découdre(c);
β_1-découdre(c) ; β_2-découdre(c);
β_1-coudre$(last, c)$; β_1-coudre(c, b);
β_1-coudre(up, d) ; β_2-coudre(up, a);
Le plongement du sommet incident à $c \leftarrow (x, y)$;
retourner last;

Algorithme 12 Code associé au précode f_2 en 2d

Entrée : *last* et *up*
(x, y) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$a \leftarrow \beta_0(last)$; $b \leftarrow \beta_1(last)$; $c \leftarrow \beta_1(up)$;
β_0-découdre$(last)$; β_1-découdre$(last)$; β_1-découdre(up);
β_1-coudre$(up, last)$; β_1-coudre$(last, c)$; β_0-coudre(b, a);
Le plongement du sommet incident à $last \leftarrow (x, y)$;
retourner up;

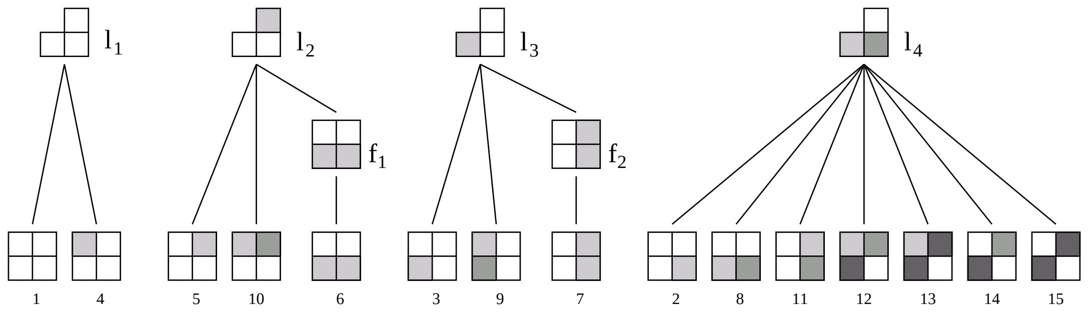


FIG. 4.24 – Les arbres de couverture pour la carte des frontières en 2d.

Pour ces deux précodes, nous ne faisons pas de modification de plongement, excepté pour le « nouveau sommet ». En effet, par exemple pour le précode f_1, nous conservons les brins a et u car ce sont ceux qui ont les plongements sommets que nous voulons obtenir dans la carte résultat. Pour le précode f_2, par exemple lors de la β_0-couture de b avec a, il y avait avant deux sommets topologiques distincts (incidents aux brins b et c) et après la couture il n'y en a plus qu'un. Dans ce cas, l'ordre de la couture importe : c'est toujours le sommet ayant un rapport avec le premier brin passé en paramètre qui est conservé. Lors du traitement du précode f_2, nous conservons donc bien le plongement du sommet incident à b.

Remarquons également que pour ces deux précodes, nous conservons un brin particulier, c pour le précode f_1 et l pour f_2, qui est d'abord totalement décousu avant d'être ensuite utilisé à un autre endroit de la carte. Nous aurions pu choisir de détruire ce brin, et ensuite de créer un nouveau brin. Mais notre solution évite une destruction suivie d'une création, et permet donc un gain de temps d'exécution.

Pour extraire directement la carte des frontières à partir d'une image, nous appliquons l'algorithme 5 en cherchant à chaque étape de la boucle quel est le précode courant, parmi les 6 précodes possibles (les 2 précodes f_1 et f_2 plus les 4 précodes lignels) et en exécutant le code correspondant. Nous pouvons vérifier que ces six précodes couvrent bien toutes les configurations que nous pouvons rencontrer dans l'image sur les arbres de couverture présentés figure 4.24. Nous voyons sur cette figure que les deux précodes f_1 et f_2 couvrent uniquement un seul précode étant donné que ce ne sont pas des précodes partiels.

4.5.5 Les précodes pour la carte topologique

Pour ce niveau, nous devons fusionner chaque couple d'arêtes incidentes à des sommets de degré deux (cf. définition 18). Il existe seulement quatre cas différents pour lesquels cette fusion doit être réalisée. Ce sont les quatre précodes présentés figure 4.25. Cette figure montre sur la première ligne ces précodes, sur la deuxième la configuration de la carte avant de traiter le précode correspondant, et sur la dernière ligne la carte à obtenir après ce traitement. Nous n'avons pas représenté le plongement des arêtes sur les cartes de cette figure, mais il faut bien entendu le modifier. Nous supposons que le plongement de

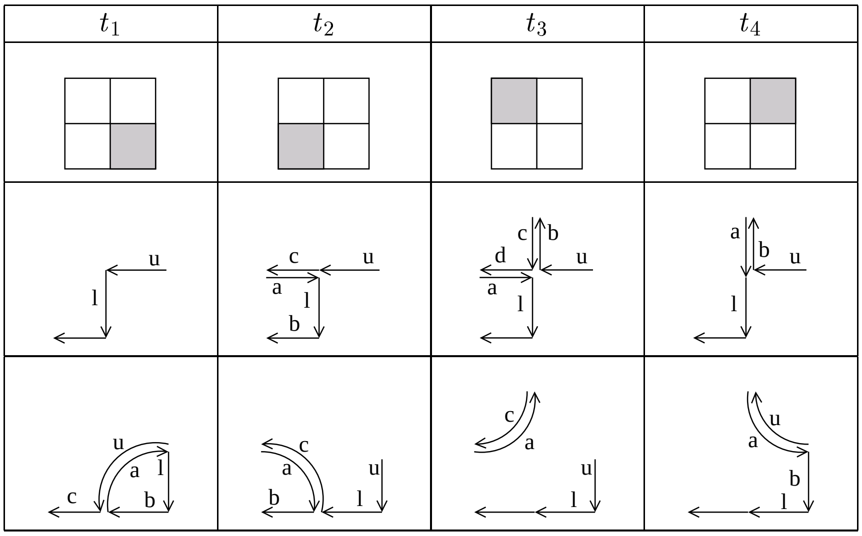


FIG. 4.25 – Les quatre précodes pour obtenir la carte topologique en 2d.

la carte est celui décrit section 4.3 au moyen d'un plongement sommet et d'un plongement arêtes ouvertes. Les algorithmes 13, 14, 15 et 16 donnent le code associé à ces quatre précodes.

Algorithme 13 Code associé au précode t_1 en 2d

Entrée : *last* et *up*
(x, y) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$a \leftarrow$ un nouveau brin ; $b \leftarrow$ un nouveau brin;
$c \leftarrow \beta_1(last)$; β_2-coudre(up, a);
Ajouter le plongement sommet de *last* en fin du plongement arête de a;
β_0-découdre$(last)$; β_1-découdre$(last)$;
β_1-coudre$(a, last)$; β_1-coudre$(last, b)$;
β_1-coudre(b, a) ; β_0-coudre(c, up);
Le plongement du sommet incident à $b \leftarrow (x, y)$;
retourner *last*;

Pour l'algorithme 13, nous savons que le brin *up* ne possède pas de plongement arête car il n'est pas β_2-cousu. Au cours du traitement de ce précode, il est β_2-cousu au brin a et forme donc désormais une arête. Afin de conserver le plongement, nous devons initialiser le plongement de cette arête comme étant une courbe réduite à un seul sommet : l'ancien sommet incident au brin *last*. Cette initialisation est réalisée au moyen d'une méthode générique qui ajoute les coordonnées d'un point en fin d'un plongement arête. Dans ce cas, étant donné qu'il n'y a pas de plongement arête, cela revient à initialiser ce plongement. Les plongements sommets n'ont pas besoin d'être mis à jour, excepté le nouveau sommet, car les méthodes de coutures se chargent de modifier correctement les plongements suivant les évolutions des orbites sommets. Par exemple pour la dernière couture de l'algorithme 13, il y a deux sommets topologiques incidents à c et à a. Le plongement sommet conservé est celui du sommet ayant un lien avec le premier paramètre, c'est-à-dire le sommet incident à c.

Algorithme 14 Code associé au précode t_2 en 2d

Entrée : *last* et *up*
(x, y) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$a \leftarrow \beta_0(last)$; $b \leftarrow \beta_1(last)$; $c \leftarrow \beta_1(up)$;
Ajouter le plongement sommet de *last* en fin du plongement arête de a;
β_0-découdre($last$) ; β_1-découdre($last$) ; β_1-découdre(up);
β_0-coudre(b, a) ; β_1-coudre($up, last$) ; β_1-coudre($last, c$);
Le plongement du sommet incident à $last \leftarrow (x, y)$;
retourner up;

Pour l'algorithme 14 traitant le précode t_2, l'arête incidente au brin a possède peut-être un plongement. Mais l'ajout en fin des coordonnées d'un sommet fonctionne correctement, que l'arête ait un plongement ou non. Pour les plongements sommets, il faut initialiser correctement celui du sommet incident au brin *last*, mais les autres n'ont pas besoin de mise à jour, toujours grâce aux coutures qui vont éventuellement les modifier.

Algorithme 15 Code associé au précode t_3 en 2d

Entrée : *last* et *up*
(x, y) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$a \leftarrow \beta_0(last)$; $b \leftarrow \beta_1(up)$; $c \leftarrow \beta_2(b)$;
$d \leftarrow \beta_1(c)$; $t_1 \leftarrow \beta_1(b)$; $t_2 \leftarrow \beta_1(d)$;
Ajouter le plongement sommet de *last* en fin du plongement arête de a;
Fusionner le plongement arête de b en fin du plongement arête de a;
β_0-découdre(b) ; β_1-découdre(b) ; β_2-découdre(b);
β_0-découdre(d) ; β_1-découdre(d) ; β_2-découdre(d);
β_0-découdre($last$) ; β_1-coudre(a, t_1) ; β_1-coudre(c, t_2);
β_1-coudre($up, last$);
β_2-coudre(a, c);
détruire les brins b et d;
Le plongement du sommet incident à $last \leftarrow (x, y)$;
retourner up;

Pour le traitement du précode t_3, nous pouvons remarquer sur l'algorithme 15 qu'il est un peu plus complexe que les précédents. En effet, il est possible que les arêtes incidentes au brin a et au brin b possèdent toutes deux un plongement arête. C'est pour cette raison que nous devons « fusionner » ces deux plongements, en insérant au milieu les coordonnées du sommet incident au brin *last*. La méthode se chargeant de cette fusion doit fonctionner sur des plongements vides, pour traiter les cas où une ou les deux arêtes n'ont pas de plongement arête. De plus, il faudra faire attention à l'ordre des plongements lors de cette fusion. Suivant la manière dont les courbes de dimension 1 sont implantées, nous pourrons avoir besoin de retourner une des deux courbes afin d'effectuer correctement la fusion. Lors de la β_2-couture de a avec c, si les deux brins ont tous deux un plongement arête, seul celui de a sera conservé, l'autre sera alors détruit.

De plus, cet algorithme nécessite un pré-traitement afin de gérer correctement le cas particulier où une région est incluse dans une autre sans avoir d'autre région adjacente (comme pour la région R_4 de notre image d'exemple). Dans ce cas, la frontière entre les deux régions est composée d'une boucle. Avant d'exécuter le traitement du précode t_3, il est donc nécessaire de tester si l'arête incidente au brin $\beta_0(last)$ est la même que celle incidente au brin $\beta_1(up)$. Lorsque c'est le cas, il n'y a pas de fusion d'arêtes à exécuter étant donné que le sommet est de degré un. Il suffit alors d'exécuter le traitement du précode l_1 à la place de celui de t_3 (l_1 est le précode père de t_3 dans la hiérarchie présentée figure 4.26). Le traitement de l_1 va effectuer les mêmes fusions de faces que celui de t_3, mais ne va pas fusionner les arêtes. Après ce traitement, nous obtenons bien la boucle représentant la frontière entre la région incluse et la région qui l'entoure.

Algorithme 16 Code associé au précode t_4 en 2d

Entrée : *last* et *up*
(x, y) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$a \leftarrow \beta_0(last)$; $b \leftarrow \beta_1(up)$; $c \leftarrow \beta_1(b)$;
Ajouter le plongement sommet de *last* en fin du plongement arête de a;
β_0-découdre($last$) ; β_0-découdre(b);
β_1-découdre(b) ; β_2-découdre(b);
β_1-coudre(up, c) ; β_2-coudre(up, a);
β_1-coudre(a, b) ; β_1-coudre($b, last$);
Le plongement du sommet incident à $last \leftarrow (x, y)$;
retourner b;

L'algorithme 16 est plus simple car nous sommes sûrs que le brin *up* n'a pas de plongement arête, étant donné qu'il n'est pas β_2-cousu. Son traitement est similaire à ceux déjà présentés des précodes t_1 et t_2.

Maintenant que nous disposons des quatre algorithmes associés aux quatre précodes topologiques, nous pouvons extraire la carte topologique en une seule passe de l'image au moyen de l'algorithme optimal (algorithme 5) qui teste quel est le précode courant, parmi les 10 possibles (les 4 lignels plus les 2 frontières plus les 4 topologiques). Nous pouvons voir que cet algorithme, bien qu'un peu long à décrire en détail, est simple, et facile à implanter. De plus, nous avons un algorithme optimal, qui ne fait aucune opération inutile, qui crée directement le nombre de brins nécessaires tout en conservant le plongement sous forme de courbes de dimension 1 combinées avec le plongement des sommets.

Nous vérifions sur la figure 4.26 que les dix précodes couvrent bien toutes les configurations que nous pouvons rencontrer dans l'image. Nous voyons que le seul précode couvrant encore plusieurs cas est le précode l_4. Ce cas regroupe tous les précodes pour lesquels il n'y a aucune fusion à faire. Ces cas sont seulement au nombre de 6, mais nous verrons qu'ils sont beaucoup plus nombreux en dimension 3. Nous voyons que les précodes non variétés sont tous couverts par le précode l_4. En effet, nous représentons uniquement la 4-connexité entre les régions, et de ce point de vue les précodes non variétés 13, 14 et 15 sont équivalents au précode 12 pour lequel aucune fusion n'est nécessaire. Il est possible d'envisager une extension de nos travaux afin de représenter la 8-connexité, et dans ce cas ces précodes devraient alors être traités de manière différente.

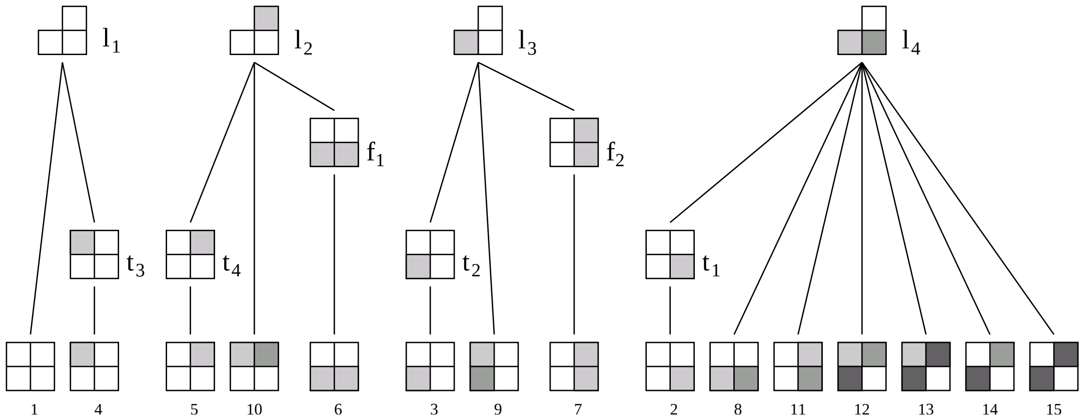


FIG. 4.26 – Les arbres de couverture pour la carte topologique en 2d.

Maintenant que nous avons défini totalement les algorithmes d'extraction des trois niveaux de carte possibles, nous allons revenir sur la gestion du bord de l'image. En effet, c'est grâce à sa définition adéquate que l'algorithme d'extraction reste simple, sans traiter un grand nombre de cas particuliers.

4.5.6 Le bord de l'image

Pour traiter les bords de l'image, une première solution consiste à traiter différemment les premières et dernières lignes et colonnes. Mais cela oblige à définir le traitement de chaque précode de plusieurs manières différentes, avec de légères variations à chaque fois. Cette multitude de méthodes risque d'être une source d'erreurs, et rend beaucoup plus difficile les évolutions ou modifications des précodes. De plus, l'algorithme d'extraction perd en lisibilité, car il faut alors effectuer le parcours de la première et de la dernière ligne dans une boucle spécifique, et pour chaque ligne traiter différemment la première et dernière colonne.

Afin de compter le nombre de méthodes supplémentaires que cette solution oblige à traiter, nous schématisons le bord d'une image au moyen des précodes figure 4.27. Cette figure montre quels précodes peuvent être rencontrés suivant la position du pixel courant dans l'image. De manière évidente, les quatre angles de l'image sont toujours fixes car un seul pixel appartient à l'image, les trois autres sont des pixels appartenant à la région infinie donc tous de la même région. Pour les quatre bords (privés des angles), nous pouvons avoir à chaque fois deux précodes différents qui sont donnés sur la figure. Par exemple, sur le bord gauche de l'image, nous pouvons avoir uniquement le précode 7 ou le précode 11. Chacun de ces précodes doit avoir un traitement particulier suivant sa position dans l'image. Par exemple le précode 7 doit avoir un traitement différent s'il se trouve sur le bord gauche, le bord droit ou au milieu de l'image. Nous devons donc définir 12 traitements particuliers afin de gérer correctement le bord de l'image.

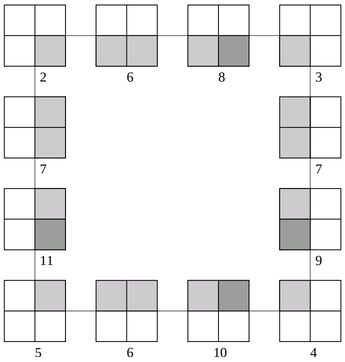
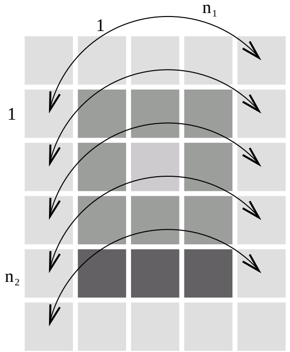
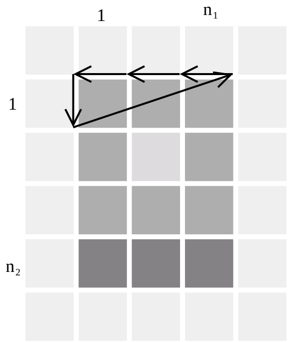


FIG. 4.27 – Les précodes possibles en fonction du bord de l'image.



a. L'image sur un cylindre : les pixels reliés par une flèche sont identifiés.



b. La carte pour le bord supérieur de l'image.

FIG. 4.28 – Gestion du bord en 2d.

Cette solution étant trop lourde à mettre en œuvre, nous avons décidé de gérer le bord en considérant que l'image est plaquée sur un cylindre (cf. figure 4.28.a). Cette solution permet de ne pas se préoccuper de la position du pixel courant dans l'image : le traitement du précode courant est identique dans tous les cas. Pour mettre en œuvre ce traitement, il suffit de créer le bord supérieur de l'image de la manière présentée sur l'exemple figure 4.28.b. Ce bord étant fermé pour β_1, il simule le fait que l'image est sur un cylindre.

La figure 4.28.a présente une image de $n_1 \times n_2$ pixels. Le fait de considérer qu'elle est entourée par la région infinie permet d'avoir des pixels tout autour de l'image (aux coordonnées $(0, y)$, $(n_1 + 1, y)$, $(x, 0)$ et $(x, n_2 + 1)$) appartenant tous à la région infinie. Ensuite, plaquer cette image sur un cylindre revient à identifier chaque pixel $(n_1 + 1, j)$ au pixel $(0, j + 1)$ (sur la figure 4.28.a nous identifions les pixels reliés par une flèche). Au cours du balayage de l'image, nous ne nous préoccupons plus de savoir si nous sommes en train de traiter un précode de bord ou non.

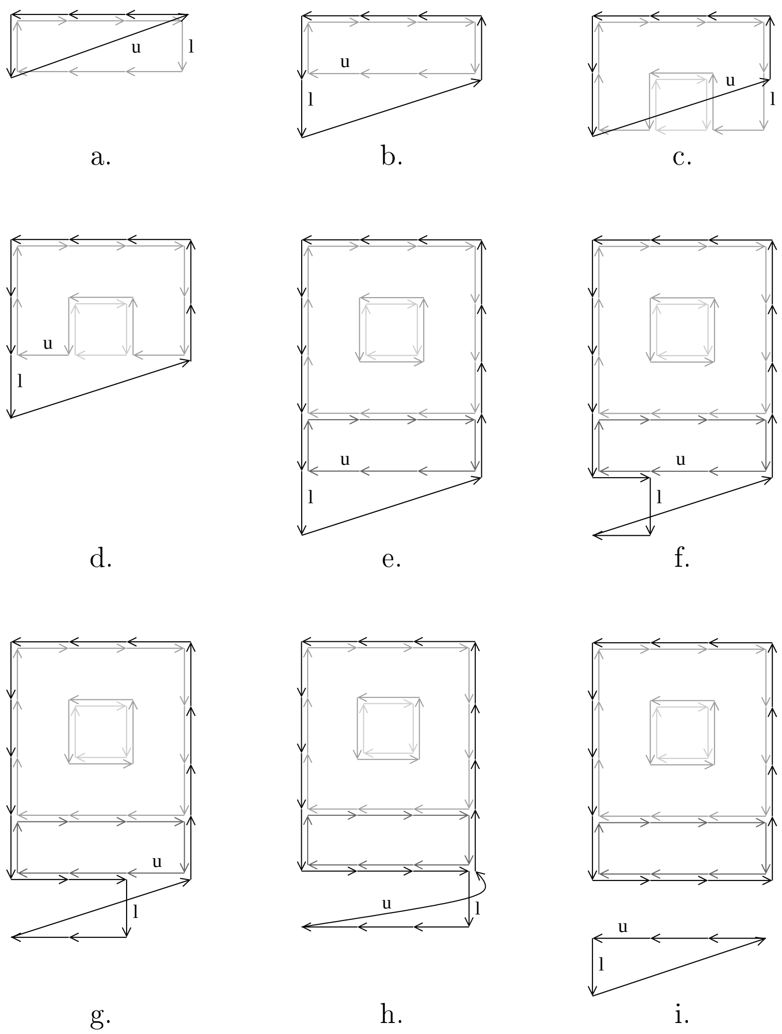


FIG. 4.29 – Déroulement de l'algorithme d'extraction de la carte lignel en 2d.

De par la conception initiale du bord, lorsque nous sommes sur le bord droit, nous créons le bord gauche de la ligne suivante. Pour cette raison, les coordonnées du prochain pointel à créer sont calculées modulo la largeur de l'image plus un. Si les coordonnées du pixel courant sont (i, j), les coordonnées du nouveau pointel sont (i modulo $(n_1 + 1), j + (i$ div $(n_1 + 1))$).

Afin de comprendre le fonctionnement de notre algorithme optimal d'extraction (algorithme 5 page 52) lorsque nous traitons des précodes de bord, nous déroulons cet algorithme sur l'image présentée figure 4.28.a pour extraire la carte lignel. La carte initiale est donc le bord supérieur de l'image (figure 4.28.b). La figure 4.29 montre quelques étapes importantes de l'algorithme. Les coordonnées des pixels sont données figure 4.28, et les coordonnées d'un pointel en haut et à gauche d'un pixel (x, y) sont $(x-1, y-1)$. De ce fait, le premier pointel de l'image a pour coordonnées $(0,0)$.

Après avoir traité la première ligne, mais avant d'avoir traité le dernier pixel, nous avons obtenu la carte *a*. Nous traitons ensuite le dernier précode de cette ligne (qui est forcément le précode 3 quelle que soit l'image). Après ce traitement, nous obtenons la carte *b*. En effet, les coordonnées du pixel courant sont $(4,1)$. Le nouveau sommet aurait dû être plongé aux coordonnées $(4,1)$, mais grâce au cylindre, il va être plongé aux coordonnées $(0,2)$. Nous voyons sur la carte *b* que *last* désigne bien le brin qui est à gauche du prochain pixel courant. Le brin *up* sera donc également correctement calculé. La carte *c* montre le cas similaire pour la ligne suivante. Cette carte montre la situation avant de traiter le dernier précode de cette ligne, et la carte *d* après l'avoir traité.

Pour le traitement de la dernière ligne, la carte *e* montre la carte avant le traitement du premier précode de cette ligne. Nous pouvons vérifier à l'aide des cartes *f*, *g* et *h* l'évolution de la carte lors du traitement des précodes de la dernière ligne. Enfin, la carte *i* montre le traitement du dernier précode de l'image. Après ce précode, la carte lignel de l'image est fermée, et le bord initial se retrouve maintenant en bas, et déconnecté. Étant devenu inutile, il est détruit.

Nous avons montré ici comment le bord intervenait lors de l'extraction d'une carte lignel. Cela fonctionnera exactement de la même manière pour l'extraction des autres niveaux de carte. En effet, quel que soit ce niveau, le bord de l'image est toujours le même et représente les lignels de celui-ci, les fusions d'arêtes étant exclusivement réalisées pour les brins au « milieu » de la carte en cours de construction.

4.6 Expérimentations et analyse

Nous avons implanté entièrement l'algorithme optimal d'extraction à base de précodes pour les trois niveaux de carte possible. Pour les niveaux 1 et 2 nous avons utilisé uniquement un plongement sommet où nous associons les coordonnées du pointel correspondant à chaque sommet topologique de la carte, et pour le niveau 3 le plongement sommet plus arête ouverte présenté section 4.3. Nous avons codé les courbes de dimension 1 avec des 1-cartes, ce qui nous permet d'avoir une structure hiérarchique. De plus, cette méthode s'étend sans problèmes en dimension supérieure, où nous pouvons plonger chaque cellule topologique de dimension i par une i-carte. Pour comparer l'espace mémoire nécessaire aux différents niveaux de carte de manière « équitable », nous avons implanté le plongement de manière optimisée pour chacun de ces niveaux.

Nous allons étudier et comparer les résultats d'extraction des trois niveaux de carte pour deux types d'images différentes. Dans un premier temps, section 4.6.1, nous avons étudié les images classiquement utilisées dans la communauté de la segmentation d'images, avec entre autres l'incontournable Lena. Section 4.6.2, nous présentons nos résultats sur quelques images médicales, qui sont des IRM de cerveau dans le cadre d'examens pour détecter des tumeurs cérébrales.

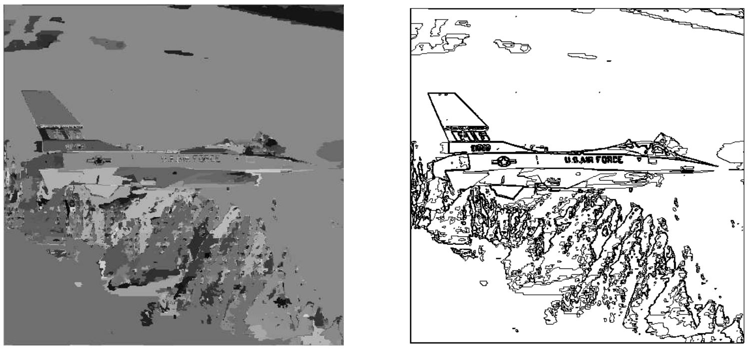


FIG. 4.30 – Airplane, 512 × 512, 6361 régions.

TAB. 4.1 – Airplane, 512 × 512, 6361 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	95 754	56 280	30 490
Nb de sommets topologiques	41 803	22 066	9 171
Espace mémoire en kilo-octets	2 744	1 665	1 230
Temps de calcul en secondes	3,45	3,26	3,12

4.6.1 Images standard

Nous présentons ici les différents résultats de l'extraction des trois niveaux de carte pour les six images segmentées présentées figures 4.30 à 4.35. Ces figures montrent dans un premier temps l'image segmentée en régions (une couleur par région) puis la carte obtenue qui représente les frontières interpixel. Les images ont fait l'objet d'une segmentation simple, sans pré ni post-traitement. Nous les utilisons ici uniquement afin de comparer les différents niveaux de carte.

Nous étudions, dans un premier temps, l'évolution du nombre de brins codant la topologie des objets, en fonction du niveau de carte. Nous regardons également le nombre de sommets topologiques, ce qui donne une idée du nombre de fusions d'arêtes effectuées. Puis nous comparons l'évolution de l'espace mémoire de la structure globale, comprenant le modèle topologique, le modèle géométrique et l'arbre d'inclusion. Nous présentons ces résultats tout d'abord de manière précise dans les tableaux 4.1 à 4.6. Dans ces tableaux, nous avons donné l'espace mémoire nécessaire en kilo-octets, et le temps d'extraction en secondes. Ce temps de calcul a été mesuré sur un Pentium III 600 Mhz, et inclus le temps nécessaire au calcul de l'arbre d'inclusion ainsi que le temps consacré aux entrées-sorties.

Afin de mieux visualiser l'évolution générale de ces caractéristiques, nous avons résumé ces résultats figure 4.36. Ces courbes montrent l'évolution de l'espace mémoire, du nombre de brins et du nombre de sommets topologiques sur nos trois niveaux de carte, en pourcentage du niveau 1. Elles permettent de vérifier que le gain entre les différents niveaux est important. Nous récapitulons le gain en espace mémoire, entre les divers niveaux dans le

FIG. 4.31 – Baboon, 512 × 512, 6705 régions.

TAB. 4.2 – Baboon, 512 × 512, 6705 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	154 450	96 444	28 348
Nb de sommets topologiques	72 396	43 393	9 345
Espace mémoire en kilo-octets	4 368	2 782	1 560
Temps de calcul en secondes	4,95	4,82	4,75

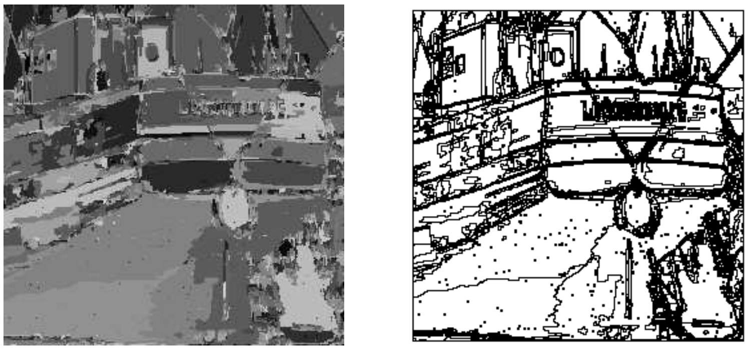


FIG. 4.32 – Cornouaille, 256 × 256, 5410 régions.

TAB. 4.3 – Cornouaille, 256 × 256, 5410 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	60 170	39 806	26 254
Nb de sommets topologiques	24 908	14 726	7 950
Espace mémoire en kilo-octets	1 752	1 195	970
Temps de calcul en secondes	1,85	1,76	1,69

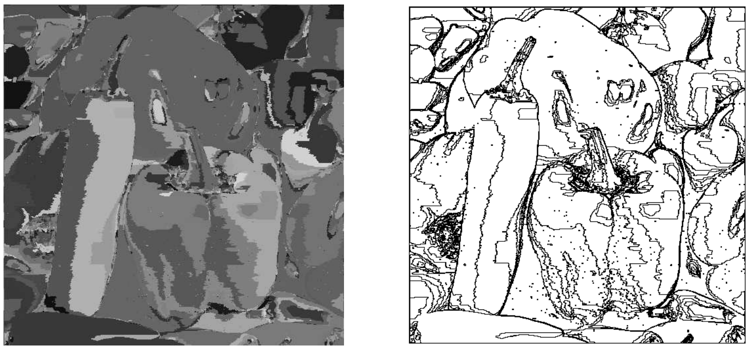


FIG. 4.33 – Peppers, 512 × 512, 5765 régions.

TAB. 4.4 – Peppers, 512 × 512, 5765 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	109 602	72 310	27 518
Nb de sommets topologiques	49 565	30 919	8 523
Espace mémoire en kilo-octets	3 113	2 093	1 285
Temps de calcul en secondes	3,16	3,01	2,92

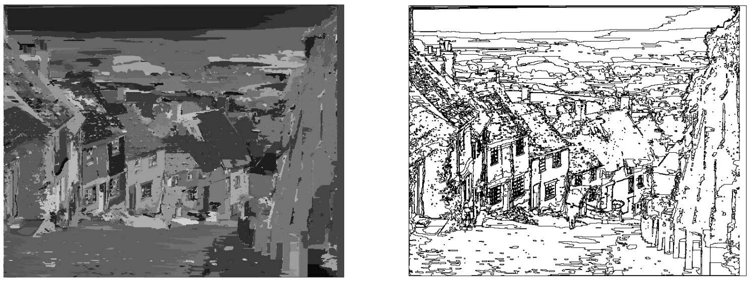


FIG. 4.34 – Goldhill, 720 × 576, 8006 régions.

TAB. 4.5 – Goldhill, 720 × 576, 8006 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	202 174	104 872	36 450
Nb de sommets topologiques	94 704	46 053	11 842
Espace mémoire en kilo-octets	5 697	3 036	1 815
Temps de calcul en secondes	9,65	9,35	9,24

FIG. 4.35 – Lena, 512 × 512, 6198 régions.

TAB. 4.6 – Lena, 512 × 512, 6198 régions.

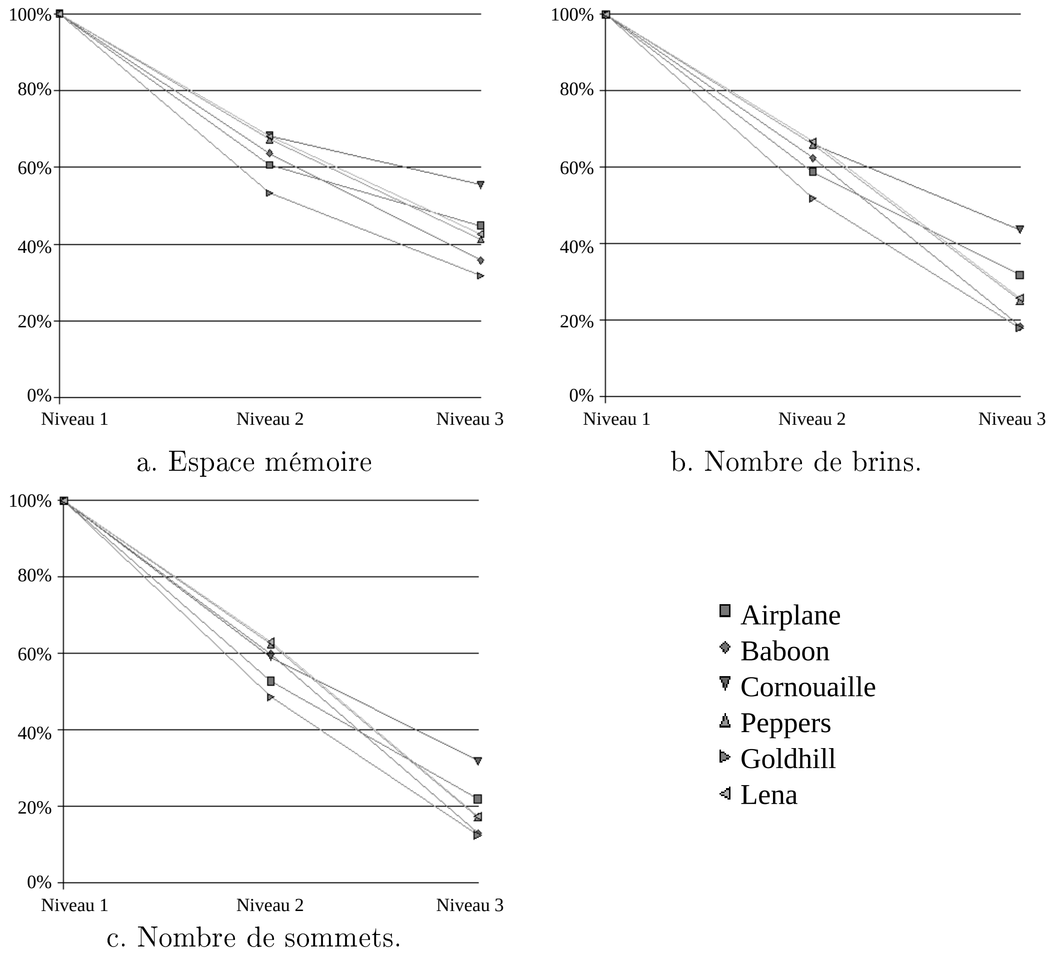
	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	109 188	72 920	28 226
Nb de sommets topologiques	49 057	30 923	8 576
Espace mémoire en kilo-octets	3 111	2 120	1 327
Temps de calcul en secondes	3,64	3,53	3,39

tableau 4.7. La première colonne de ce tableau donne le gain lorsque nous passons du niveau 1 au niveau 2. Nous pouvons voir qu'il est en moyenne de l'ordre de 35%. La deuxième colonne donne le gain lorsque nous passons du niveau 2 au niveau 3 qui est du même ordre de grandeur. La dernière colonne donne le gain entre le niveau 1 et le niveau 3, qui est ici presque de 60%. Ces chiffres montrent qu'il est très intéressant de travailler avec le niveau 3 malgré le surcoût de difficulté d'implantation.

Nous pouvons également voir, sur ces divers exemples d'extraction, que le temps d'exécution ne varie que très peu pour les différents niveaux. Il y a même une légère diminution du temps nécessaire à l'extraction des cartes de plus grand niveau, ce qui peut sembler étrange au premier abord. Mais cette légère diminution provient en fait du nombre d'allo-

TAB. 4.7 – Récapitulatif des gains en espace mémoire.

	Gain Niveau 1 vers Niveau 2	Gain Niveau 2 vers Niveau 3	Gain Niveau 1 vers Niveau 3
Airplane	39,32%	26,13%	55,18%
Baboon	36,30%	43,90%	64,27%
Cornouaille	31,77%	18,81%	44,61%
Peppers	32,75%	38,62%	58,72%
Goldhill	46,70%	40,23%	68,14%
Lena	31,87%	37,37%	57,33%
Moyenne	36,45%	34,18%	58,04%



a. Espace mémoire

b. Nombre de brins.

c. Nombre de sommets.

FIG. 4.36 – Évolution des caractéristiques, en pourcentage du niveau 1.

cations mémoire qui est beaucoup plus important lorsque le niveau est petit, et qui entraîne donc un surcoût en temps d'exécution.

4.6.2 Images médicales

Ces images proviennent d'examens IRM afin de détecter d'éventuelles tumeurs cérébrales. Elles résultent d'examens réalisés sur trois patients différents, et sont toutes de taille 256×256 pixels. Nous présentons figures 4.37 à 4.42 tout d'abord l'image segmentée, puis la carte résultante obtenue par notre algorithme d'extraction. Le nom de ces images est composé, tout d'abord du nom de l'examen, puis du numéro de la coupe correspondante, sachant que la première image correspond à la coupe inférieure, c'est-à-dire la plus proche de la base du crâne, et que la dernière correspond à la coupe supérieure.

Nous présentons dans un premier temps, comme pour les images standard, l'évolution du nombre de brins, du nombre de sommets topologiques et de l'espace mémoire total.

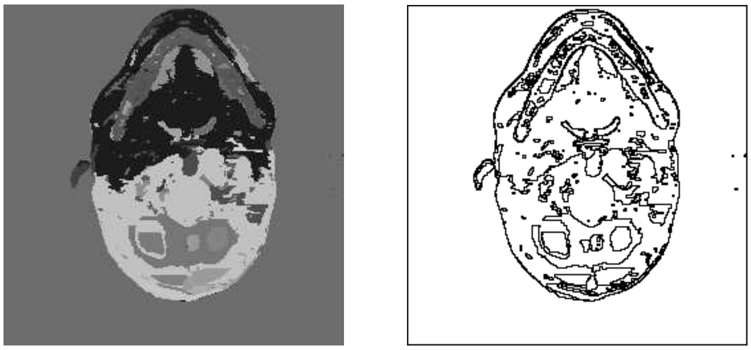


FIG. 4.37 – Pa2-i0004, 256 × 256, 771 régions.

TAB. 4.8 – Pa2-i0004, 256 × 256, 771 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	17 136	10 354	3 496
Nombre de sommets topologiques	7 910	4 519	1 090
Espace mémoire en kilo-octets	493	308	183
Temps de calcul en secondes	0,08	0,08	0,09

Ces résultats sont présentés dans les tableaux 4.8 à 4.13, pour les trois niveaux de carte possibles.

Nous pouvons remarquer sur ces quelques résultats, que le temps d'exécution est beaucoup plus court que pour les images standard. En effet, tout d'abord les images sont ici plus petites, mais contiennent également beaucoup moins de régions. Afin de comparer, pour chacune de ces images, les différentes caractéristiques des trois niveaux de carte, nous présentons les résultats dans la figure 4.43 au moyen de courbes donnant l'espace mémoire, le nombre de brins et le nombre de sommets topologiques en pourcentage du niveau 1.

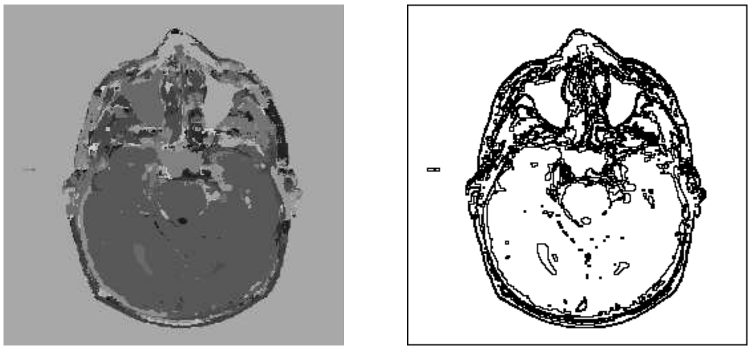


FIG. 4.38 – Pa2-i0038, 256 × 256, 2298 régions.

TAB. 4.9 – Pa2-i0038, 256 × 256, 2298 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	28 010	19 710	11 166
Nombre de sommets topologiques	11 811	7 661	3 389
Espace mémoire en kilo-octets	838	611	462
Temps de calcul en secondes	0,22	0,22	0,23

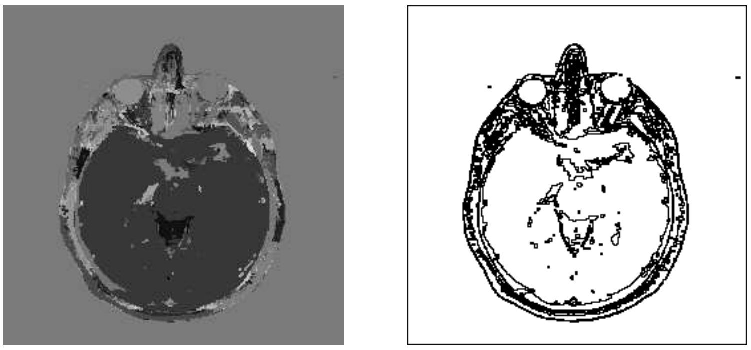


FIG. 4.39 – Pa5-i0030, 256 × 256, 1868 régions.

TAB. 4.10 – Pa5-i0030, 256 × 256, 1868 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	22 790	16 138	9 218
Nombre de sommets topologiques	9 605	6 279	2 819
Espace mémoire en kilo-octets	682	500	379
Temps de calcul en secondes	0,17	0,18	0,16

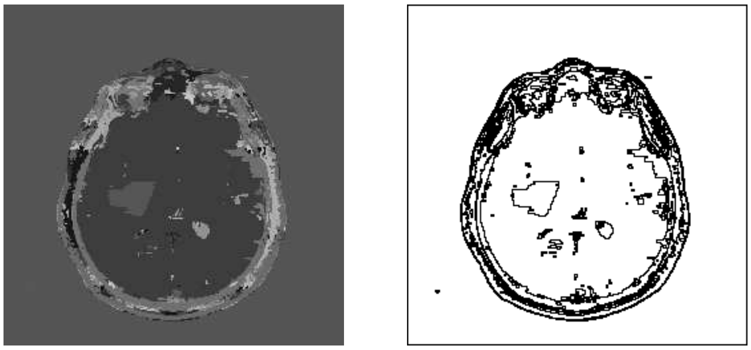


FIG. 4.40 – Pa5-i0046, 256 × 256, 1589 régions.

TAB. 4.11 – Pa5-i0046, 256 × 256, 1589 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	20 314	13 844	7 714
Nombre de sommets topologiques	8 655	5 420	2 355
Espace mémoire en kilo-octets	605	428	321
Temps de calcul en secondes	0,14	0,14	0,14

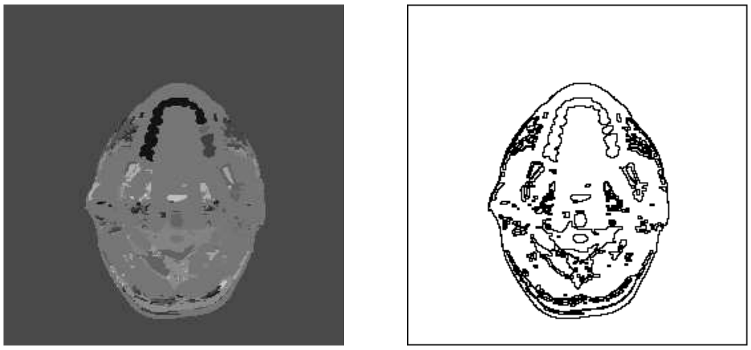


FIG. 4.41 – Pa6-i0012, 256 × 256, 595 régions.

TAB. 4.12 – Pa6-i0012, 256 × 256, 595 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	12 834	7 430	2 728
Nombre de sommets topologiques	5 894	3 192	841
Espace mémoire en kilo-octets	370	222	136
Temps de calcul en secondes	0,08	0,07	0,07

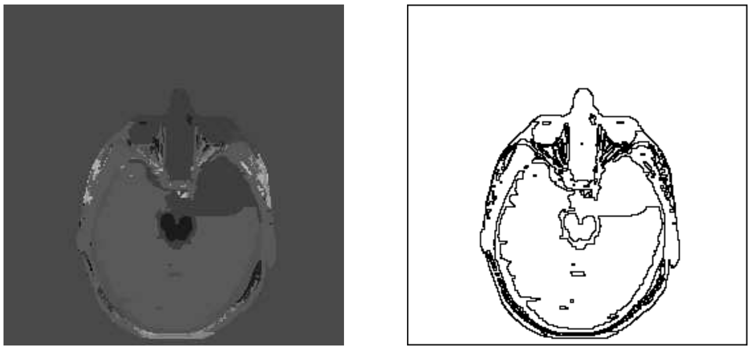


FIG. 4.42 – Pa6-i0054, 256 × 256, 452 régions.

TAB. 4.13 – Pa6-i0054, 256 × 256, 452 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	11 720	6 358	2 238
Nombre de sommets topologiques	5 441	2 760	700
Espace mémoire en kilo-octets	334	188	112
Temps de calcul en secondes	0,06	0,07	0,07

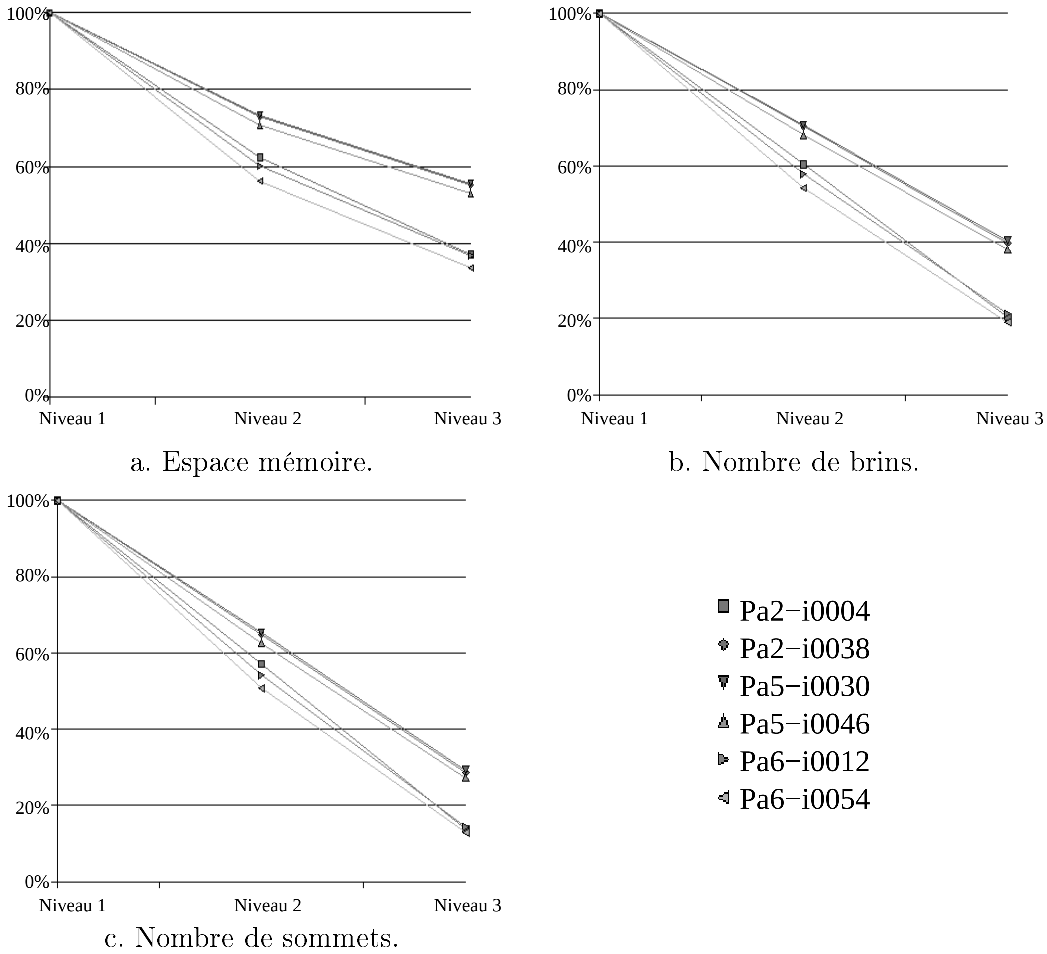


FIG. 4.43 – Évolution des caractéristiques, en pourcentage du niveau 1.

TAB. 4.14 – Récapitulatif des gains en espace mémoire.

	Gain Niveau 1 vers Niveau 2	Gain Niveau 2 vers Niveau 3	Gain Niveau 1 vers Niveau 3
Pa2-i0004	37,57%	40,44%	62,81%
Pa2-i0038	27,06%	24,40%	44,86%
Pa5-i0030	26,66%	24,08%	44,32%
Pa5-i0046	29,20%	25,05%	46,93%
Pa6-i0012	39,92%	38,63%	63,13%
Pa6-i0054	43,78%	40,06%	66,30%
Moyenne	34,03%	32,11%	54,73%

Cette figure permet de visualiser plus facilement l'évolution de ces caractéristiques pour les trois niveaux. Nous pouvons remarquer que le gain est à peu près comparable à celui des images standard, pour les trois caractéristiques. Nous pouvons vérifier plus précisément cette évolution pour l'espace mémoire dans le tableau 4.14 donnant le pourcentage de gain entre chaque niveau de carte.

Ici aussi, comme pour les images standard, nous observons des gains de l'ordre de 35% entre le niveau 1 et le niveau 2, de l'ordre de 30% entre le niveau 2 et le niveau 3 et de l'ordre de 55% entre le niveau 1 et le niveau 3.

4.7 Conclusion

Dans ce chapitre, nous avons défini un modèle combinatoire permettant de représenter les images segmentées en dimension 2, donné deux algorithmes d'extraction de ce modèle : un premier simple et facile à implanter, et un deuxième optimal s'appuyant sur la notion de précode. Nous avons également expliqué formellement cette notion de précode, et introduit une nouvelle notion permettant de regrouper plusieurs précodes. Enfin, nous avons testé et comparé nos algorithmes afin d'étudier précisément l'évolution de différentes caractéristiques sur différentes images.

De nombreux modèles existaient déjà en dimension 2, et notre modèle est similaire à deux d'entre eux : le TGF et les cartes discrètes. Notre apport, pour cette dimension, n'est pas d'avoir défini un nouveau modèle, qui de surcroît ne peut pas vraiment être considéré comme nouveau, mais plutôt de l'avoir défini de manière différente. En effet, nous avons introduit un nouveau concept de niveau de simplification permettant de donner une définition simple, générique, et amenant directement à la définition des algorithmes d'extraction. Notre approche progressive permet la définition de la carte topologique de manière hiérarchique, chaque étape pouvant se définir simplement à partir de l'étape précédente et d'un type particulier de fusion.

Cette définition progressive nous a ensuite permis de définir de manière immédiate un premier algorithme d'extraction, qui découle directement de cette définition. Cet algorithme est simple à comprendre et à implanter, mais n'est pas optimal, bien que linéaire en le nombre de pixels de l'image. En effet, il construit la carte topologique en plusieurs passes de l'image.

Nous avons donc ensuite présenté un algorithme optimal d'extraction de n'importe quel niveau de carte à partir d'une image segmentée en régions. Cet algorithme est générique, simple, et effectue la construction de n'importe quel niveau de carte en une unique passe de l'image. Pour cela, nous avons étudié précisément la notion de précode et de précode partiel, et cherché quels précodes nous devions traiter, et de quelle manière, pour chacun des différents niveaux de carte. Notre définition progressive nous a ici permis d'étudier ces précodes de manière simple, en étudiant à chaque niveau uniquement les nouveaux précodes à traiter par rapport au niveau précédent. Nous obtenons au final une hiérarchie de précodes classés par niveau, qui nous permet d'obtenir immédiatement les précodes à traiter afin d'extraire n'importe quel niveau de carte. Cette définition progressive nous a également permis de factoriser ces précodes sans difficulté supplémentaire, alors que cela aurait demandé beaucoup plus de travail et d'attention avec une définition directe. Nous obtenons finalement 4 précodes pour le niveau 1, 6 pour le niveau 2 et 10 pour le niveau 3, au lieu des 15 précodes existant en dimension 2 (ou des 12 précodes variétés).

Ces avantages justifient donc ce travail en dimension 2. Mais cette définition présente encore un dernier avantage, qui est en fait le plus important. En effet, notre définition progressive peut s'étendre en dimension supérieure. De plus, les algorithmes utilisés en dimension 2 vont pouvoir également être étendus en dimension supérieure, et les méthodes de travail utilisées ici afin de définir les précodes nécessaires à chaque niveau vont être similaires. Et c'est en dimension supérieure que les avantages apportés par la simplicité de notre définition seront des avantages primordiaux dans la définition d'un modèle représentant les images segmentées en dimension n.