Analyse des documents graphiques : une approche par reconstruction d'objets

15 juin 2007

Chapitre 1

Opérateurs d'extraction de primitives graphiques

1.1 Introduction

Nous présentons dans ce chapitre nos opérateurs d'extraction de primitives graphiques. Notre approche dans ce manuscrit vise à la reconstruction d'objets. Dans cette optique nous avons défini des opérateurs basés sur les différentes représentations région, contour et squelette. Celles-ci sont en effet complémentaires en raison de leurs différences de niveaux d'extraction. De cette façon, nos opérateurs permettront à terme de multiples cas de reconstruction d'objets. Évidemment nous avons dû faire des choix pour l'implémentation de nos opérateurs parmi le nombre important de méthodes existantes[1]. Le but de notre approche est avant tout d'illustrer l'intérêt de la combinaison des opérateurs, et donc de la reconstruction d'objets. Nous avons donc privilégié dans le choix de nos méthodes la diversité des représentations au détriment dans certains cas de la complexité et de la précision des méthodes.

Dans la suite de ce chapitre, nous présentons tout d'abord dans la section (1.2) nos opérateurs d'extraction par approche contour. Nous y présentons une méthode originale exploitant un paradigme marquage/suivi pour l'extraction robuste de contours. La section (1.3) présente ensuite nos opérateurs d'extraction par approche région. Ceux-ci permettent l'extraction des composantes et/ou occlusions ainsi que la construction de différentes relations (de voisinage, d'inclusion et sous contraintes de distance). Le caractère original de notre approche réside dans la combinaison des relations afin de multiplier les modèles de représentation. Nous présentons ensuite dans la section (1.4) nos opérateurs d'extraction par approche squelette. Nous y avons développé une approche originale de construction du graphe de squelette par catégorisation des pixels. Celle-ci est alors particulièrement adaptée pour le traitement des jonctions multiples. Finalement, dans la section (1.5) nous concluons sur ce chapitre.

[1] Nous reportons le lecteur à notre chapitre (**??**) d'état de l'art sur ces méthodes.

1.2 Extraction par approche contour

1.2.1 Introduction

Dans cette section, nous présentons nos premiers opérateurs d'extraction de primitives graphiques utilisés dans notre approche de reconstruction d'objets : ceux par approche contour. Au cours du chapitre (**??**) nous avons présenté une étude bibliographique sur les méthodes[2] par approche contour. Nous avons montré que celles-ci pouvaient opérer par morphologie mathématique, balayage de ligne ou suivi de contours. Nous avons alors argumenté la supériorité des méthodes de suivi de contours. Celles-ci permettent en effet une extraction à la fois robuste et rapide des contours. Ces méthodes peuvent être également employées[3] pour le marquage de composantes connexes. Elles présentent donc divers avantages, elles restent cependant sensibles dans le cas de contours fortement bruités. La figure (1.1) donne un exemple de résultat de suivi de contours par la méthode de [Black 81] (b) d'une image bruitée (a).

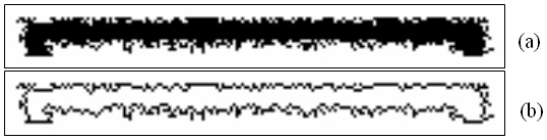


FIG. 1.1 – (a) image bruitée (b) résultat du suivi de contours

Nous argumentons ici que cette sensibilité est principalement due au manque d'informations employées a priori par les méthodes de suivi de contours. En effet celles-ci suivent les contours "à l'aveugle" sur l'image sans aucune information préliminaire sur sa topologie. Pour nos opérateurs, nous avons décidé d'étendre les méthodes de suivi de contours afin de les rendre moins sensibles aux contours bruités. Nous avons pour cela développé une approche originale basée sur un marquage préliminaire des composantes connexes. Notre approche est détaillée dans la figure (1.2). Elle procède en trois étapes : marquage des composantes & détection des contours, affinage puis suivi. De cette manière, nous employons un paradigme inverse aux méthodes de la littérature[3] qui utilisent une méthode de suivi pour le marquage des composantes. Nous présentons chacune de nos étapes dans les trois sous-sections suivantes (1.2.2), (1.2.3) et (1.2.4).

FIG. 1.2 – Approche par marquage/suivi

[2] Nous reportons le lecteur page **??** sur ces méthodes.
[3] Nous reportons le lecteur page **??** sur ces aspects.

1.2.2 Marquage de composantes et détection de contours

Dans une première étape nous procédons à la détection des contours. Celle-ci vise à marquer les pixels contours de l'image sans les chaîner. Nous utilisons pour cela une méthode basée sur un marquage des composantes connexes par propagation[4] [Rosenfeld 82]. Cette dernière est détaillée dans le pseudo-algorithme[5,6,7] (1.2.1), la figure (1.3) en illustre le principe. À partir d'un pixel forme d'entrée (b) la méthode propage l'étiquette de la composante (a) à l'ensemble des pixels voisins connexes (c). La propagation est ainsi répétée (d) à partir des pixels marqués (c) tant que des pixels formes non marqués subsistent dans la composante.

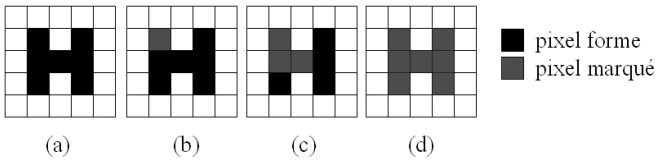


FIG. 1.3 – (a) image (b) pixel d'entrée (c) propagation 1 (d) propagation 2

Contrairement aux autres méthodes existantes dans la littérature[4], cette méthode par propagation emploie une méthodologie de marquage séquentiel. Ceci signifie que chaque composante est marquée en un passage. Dans un marquage parallèle, le marquage définitif d'une composante est obtenu à l'issue du marquage complet de l'image. Cette séquentialité permet une détection simultanée des contours durant le marquage, ce qui nous a incité à choisir cette méthode. Le pseudo-algorithme (1.2.1) détaille le principe de la détection. Elle est basée sur la recherche dans le 8-voisinage des pixels formes de la composante des pixels voisins fonds. Les pixels contours détectés correspondent donc aux pixels fonds connexes aux pixels formes de la composante. Néanmoins, notre méthode s'applique également aux pixels contours formes de la composante[8]. Les pixels détectés sont marqués comme pixels contours sur l'image et ajoutés dans un tableau (*contours*) propre à chaque composante. La figure (1.4) donne un exemple de résultat de détection de contours (b) de l'image (a).

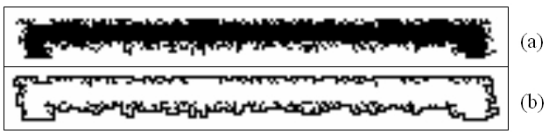


FIG. 1.4 – (a) image (b) contours détectés

[4] Nous reportons le lecteur page ?? pour une introduction aux méthodes de marquage.
[5] Algorithme où les actions à entreprendre y sont exprimées en langage naturel.
[6] La présentation de ces pseudo-algorithmes est basée sur le package LaTeXPseudocode [Kreher 05].
[7] Nous emploierons cette forme d'algorithme plus synthétique dans ce manuscrit.
[8] Nous n'abordons pas ces aspects afin d'alléger la présentation de notre méthode.

Pseudo-algorithme 1.2.1: PROPAGATION(*image*)

initialisation par $fond$ des bords de *image*
$composantes \leftarrow \emptyset$
$label \leftarrow 1$ $\quad ajouté \leftarrow -1$
tant que il existe un point d'entrée p_e de la $forme$ dans *image*
faire {
- $pile \leftarrow \emptyset$
- $label \leftarrow label + 1$
- $contours \leftarrow englobant \leftarrow gravité \leftarrow \emptyset$
- ajouter p_e dans *pile*
- **pour chaque** p_i de *pile*
  - **faire** {
    - marquer p_i dans *image* par la valeur de *label*
    - mise à jour de *englobant* par p_i
    - $gravité \leftarrow gravité + p_i$
    - **pour chaque** voisin p_v 8-connexe de p_i
      - **faire** {
        - **si** p_v est de la $forme$ et non marqué dans *image*
          - **alors** {
            - ajouter p_v dans *pile*
            - effacer p_v dans *image* comme *ajouté*
        - **si** p_v est du $fond$
          - **alors** {
            - ajouter p_v dans *contours*
            - marquer p_v dans *image* par $-label$
- $surface \leftarrow$ taille de *pile*
- $gravité \leftarrow \frac{gravité}{surface}$
- ajouter $c = \{label, contours, englobant, gravité, surface\}$ à *composantes*

retourner $(image, composantes)$

Ce marquage séquentiel permet également un calcul direct des caractéristiques topologiques des composantes connexes durant le marquage. Nous calculons ainsi les centres de gravité, les rectangles englobants, les surfaces, ... Le pseudo-algorithme (1.2.1) détaille ces différents calculs. La figure (1.5) les illustre avec un exemple d'image de symbole (a) et la représentation graphique des rectangles englobants de ses composantes connexes (b).

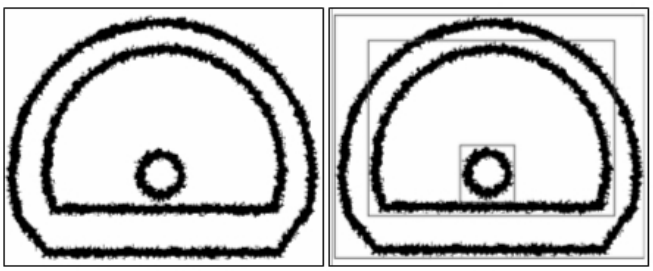


FIG. 1.5 – (a) symbole (b) rectangles englobants

Dans cette sous-section, nous avons présenté notre méthode de détection de contours. Celle-ci repose sur un marquage des composantes connexes par propagation. Ce marquage, dit séquentiel, est particulièrement adapté pour la détection de contours. Cette dernière s'effectue alors par recherche, durant le marquage, des pixels fonds connexes à la composante. De cette manière, les contours détectés pour chacune des composantes sont connexes. Cependant, cette méthode ne garantit pas d'obtenir une détection précise des contours. En effet, elle est basée sur une analyse du 8 voisinage, ce qui introduit des pixels dits redondants. Nous introduisons ces aspects dans la sous-section suivante en présentant notre méthode d'affinage.

1.2.3 Affinage des contours

Les contours détectés précédemment peuvent être composés de pixels que nous qualifions de redondants. Ces derniers se répartissent en deux classes : les barbules et les sur-épaisseurs. Les figures (1.6) (a) et (b) en donnent des exemples. Les barbules correspondent à l'apparition de courts segments connectés aux contours extraits. Les sur-épaisseurs elles, correspondent à la présence de pixels contours connectés à au moins deux pixels contours connexes dans leur voisinage. Ces pixels redondants sont issus de la présence de renfoncements des contours sur les bords des composantes. Leur présence n'est donc pas spécifique à notre approche mais inhérente au problème de l'extraction de contours. La figure (1.7) donne un exemple de bord d'une composante (a) et le résultat de sa détection de contours (b).

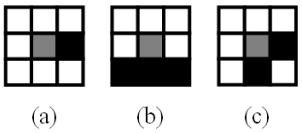


FIG. 1.6 – (a) barbule (b) sur-épaisseur (c) sur-épaisseur $N_c = 2$ $N_{cc} = 1$

FIG. 1.7 – (a) bords d'une composante (b) contours détectés

De façon à supprimer ces contours redondants nous employons une méthode d'affinage. Celle-ci est détaillée dans le pseudo-algorithme (1.2.2). Elle calcule, pour chacun des pixels contours (c_k), son nombre de croisements (N_c) [Rutovitz 66] avec les autres pixels contours de même label dans le voisinage (équation (1.1)). Le calcul de (N_c) permet de détecter les deux classes de pixels redondants (barbule et sur-épaisseur). En effet, ceux-ci sont caractérisées par par un (N_c) égal à 1.

Pseudo-algorithme 1.2.2: AFFINAGE($contours, image, -label$)

$modification \leftarrow$ **vrai**
tant que modification est **vrai**
{
$modification \leftarrow$ **faux**
pour chaque c_k de *contours*
faire {
$[N_c, N_{cc}] \leftarrow$ CROISEMENTS($c_k, -label$)
si $(N_c - N_{cc}) == 1$
alors {
supprimer c_k dans *contours*
marquer c_k dans *image* comme $fond$
$modification \leftarrow$ **vrai**
}
}
}

retourner (*contours*)

$$C_n = \begin{array}{|c|c|c|}\hline c_3 & c_2 & c_1 \\ \hline c_4 & c & c_{0,8} \\ \hline c_5 & c_6 & c_7 \\ \hline \end{array} \qquad N_c = \frac{1}{2} \times \sum_{i=0}^{7} |c_{i+1} - c_i| \tag{1.1}$$

$$N_{cc} = \sum_{i=0,2,4,6} (c_i \times c_{i+2}) \times (1 - c_{i+1}) \tag{1.2}$$

Le calcul du nombre de croisements (N_c) permet de filtrer les pixels barbules, néanmoins il n'est pas suffisant pour filtrer l'intégralité des pixels contours de sur-épaisseur. En effet, le calcul de (N_c) ne tient compte que des relations de 4-connexité pour déterminer le nombre de croisements. Certains pixels de sur-épaisseur présentant une relation de 8-connexité entre leur voisins (figure (1.6) (c)) pourraient également être filtrés. Afin de détecter ces relations de 8-connexité nous calculons en simultané de (N_c) le nombre de croisements connexes (N_{cc}). Celui-ci est défini dans l'équation (1.2). Il est alors soustrait à (N_c) dans notre méthode (pseudo-algorithme (1.2.2)) afin de filtrer l'intégralité des pixels contours de sur-épaisseur.

Notre méthode d'affinage parcourt ensuite les tableaux (*contours*) obtenus à l'issue de l'étape de détection dans le but de calculer (N_c) et (N_{cc}) pour chacun des pixels. Elle supprime alors itérativement tous les pixels contours répondant aux critères ($N_c - N_{cc} = 1$). Ces contours sont ensuite re-labellisés comme pixels fonds sur l'image. Notre méthode est itérative : toute suppression d'un pixel contour impose un nouveau parcours du tableau (*contours*). Néanmoins le nombre d'itération nécessaire est faible[9] car il dépend en grande partie de la taille des barbules présentes dans les contours. De cette façon, notre méthode filtre tous les pixels contours en extrémité ou possédant au moins deux pixels contours voisins connectés. Elle affine donc les contours tout en les corrigeant, et cela sans briser leur connexité. La figure (1.8) donne un exemple de contours affinés (b) des contours détectés (a).

[9] 4 pour l'exemple de la figure (1.7).

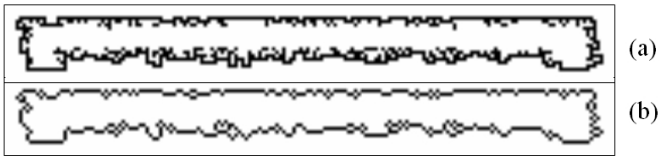


FIG. 1.8 – (a) contours détectés (b) contours affinés

Dans cette sous-section nous avons présenté notre méthode d'affinage des contours. Celle-ci exploite les contours détectés durant notre première étape afin de supprimer les contours redondants. Ces derniers se composent de pixels barbules et de sur-épaisseur. Nous les filtrons au travers d'un algorithme itératif par calcul de deux nombres (N_c) et (N_{cc}). De cette façon, notre méthode affine les contours tout en les corrigeant et cela sans briser leur connexité. Cependant ces contours affinés ne sont pas chaînés. Nous présentons comment nous chaînons ces contours durant notre étape de suivi dans la sous-section suivante.

1.2.4 Suivi des contours

Suite à la détection puis à l'affinage des contours, nous procédons à leur suivi. La principale problématique de ce suivi est la présence de boucles dans les contours (figure (1.9) (a)). Ces boucles sont issues de la présence de barbules et d'occlusions 8-connexe du fond aux abords des composantes. Leur présence n'est donc pas spécifique à notre approche mais inhérente au problème de l'extraction de contours. Elle implique que plusieurs suivis différents peuvent être effectués à partir des mêmes contours affinés d'une composante. Deux stratégies principales sont alors possibles : suivre les contours les plus internes ou les plus externes à la composante (figure (1.9) (b)). Chacune de ces deux stratégies présente des avantages et inconvénients. Le suivi des contours les plus internes fournit de meilleurs descripteurs. En effet, le suivi le plus externe englobe les barbules de la composante ce qui rend les contours extraits d'avantage bruités. En contrepartie, le suivi des contours les plus internes peut poser des problèmes de segmentation de la composante lorsque celle-ci comporte des traits de faibles épaisseurs et diagonaux. La figure (c) en donne un exemple.

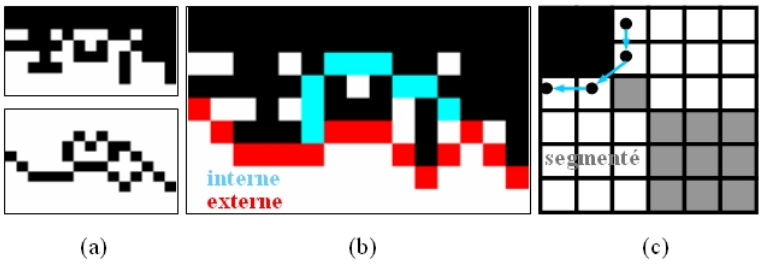


FIG. 1.9 – (a) boucles de contours (b) suivi interne/externe (c) segmentation

Pour notre méthode nous avons adopté un suivi des contours les plus internes. Cependant, comme nous l'avons illustré ce type de suivi peut poser des problèmes de segmentation. Néanmoins, ce cas de figure n'intervient que pour les composantes de très faibles épaisseurs, et donc dans le cas des images de faibles résolutions. Ce type d'image constitue une minorité des images de document, ce qui nous a encouragé à adopter cette stratégie.

Notre méthode de suivi est détaillée dans le pseudo-algorithme (1.2.3). Celle-ci est comparable à la méthode de suivi de contours de [Black 81]. Cependant, contrairement à celle-ci notre méthode repose sur l'utilisation des contours préalablement détectés et affinés. De même, elle adopte un suivi trigonométrique (et non anti-trigonométrique) de façon à chaîner les contours les plus internes aux composantes. La figure (1.10) en illustre le principe. Notre méthode utilise comme point contour d'entrée le premier point du tableau *contours* obtenu à l'issue des étapes de détection et de filtrage. Elle suit par la suite itérativement les points contours de même label à partir de deux points (*précédent*) et (*suivant*). Ce suivi s'effectue dans les 8 directions de [Freeman 61] dans le sens trigonométrique (de 0 à 7). La direction de départ correspond à l'opposé de la direction précédemment empruntée pour chaîner le point (*précédent*). Notre méthode n'effectue qu'un seul suivi des contours, ce que veut dire que seuls les contours englobant[10] la composante sont chaînés. Les contours correspondant aux occlusions de la composante sont ignorés alors que ceux-ci sont présents dans le tableau *contours* et marqués sur l'image. Nous considérons dans notre approche que ces contours appartiennent aux occlusions et ne doivent pas être extraits à partir des composantes. La figure (1.11) donne un exemple de résultat de suivi (b) de l'image (a).

Pseudo-algorithme 1.2.3: SUIVI($contours, image, -label$)

départ $\leftarrow$ *contours*[0]
insérer *départ* dans *chaîne*
marquer *départ* comme $fond$ dans *image*
précédent $\leftarrow$ *départ* *orientation* $\leftarrow 4$
faire {
  orientation $\leftarrow$ OPPOSÉ(*orientation*)
  pour chaque d_i dans le sens trigonométrique à partir de *orientation*
  faire {
    orientation $\leftarrow d_i$
    suivant $\leftarrow$ VOISIN(*précédent*, d_i)
    si *suivant* est de $-label$ dans *image*
    alors {
      insérer *suivant* dans *chaîne*
      marquer *suivant* comme $fond$ dans *image*
      stop pour chaque
  précédent $\leftarrow$ *suivant*
tant que *suivant* n'est pas connexe à *départ* **ou** taille de *chaîne* < 3
retourner (*chaîne*)

[10] Nous reportons le lecteur page **??** sur ces aspects.

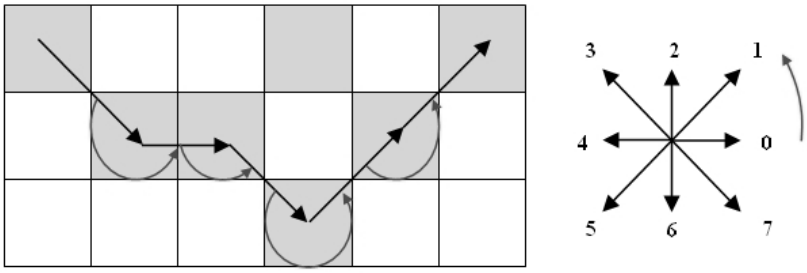


FIG. 1.10 – Suivi trigonométrique des contours

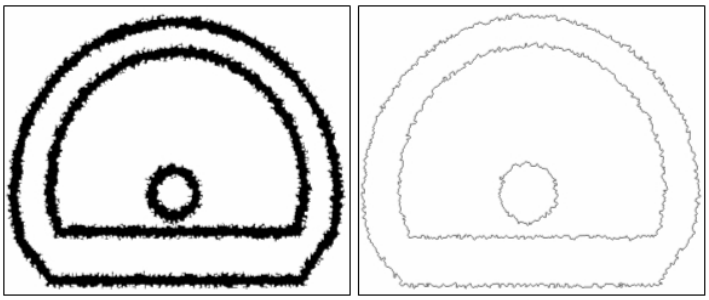


FIG. 1.11 – (a) image (b) contours suivis

Basé sur les contours détectés a priori, notre méthode de suivi permet une extraction robuste des contours. Les figures (1.12) illustre cette robustesse en comparant des résultats de suivi par la méthode de [Black 81] (b) et notre méthode (c). La figure (d) compare directement ces résultats : les contours extraits par la méthode de [Black 81] sont d'avantage bruités et redondants. Cette robustesse de notre méthode est assurée par la détection et l'affinage préliminaire des contours. En effet, ceci permet de baliser les contours de façon à guider a priori la méthode de suivi.

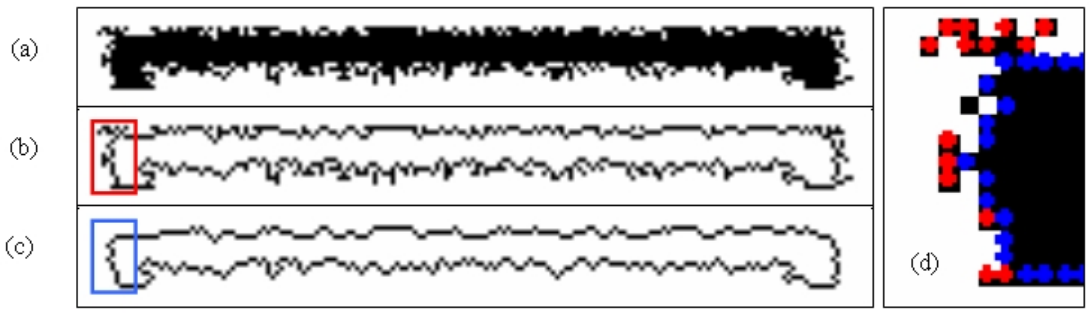


FIG. 1.12 – (a) image (b) suivi de contours (c) notre méthode (d) comparaison

1.2.5 Conclusion

Dans cette section, nous avons présenté nos opérateurs d'extraction de primitives graphique par approche contour. Ceux-ci reposent sur un marquage préliminaire des composantes connexes. À partir de ce marquage les contours sont extraits en trois étapes : détection, affinage et suivi. Basé sur ces trois étapes, nos opérateurs permettent une extraction robuste des contours. Cette robustesse est assurée par la détection et l'affinage préliminaire des contours avant le suivi. En effet, ceci permet de "baliser" les contours à extraire de façon à guider a priori l'étape de suivi.

Cependant, notre méthode est de complexité supérieure aux méthodes de suivi de contours de la littérature. En effet, nous utilisons deux étapes supplémentaires pour le marquage et l'affinage. La première nécessite une analyse du 8-voisinage de chacun des pixels formes durant le parcours de l'image. La seconde parcourt itérativement chaque tableau de contours des composantes jusqu'à suppression des pixels redondants. Ce gain en complexité se justifie par celui de robustesse des contours extraits. Le choix entre notre méthode et celles de la littérature sera donc fonction du compromis robustesse/compléxité souhaité. Dans la sous-section suivante, nous présentons comment nous étendons l'extraction de nos primitives graphiques via des opérateurs basés sur une approche région.

1.3 Extraction par approche région

1.3.1 Introduction

Au cours de la section précédente, nous avons présenté nos opérateurs d'extraction de primitives graphiques par approche contour. Comme présenté en introduction de ce chapitre, nos contributions visent ici à préparer la mise en oeuvre de notre approche de reconstruction d'objets présentée dans le chapitre suivant. Dans cette optique, nous souhaitons définir des opérateurs permettant différentes représentations des objets graphiques. Nous présentons dans cette section des opérateurs basés sur une approche région. Ces derniers extraient des graphes[11] décrivant différentes relations entre des régions (composantes et occlusions) comme le voisinage, l'inclusion, ... Ceux-ci sont ainsi complémentaires aux opérateurs par approche contour présentés précédemment en raison de leur différence de niveaux d'extraction.

Au cours du chapitre (**??**) nous avons présenté une étude bibliographique sur les méthodes[12] par approche région. Nous avons illustré qu'elles reposaient tout d'abord sur le marquage de composantes connexes. Nous avons ensuite montré que trois relations principales pouvaient être construites entre les composantes marquées : voisinage, inclusion et sous contraintes de distance.

[11] Nous reportons le lecteur à l'Annexe A pour une introduction sur les graphes.
[12] Nous reportons le lecteur page **??** sur ces méthodes.

Dans cette section, nous présentons notre approche pour la construction de graphes de régions. Le caractère original de cette approche réside dans la combinaison des relations entre les régions afin de multiplier les modèles de représentation. Nous avons pour cela utilisé différentes méthodes usuelles de la littérature pour l'extraction des régions et la construction de leurs relations. La figure (1.13) illustre le principe de notre approche. Celle-ci repose tout d'abord sur une méthode de marquage de composantes connexes et d'extraction d'occlusions. Nous exploitons ensuite les composantes et occlusions pour la construction de trois types de graphe : d'inclusion, de voisinage et sous contraintes de distance. Ces graphes sont ensuite combinés afin de produire des graphes que nous qualifions, dans le cadre de notre approche, d'hybrides.

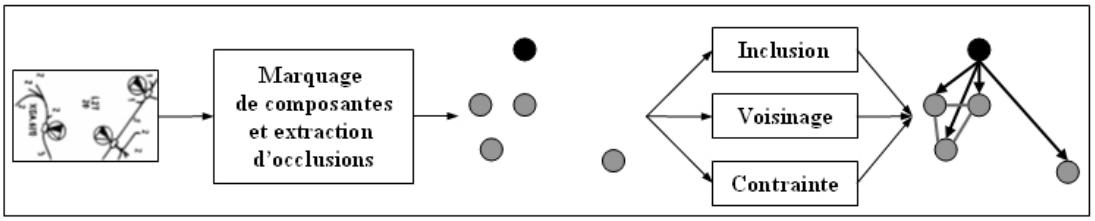


FIG. 1.13 – Construction de graphes de régions hybrides

Dans les sous-sections suivantes nous présentons tout d'abord les différentes méthodes usuelles[13] que nous employons dans notre approche. La sous-section (1.3.2) présente notre méthode pour le marquage de composantes connexes et l'extraction d'occlusions. Nous présentons ensuite dans les sous-sections (1.3.3), (1.3.4) et (1.3.5) nos méthodes pour la construction des graphes d'inclusion, de voisinage et sous contraintes de distance. Dans la sous-section (1.3.6) nous présentons notre méthode pour la construction de graphes hybrides. Finalement dans la sous-section (1.3.7) nous concluons.

1.3.2 Marquage de composantes et extraction d'occlusions

Au cours du chapitre (**??**) nous avons présenté une étude bibliographique sur les méthodes[14] de marquage de composantes connexes. Nous avons montré que celles-ci pouvaient opérer par propagation, par balayage de lignes et par suivi de contours. Dans le cadre de nos opérateurs par approche région, nous avons décidé d'utiliser une méthode par propagation. En effet, nous avons déjà employé[15] cette dernière pour nos opérateurs d'extraction par approche contour. Notre choix est donc uniquement motivé ici par la ré-utilisabilité de notre opérateur.

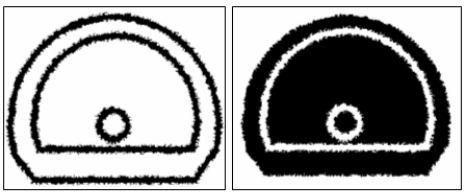
[13] Nous positionnons ici nos contributions majoritairement sur la combinaison des méthodes, non sur l'originalité des approches développées par ces méthodes.
[14] Nous reportons le lecteur page **??** sur ces méthodes
[15] Nous reportons le lecteur page 2 sur cette méthode.

Nous employons également une méthode dérivée de celle par propagation pour l'extraction des occlusions. Nous la présentons dans le pseudo-algorithme[16] (1.3.1) suivant. Celle-ci s'applique à la propagation au fond de l'image. Les variables $fond$ et $forme$ y sont inversées par rapport à la méthode initiale[17] de façon à éviter une inversion préliminaire de l'image. Notre méthode parcourt l'image et marque de façon séquentielle chacune des occlusions. Ces dernières sont par la suite ajoutées à un tableau (*occlusions*). Dans cette méthode, l'initialisation préalable des bords de l'image dans la couleur du fond implique systématiquement que la première composante marquée soit son fond principal[18]. Celui-ci est alors marqué sans ajout dans le tableau (*occlusions*). Les autres composantes marquées correspondent alors aux occlusions de l'image, elles sont ajoutées au tableau (*occlusions*). La figure (1.14) donne un exemple d'extraction d'occlusions (b) de l'image (a).

Pseudo-algorithme 1.3.1: OCCLUSIONS(*image*)

initialisation par $fond$ des bords de *image*
occlusions $\leftarrow \emptyset$
label $\leftarrow$ *ajouté* $\leftarrow -1$
tant que il existe un point d'entrée p_e du $fond$ dans *image* différent des bords
faire
- *pile* $\leftarrow \emptyset$
- *label* $\leftarrow$ *label* $- 1$
- ajouter p_e dans *pile*
- **pour chaque** p_i de *pile*
  - **faire**
    - marquer p_i dans *image* par *label*
    - **pour chaque** voisin p_v 8-connexe de p_i
      - **faire**
        - **si** p_v est du $fond$ et non marqué dans *image*
          - **alors**
            - ajouter p_v dans *pile*
            - marquer p_v dans *image* comme *ajouté*
- **si** *label* < -1
  - **alors** {ajouter $o =$ {*label*,} à *occlusions*

retourner (*image*, *occlusions*)



(a) (b)

FIG. 1.14 – (a) image (b) occlusions

[16] Cette méthode permet aussi le calcul de caractéristiques topologiques non présenté ici.
[17] Nous reportons le lecteur page 2 sur cette méthode.
[18] Nous reportons le lecteur page ?? sur cet aspect.

Dans cette sous-section, nous avons présenté nos méthodes pour le marquage de composantes connexes et l'extraction d'occlusions. Dans les sous-sections suivantes, nous présentons comment nous exploitons les composantes et occlusions extraites pour la construction de graphes d'inclusion, de voisinage et sous contraintes de distance.

1.3.3 Graphes d'inclusion

À partir des composantes et occlusions extraites précédemment, nous utilisons une première méthode pour construire des graphes d'inclusion. Ceux-ci seront par la suite exploités dans le cadre de la construction de nos graphes hybrides (sous-section (1.3.6)). Au cours du chapitre (**??**) nous avons présenté une étude bibliographique des méthodes[19] de construction de graphes d'inclusion. Nous avons illustré que les relations d'inclusion étaient obtenues par calcul des rapports d'englobement. Ces derniers se définissent à partir des rectangles englobants des composantes connexes comme l'illustre la figure (1.15) (a). La figure (b) donne un exemple de graphe d'inclusion basé sur cette relation.

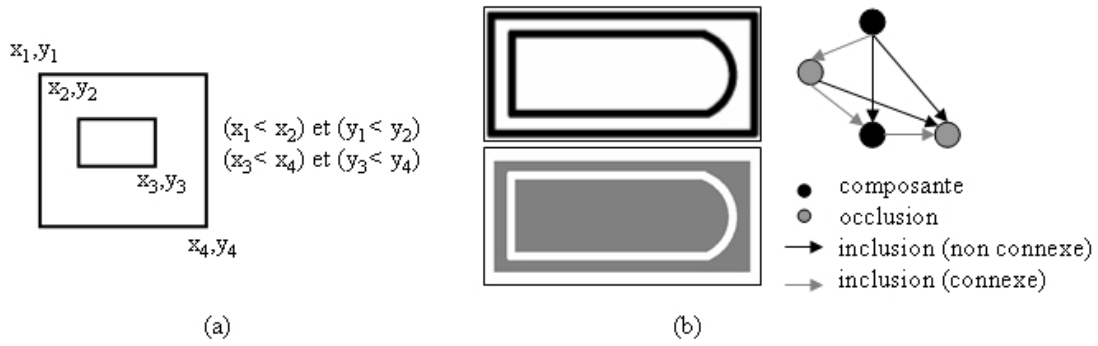


FIG. 1.15 – (a) rapport d'englobement (b) graphe d'inclusion

Dans le cadre de notre méthode, nous proposons une extension de cette approche de construction par rectangle englobant. En effet l'approche initiale ne permet pas de distinction entre les relations d'inclusion dites non connexes et connexes. Ces dernières concernent composantes et occlusions connexes deux à deux. La figure (1.15) (b) distinguent ces deux types de relation. La distinction entre ces deux relations constitue donc une information importante dans les graphes d'inclusion. Afin de procéder à cette distinction, nous avons étendu l'approche initiale par un parcours des rectangles englobants. Ces parcours permettent ainsi de déterminer les relations d'inclusion connexes. La figure (1.16) donne un exemple de graphe que nous construisons (c) à partir des composantes (a) et d'occlusions (b) d'un symbole. Nos relations d'inclusion relient alors de façon arborescente les composantes et leurs occlusions.

[19] Nous reportons le lecteur page **??** sur ces méthodes

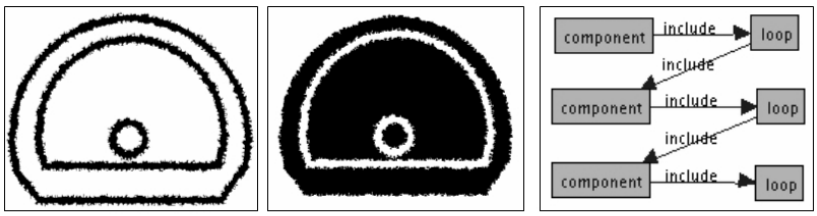


FIG. 1.16 – (a) composantes (b) occlusions (c) graphe d'inclusion

Notre méthode est présentée dans le pseudo-algorithme (1.3.2). Elle exploite en entrée les résultats de marquage d'une image (*composantes*,$image_c$) ainsi que ceux de l'extraction de ses occlusions (*occlusions*,$image_o$). Dans une première étape, un graphe (*inclusion*) est créé et défini comme orienté. Les noeuds de ce graphe sont créés à partir de chacune des composantes et occlusions de l'image. Les relations d'inclusion sont ensuite cherchées en deux étapes : entre les composantes et les occlusions, puis entre les occlusions et les composantes. Dans chacune de ces étapes, pour chaque composante (c_i) ou occlusion (o_i) donné(e) les relations d'inclusion sont recherchées avec toutes les (n_o) occlusions ou (n_c) composantes de l'image. En effet, une relation d'inclusion ne pouvant être définie qu'entre composantes et occlusions, il est alors inutile de parcourir l'ensemble des noeuds ($n_c + n_o$) du graphe (*inclusion*).

Pseudo-algorithme 1.3.2: INCLUSION($composantes, image_c, occlusions, image_o$)

définir *inclusion* comme orienté
$n_c \leftarrow$ taille de *composantes*
pour i variant de 1 **à** n_c
  faire ajouter un noeud attribué *composantes*[i] à *inclusion*
$n_o \leftarrow$ taille de *occlusions*
pour i variant de 1 **à** n_o
  faire ajouter un noeud attribué *occlusions*[i] à *inclusion*
pour i variant de 1 **à** n_c
  faire { $c_i \leftarrow$ lire attribut du noeud d'indice i de *inclusion*
  pour j variant de $n_c + 1$ **à** $n_c + n_o$
    faire { $o_i \leftarrow$ lire attribut du noeud d'indice j de *inclusion*
    si INCLU($c_i, image_c, o_i, image_o$)
      alors ajouter $a = \{c_i, o_i, inclu\}$ à *inclusion*
pour i variant de $n_c + 1$ **à** $n_c + n_o$
  faire { $o_i \leftarrow$ lire attribut du noeud d'indice i de *inclusion*
  pour j variant de 1 **à** n_c
    faire { $c_i \leftarrow$ lire attribut du noeud d'indice j de *inclusion*
    si INCLU($o_i, image_o, c_i, image_c$)
      alors ajouter $a = \{o_i, c_i, inclu\}$ à *inclusion*
retourner (*inclusion*)

La partie centrale de notre approche repose sur l'utilisation d'une méthode de test d'inclusion (*inclu*). Celle-ci parcourt les rectangles englobants des composantes et occlusions afin d'y rechercher les relations d'inclusion connexes. Nous détaillons cette méthode dans le pseudo-algorithme (1.3.3). La figure (1.17) l'illustre. Elle utilise en entrée deux composantes connexes (c_1) et (c_2) ainsi que leurs images marquées d'appartenance ($image_1$) et ($image_2$). La composante (c_1) est supposée englober la composante (c_2). Elle recherche alors si une relation de connexion existe entre (c_1) et (c_2) dans ($image_1$) et ($image_2$). Elle teste dans un premier temps, si une relation d'englobement existe entre les deux composantes (c_1) et (c_2). Dans un cas positif notre méthode parcourt alors l'axe $(x_{02}; y_{02})$ $(x_{02} + d_{x2}; y_{02})$ sur ($image_2$). Cet axe correspond au côté haut du rectangle englobant de (c_2). Durant ce parcours elle recherche le premier pixel labellisé ($label_2$). À partir des coordonnées de ce pixel, elle teste les labels des 3 pixels voisins selon les trois directions de [Freeman 61] {1,2,3} sur ($image_1$). Si un de ces pixels est labellisé ($label_1$) une relation d'inclusion peut être alors définie entre (c_1) et (c_2).

Pseudo-algorithme 1.3.3: INCLU$(c_1, image_1, c_2, image_2)$

$\{label_1, x_{01}, y_{01}, d_{x1}, d_{y1}\} \leftarrow$ CARACTÉRISTIQUES(c_1)
$\{label_2, x_{02}, y_{02}, d_{x2}, d_{y2}\} \leftarrow$ CARACTÉRISTIQUES(c_2)

si $(x_{01} < x_{02})$ **et** $(y_{01} < y_{02})$ **et** $(x_{01} + d_{x1} > x_{02} + d_{x2})$ **et** $(y_{01} + d_{y1} > y_{02} + d_{y2})$
alors
- **pour** x variant de x_{02} **à** $x_{02} + d_{x2}$
  - **faire**
    - $v_2 \leftarrow$ LIRE$(image_2, x, y_{02})$
    - **si** $v_2 = label_2$
      - **alors**
        - **pour chaque** voisin $\{x_v, y_v\}$ au dessus de $\{x, y_{02}\}$
          - **faire**
            - $v_1 \leftarrow$ LIRE$(image_1, x_v, y_v)$
            - **si** $v_1 = label_1$
              - **alors retourner** (**vrai**)
        - **retourner** (**faux**)

sinon retourner (**faux**)

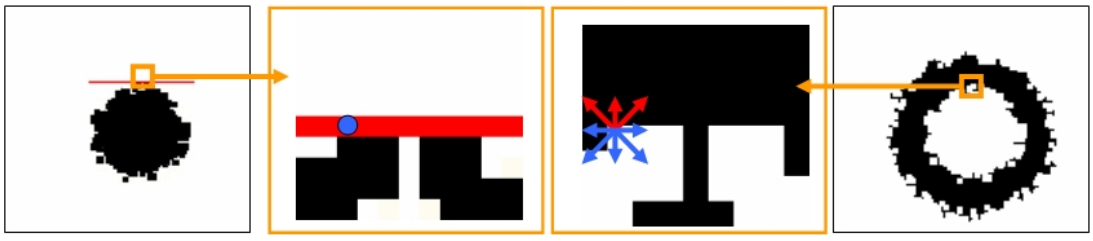


FIG. 1.17 – (a) c_2 (b) pixel de $label_2$ sur c_2 (c) voisinage analysé sur c_1 (d) c_1

Dans cette sous-section, nous avons présenté notre méthode pour la construction de graphes d'inclusion. Celle-ci emploie une approche à base de parcours des rectangles englobants pour la construction des relations d'inclusion dites connexes. Ces dernières concernent composantes et occlusions connexes deux à deux. Les relations d'inclusion relient ainsi de façon arborescente les composantes et de leurs occlusions. Dans la sous-section suivante, nous présentons une autre alternative pour construire des graphes de régions basés sur les relations de voisinage.

1.3.4 Graphes de voisinage

Nous utilisons une méthode de construction complémentaire à celle des graphes d'inclusion afin de construire des graphes de voisinage. Ceux-ci seront par la suite exploités pour la construction de nos graphes hybrides (sous-section (1.3.6)). Au cours du chapitre (**??**) nous avons présenté une étude bibliographique des méthodes[20] de construction de graphes de voisinage. Nous avons illustré que celles-ci sont basées principalement sur les diagrammes de Voronoi. Nous avons également montré que les relations de voisinage sont non orientées et permettent de construire des graphes dits planaires[21]. La figure (1.18) donne un exemple de graphe de voisinage (b) de l'image (a).

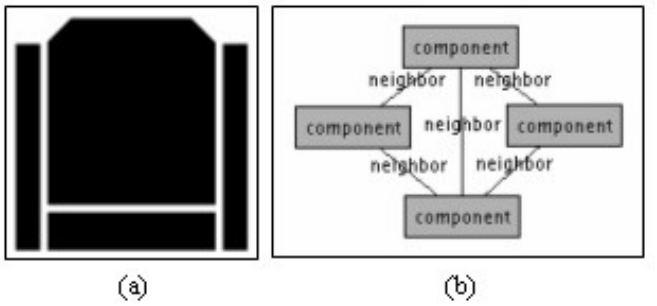


FIG. 1.18 – (a) image (b) graphe de voisinage

Notre méthode de construction de graphes de voisinage est basée sur une approche à base d'extension des contours des composantes. Cette approche constitue une alternative aux approches à base de diagrammes de Voronoi généralement utilisées dans la littérature. En effet, elle est élémentaire[22] à ces dernières pour la construction des relations de voisinage. Elle constitue cependant un complément naturel à notre méthode de marquage par propagation[23] utilisée dans nos opérateurs par approche contour. Notre choix est donc uniquement motivé ici par la ré-utilisabilité de notre méthode.

[20] Nous reportons le lecteur page **??** sur ces méthodes.
[21] Nous reportons le lecteur à l'Annexe A sur cet aspect.
[22] Les relations extraites sont identiques mais la complexité en est supérieure.
[23] Nous renvoyons le lecteur page 2 pour une présentation de cette méthode.

Le pseudo-algorithme (1.3.4) détaille le principe de notre méthode. Celle-ci est basée sur les résultats de marquage préliminaire d'une image. Elle étend ensuite progressivement les pixels contours par propagation de leur label aux pixels voisins fonds. Cette extension est alternative, c'est à dire que l'ordre des contours propagés de chacune des composantes est inversé à chaque cycle. Ceci permet d'équilibrer la vitesse de propagation entre les différents contours de chacune des composantes. Le processus d'extension est bloqué lorsque les contours étendus rencontrent des zones déjà marquées (autres contours étendus, composantes) ou des zones nulles (en dehors de l'image). La rencontre d'autres contours étendus dans le voisinage provoque la sauvegarde et le marquage des contours comme points frontières. La figure (1.19) donne un exemple de quatre étapes successives d'extension de contours (b) de l'image (a). La figure (1.20) donne un exemple de représentation des points frontières (b) extraits de l'image (a).

Pseudo-algorithme 1.3.4: EXTENSION()

globale $image, tableau_c, frontières$

fonction ÉTENDRE($contours$)
$étendus \leftarrow \emptyset$
pour chaque p_i de $contours$
faire
- **pour chaque** voisin p_v 8-connexe de p_i appartenant à $image$ **faire**
  - **si** p_v est du $fond$ dans $image$
    - **alors**
      - ajouter p_v dans $étendus$
      - marquer p_v par label de $contours$ dans $image$
    - **sinon si** p_v est un point contour mais pas de $contours$
      - **alors**
        - ajouter p_v à $frontières$
        - marquer p_v comme $frontières$ dans $image$

si taille de $étendus$ est $\neq$ de 0
alors substituer $étendus$ à $contours$ dans $tableau_c$
sinon supprimer $contours$ dans $tableau_c$

fonction EXTENSION()
$sens \leftarrow$ **vrai**
tant que taille de $tableau_c$ est $\neq 0$
faire
- **si** $sens$ est **vrai**
  - **alors**
    - **pour chaque** $contours$ de $tableau_c$ de $début$ à fin
      - **faire** ÉTENDRE($contours$)
  - **sinon**
    - **pour chaque** $contours$ de $tableau_c$ de fin à $début$
      - **faire** ÉTENDRE($contours$)
- $sens \leftarrow$ **non** $sens$

principal
$image \leftarrow$ LECTURE($adresse$)
$tableau_c \leftarrow$ PROPAGATION($image$)
EXTENSION()

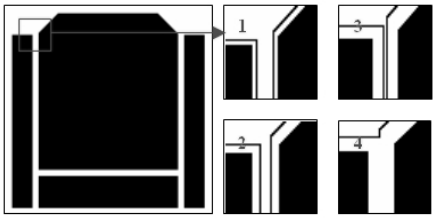


FIG. 1.19 – (a) image (b) extension des contours

FIG. 1.20 – (a) image (b) points frontières

Les relations de voisinage sont ensuite construites à partir du parcours des frontières. Chaque frontière correspond alors à un sous-ensemble de points frontières définissant une même relation de voisinage entre deux composantes. Le pseudo-algorithme (1.3.5) détaille le principe de cette construction. Le tableau des points frontières construit durant l'extension des contours est utilisé pour parcourir l'image. L'analyse du 8-voisinage des points frontières permet alors d'extraire les labels des contours étendus. Ces derniers définissent les relations de voisinage entre composantes car ils correspondent[24] aux négatifs de leurs labels. Durant ce parcours, les longueurs des différentes frontières ($longueur_f$) sont également calculées.

Pseudo-algorithme 1.3.5: VOISINAGE(*frontières*, *image*)

$longueur_f \leftarrow 1$
$voisins \leftarrow \emptyset$
pour chaque p_i de *frontières*
faire {
  $r_1 \leftarrow$ lire label de p_i sur *image*
  pour chaque voisin p_v contour 8-connexe de p_i dans *image*
  faire {
    $r_2 \leftarrow$ lire label de p_v sur *image*
    si $r_2 \neq r_1$
    alors {
      si $\{r_1 \cap r_2\}$ n'existe pas dans *voisins*
      alors ajouter $\{r_1 \cap r_2, longueur_f\}$ à *voisins*
      sinon incrémenter $taille_f$ de $\{r_1 \cap r_2\}$ dans *voisins*
    }
  }
}

[24] Nous reportons le lecteur page 3 sur ces aspects.

À partir des relations de voisinage et des labels des composantes connexes un graphe de voisinage peut être construit. Les noeuds de ce graphe sont construits à partir des composantes extraites à l'issue de notre marquage. Les arcs sont construits à partir des relations de voisinage obtenues à l'issue du parcours des frontières. Chaque arc construit est également attribué d'une distance euclidienne (d) séparant les centres de gravité des composantes qu'il relie. La figure (1.21) donne un exemple de graphe de voisinage construit (c) à partir de l'image des occlusions (b) extraite de l'image (a).

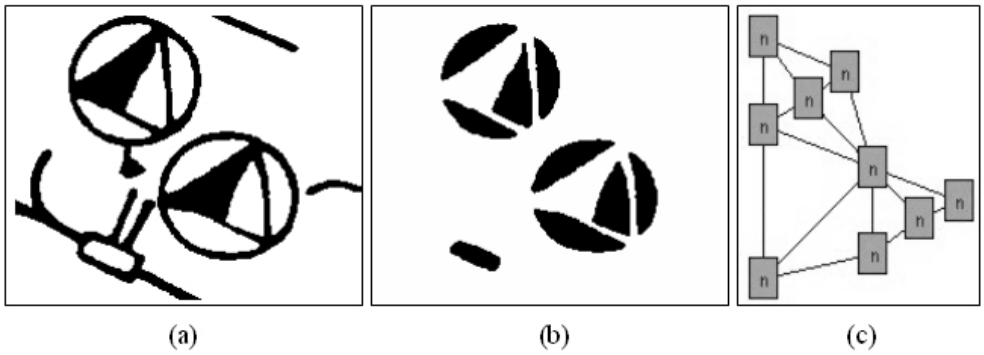


FIG. 1.21 – (a) image (b) occlusions (c) graphe de voisinage

Les relations de voisinage entre composantes peuvent être dans certains cas ambiguës. En effet, une faible variation de la position d'une composante peut modifier ces relations. La figure (1.22) en donne un exemple. Dans cet exemple, la variation de position du (•) inclu dans le (C) modifie sa relation de voisinage avec le (•) extérieur au (C). Cette ambiguïté n'est pas propre à notre approche mais générale au problème de la détermination des relations de voisinage entre composantes.

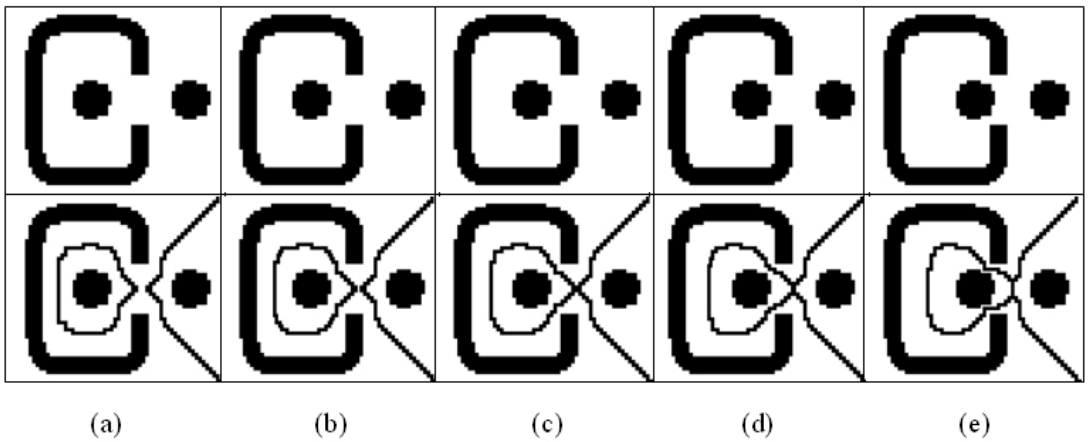


FIG. 1.22 – (a) non voisin (b) presque voisin (c) voisin ? (d) presque pas voisin (e) voisin

Dans notre approche, cette ambiguïté de voisinage se traduit comme un problème de filtrage des frontières minoritaires. Une frontière minoritaire est de longueur négligeable à la vue des autres frontières existantes entre composantes. La figure (1.23) en donne un exemple. Dans cet exemple, des points frontières sont extraits (b) à partir d'une image (a). Le parcours des points frontières (c) indique une connexion de 1 pixel entre les contours étendus des composantes haute et basse respectivement labellisées 1 et 4. La frontière entre ces deux composantes peut donc être considérée comme minoritaire. Les relations de voisinage construites (d) incluent donc une relation entre les composantes 1 et 4 résultant du parcours de cette frontière minoritaire.

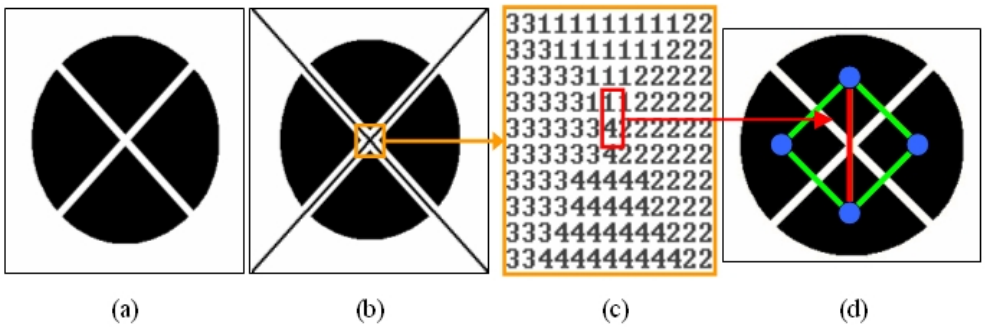


FIG. 1.23 – Filtrage des frontières minoritaires
(a) image (b) points frontières (c) parcours (d) relations de voisinage

Notre méthode de filtrage des frontières minoritaires exploite un traitement d'histogramme. Nous la présentons dans le pseudo-algorithme (1.3.6). La figure (1.24) l'illustre à travers un exemple. Un histogramme ($longueur_h$) des longueurs des frontières est tout d'abord créé. Les abscisses de cet histogramme représentent alors les longueurs des frontières. Les ordonnées représentent les nombres de frontière pour chacune des longueurs. À partir de cet histogramme un tableau (*positions*) des limites inférieure et supérieure des séquences de longueurs non valides est construit. Une longueur non valide (l) est telle que ($longueur_h[l] = 0$). Les limites inférieure et supérieure correspondent aux longueurs valides ($longueur_h[l] \neq 0$) délimitant les séquences de longueurs non valides. Un tableau (*rapports*) est ensuite calculé à partir des différentes limites inférieure et supérieure du tableau (*positions*). Le seuil est alors obtenu par recherche du rapport maximum. Ce rapport maximum correspond alors à la disproportion la plus importante entre deux longueurs de frontière sur l'image. À partir de ce rapport la méthode détermine le seuil pour le filtrage des frontières. Ce seuil correspond à la limite inférieure dans le tableau (*positions*) correspondant au rapport maximum trouvé. Pour un rapport d'indice (i) le seuil sera donc obtenu à l'indice ($(i \times 2) - 1$) dans le tableau (*positions*).

Pseudo-algorithme 1.3.6: FILTRAGE(*voisins*)

$longueur_h \leftarrow \emptyset$
$longueur_{max} \leftarrow 0$
pour chaque $v_i = \{r_1 \cap r_2, longueur_f\}$ de *voisins*
faire incrémenter $longueur_h[longueur_f]$
si $longueur_f > longueur_{max}$
alors $longueur_{max} \leftarrow longueur_f$

début $\leftarrow 1$
tant que $longueur_h[début] == 0$ **et** $début < longueur_{max}$
faire incrémenter *début*

positions $\leftarrow \emptyset$
$j \leftarrow 0$
pour i variant de *début* **à** $longueur_{max} - 1$
faire **si** $longueur_h[i+1] == 0$
alors incrémenter j
$positions[j] = i$
tant que $longueur_h[i+1] == 0$ **et** $i < longueur_{max}$
faire incrémenter i
incrémenter j
$positions[j] = i + 1$

rapports $\leftarrow \emptyset$
pour i variant de 1 **à** taille de *positions* - 1 par pas de 2
faire $rapports[\frac{i+1}{2}] \leftarrow \frac{positions[i+1]}{positions[i]}$

$rapport_{max} \leftarrow 0$
$position_{max} \leftarrow 0$
pour chaque i variant de 1 à taille de *rapports*
faire **si** $rapports[i] > rapport_{max}$
alors $rapport_{max} \leftarrow rapports[i]$
$position_{max} \leftarrow (i \times 2) - 1$

seuil $\leftarrow -1$
si $position_{max} \neq 0$
alors $seuil \leftarrow positions[position_{max}]$

retourner (*seuil*)

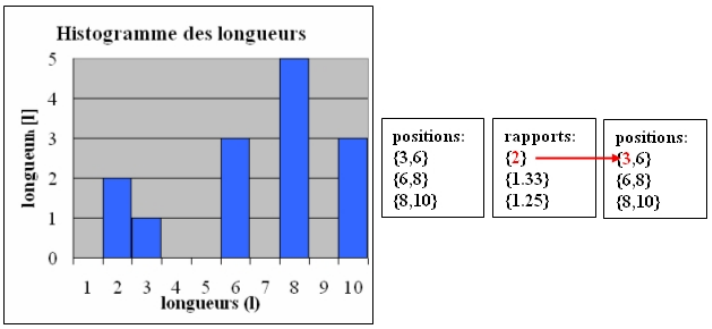


FIG. 1.24 – Exemple de traitement d'histogramme

Cette méthode permet de filtrer efficacement les frontières minoritaires. En effet, celles-ci sont caractérisées par leurs faibles longueurs à la vue des autres frontières présentes sur l'image. La recherche du rapport maximum entre ces longueurs permet alors de déterminer le seuil de filtrage. La figure (1.25) donne des exemples de graphe de voisinage construits en utilisant (d) ou non (c) une étape de filtrage des frontières minoritaires. Les figures (a) et (b) présentent respectivement une image de symbole et la représentation des points frontières de l'image de ses occlusions. Dans cet exemple, l'étape de filtrage des frontières minoritaires de l'image (b) permet la suppression dans le graphe (d) des relations de voisinage diagonales entre les composantes des zones haute et basse de l'image.

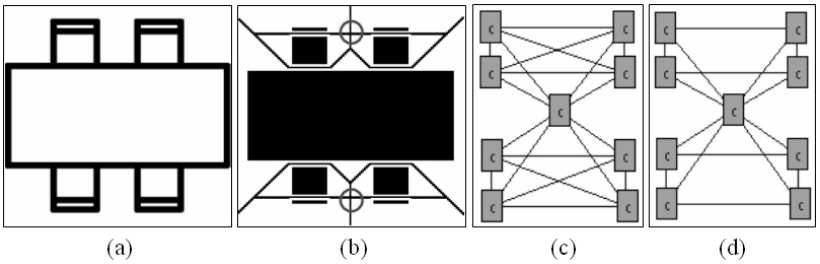


FIG. 1.25 – (a) symbole (b) points frontières[25](c) sans filtrage (d) avec filtrage

Dans cette sous-section, nous avons présenté notre méthode de construction de graphes de voisinage. Cette dernière est basée sur l'utilisation préliminaire de notre méthode de marquage de composantes connexes par propagation. Elle emploie ensuite une extension des contours dans le but de construire les points frontières entre les composantes. Ces derniers sont ainsi parcourus afin d'extraire les relations de voisinage. Une méthode de filtrage est utilisée afin de supprimer les relations de voisinage induites par les frontières dites minoritaires. Dans la sous-section suivante, nous présentons une dernière méthode de construction de graphes de régions sous contraintes de distance.

[25] Les cercles indiquent les frontières minoritaires.

1.3.5 Graphes sous contraintes de distance

Nous utilisons une dernière méthode de construction complémentaire à celles des graphes d'inclusion et de voisinage. Cette dernière permet, en effet, de construire des graphes sous contraintes de distance. Ceux-ci seront par la suite exploités dans le cadre de la construction de nos graphes hybrides (sous-section (1.3.6)). Au cours du chapitre (**??**) nous avons présenté une étude bibliographique des méthodes[26] de construction de graphes sous contraintes. Nous avons illustré que celles-ci emploient des contraintes variées (pavage, projection, alignement, ...) dans le but de construire des relations entre les composantes.

Dans le cadre de notre méthode, nous nous sommes basés sur des contraintes de distance. Celles-ci s'appliquent à des graphes représentant des composantes et les distances qui les séparent. Ces distances sont calculées à partir des centres de gravité des composantes. Des contraintes sont alors appliquées sur les distances afin de grouper les composantes selon leur proximité. Pour ce faire nous avons mis en oeuvre une méthode élémentaire de construction. Celle-ci se déroule en deux étapes : une de construction du graphe complet des composantes, et une autre de filtrage des arcs de ce graphe par application des contraintes de distance.

Le graphe complet des composantes est un graphe dans lequel chaque noeud (ou composante) est connecté à tous les autres. Les arcs décrivent alors les relations d'interconnexion entre toutes les composantes de l'image. Notre méthode de construction du graphe complet est détaillée dans le pseudo-algorithme (1.3.7). Celle-ci est basée sur l'utilisation préliminaire de nos méthodes de marquage de composantes et d'extraction d'occlusions (sous-section (1.3.2)).

Pseudo-algorithme 1.3.7: COMPLET(*composantes*)

$complet \leftarrow \emptyset$ $\quad n \leftarrow$ taille de *composantes*
pour i variant de 1 à n
faire ajouter un noeud c_i attribué *composantes*$[i]$ à *complet*

pour i variant de 1 **à** n
faire
- $c_1 \leftarrow$ NOEUD$(complet, i)$
- $\{cg_{x1}, cg_{y1}\} \leftarrow$ CARACTÉRISTIQUES(c_1)
- **pour** j variant de i+1 **à** taille de *complet*
  - **faire**
    - $c_2 \leftarrow$ NOEUD$(complet, j)$
    - $\{cg_{x2}, cg_{y2}\} \leftarrow$ CARACTÉRISTIQUES(c_2)
    - $d \leftarrow \sqrt{(cg_{x1} - cg_{x2})^2 + (cg_{y1} - cg_{y2})^2}$
    - ajouter un arc à *complet* entre c_1 et c_2 attribué d

[26] Nous reportons le lecteur page **??** sur ces méthodes.

Notre méthode construit dans un premier temps un graphe (*complet*) d'ordre (n) ou chaque noeud (c_i) représente une composante. Elle parcourt ensuite chacun de ces noeuds (c_i). Pour chaque noeud d'indice (i) donné, des arcs sont construits entre ce noeud et les $(i+1)$ à (n) noeuds suivants de (*complet*). Chacun des arcs est attribué d'une distance euclidienne (d) calculée à partir des centres de gravité des composantes qu'il relie. Chaque noeud du graphe (*complet*) est donc de degré $(n-1)$, et (*complet*) est donc un p-graphe tel que $(p = n-1)$. Le nombre d'arcs de (*complet*) est donc égal à $((n-1)!)$ et croît de façon factorielle en fonction du nombre de composantes (n) présentes sur l'image. La figure (1.26) (a) donne une représentation graphique d'un graphe complet de composantes au format SVG.

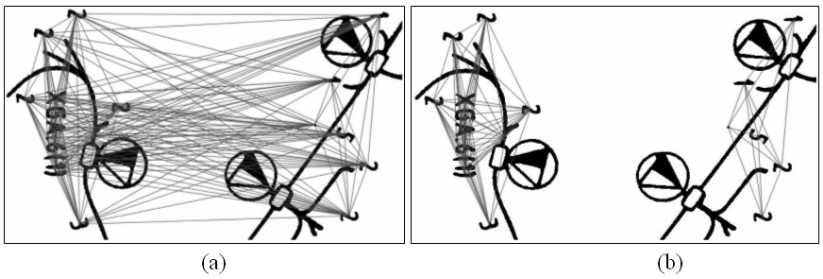


FIG. 1.26 – (a) graphe complet (b) graphe sous contraintes de distance

Dans une deuxième étape nous utilisons une méthode de filtrage des arcs sous contraintes de distance. Celle-ci est présentée dans le pseudo-algorithme (1.3.8). Elle utilise en entrée un ensemble de deux contraintes $\{c_{d1}, c_{d2}\}$. Celles-ci sont définies en paramètre de notre méthode par un utilisateur en fonction des images à traiter. À partir de ces contraintes trois types de filtrage sont possibles : passe-bas, passe-haut et passe-bande. Pour appliquer ces filtrages, nous utilisons deux variables $\{s_1, s_2\}$ définies selon un argument (*mode*) paramètre de la méthode et des deux contraintes $\{c_{d1}, c_{d2}\}$. La méthode de filtrage supprime alors les arcs dans le graphe complet attribués d'une distance (d) tel que $(s_1 \leq d \leq s_2)$. La figure (1.26) (b) donne un exemple de résultat de filtrage passe bas au format SVG du graphe complet (a).

Pseudo-algorithme 1.3.8: FILTRAGE$(graphe, c_{d1}, c_{d2}, mode)$

$c_{dmax} \leftarrow$ distance maximum de *graphe*
si *mode* est passe bas **alors** $s_1 \leftarrow c_{d1} \quad s_2 \leftarrow c_{dmax}$
si *mode* est passe haut **alors** $s_1 \leftarrow 0 \quad s_2 \leftarrow c_{d2}$
si *mode* est passe bande **alors** $s_1 \leftarrow c_{d1} \quad s_2 \leftarrow c_{d2}$
pour chaque arc a_i de *graphe*
faire { **si** a_i est attribué d'une distance d
  alors si $s_1 \leq d \leq s_2$
  alors { supprimer a_i dans graphe
  décrémenter i
retourner (*graphe*)

Dans cette sous-section, nous avons présenté notre méthode pour la construction de graphes sous contraintes de distance. Celle-ci est basée sur un traitement en deux étapes : construction du graphe complet et filtrage des arcs. Dans la sous section suivante, nous illustrons comment nous combinons cette méthode avec celles précédemment présentées dans cette section pour la construction de graphes hybrides.

1.3.6 Graphes hybrides

Comme présenté dans la sous-section d'introduction (1.3.1) le caractère original de notre approche réside dans la combinaison des relations entre régions afin de multiplier les modèles de représentation. Dans les sous-sections précédentes nous avons tout d'abord présenté nos méthodes pour le marquage des composantes connexes et l'extraction d'occlusions. Nous avons ensuite présenté nos méthodes pour la construction des graphes d'inclusion, de voisinage et sous contraintes de distance. Dans cette sous-section nous présentons comment nous combinons ces méthodes dans le but de construire des graphes hybrides. La figure (1.27) donne un exemple de graphe hybride (b) du symbole (a). Celui-ci combine des relations d'inclusion entre composantes et occlusions avec des relations de voisinage entre occlusions.

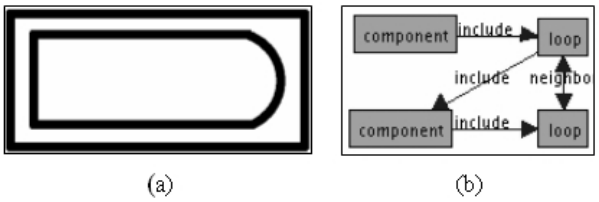


FIG. 1.27 – (a) symbole (b) graphe hybride

Notre méthode de construction de graphes hybrides est décrite dans le pseudo-algorithme (1.3.9). Elle est basée sur un principe de fusion des graphes (*inclusion*) et (*voisinage*). Dans une première étape, le graphe (*hybride*) est initialisé à partir du graphe (*inclusion*). En effet, ce dernier contient l'intégralité des composantes connexes de l'image (composantes et occlusions). Le problème de construction du graphe (*hybride*) se ramène par la suite à celui de la recopie des arcs du graphe (*voisinage*). Pour cela notre méthode parcourt chacun des arcs (a_i) de ce graphe de voisinage. Les noeuds de début (v_1) et de fin (v_2) de chacun de ces arcs sont comparés avec ceux (n_i) du graphe (*hybride*). Cette comparaison s'effectue sur la base d'un test d'égalité entre les caractéristiques topologiques (*topologie*) des composantes de chacun des noeuds. Les caractéristiques comparées correspondent à la fois aux centres de gravité (cg_x,cg_y) et aux rectangles englobants (x_0,y_0,dx,dy) afin de comparer l'ensemble de la topologie. Une fois les noeuds correspondants ($v_1^{'}$) ($v_2^{'}$) trouvés un arc de voisinage ($a_i^{'}$) attribué d'une distance euclidienne (d) recopiée de (a_i) est construit dans le graphe (*hybride*).

Pseudo-algorithme 1.3.9: HYBRIDE($inclusion, voisinage$)

hybride ← *inclusion*
pour chaque arc $a_i = \{v_1, v_2, \text{d}\}$ de *voisinage*
faire {
  $topologie_{v_1}$ ← CARACTÉRISTIQUES(v_1)
  $topologie_{v_2}$ ← CARACTÉRISTIQUES(v_2)
  $v_1' \leftarrow v_2' \leftarrow \emptyset$
  pour chaque noeud n_i de hybride
  faire {
    $topologie_i$ ← CARACTÉRISTIQUES(n_i)
    si $topologie_i = topologie_{v_1}$
    alors $v_1' \leftarrow v_1$
    sinon si $topologie_i = topologie_{v_2}$
    alors $v_2' \leftarrow v_2$
    sinon si $v_1' \neq \emptyset$ **et** $v_2' \neq$
    alors stoppe pour chaque
  }
  ajouter $a_i = \{v_1, v_2, \text{d}\}$ à *hybride*
}

retourner (*hybride*)

À partir de notre méthode, nous pouvons construire trois types différents de graphe hybride. La figure (1.28) en donne des exemples : il s'agit de celui basé sur les composantes de la forme (a), celui basé sur les occlusions (b) et celui basé sur les composantes de la forme et sur les occlusions (c). La construction du graphe hybride (c) est particulière car elle s'effectue en deux étapes. Dans la première étape, le graphe d'inclusion est fusionné avec le graphe de voisinage des composantes. Dans la seconde étape, le graphe hybride résultant de la première étape est assimilé à un graphe d'inclusion. Ce dernier est alors fusionné avec le graphe de voisinage des occlusions.

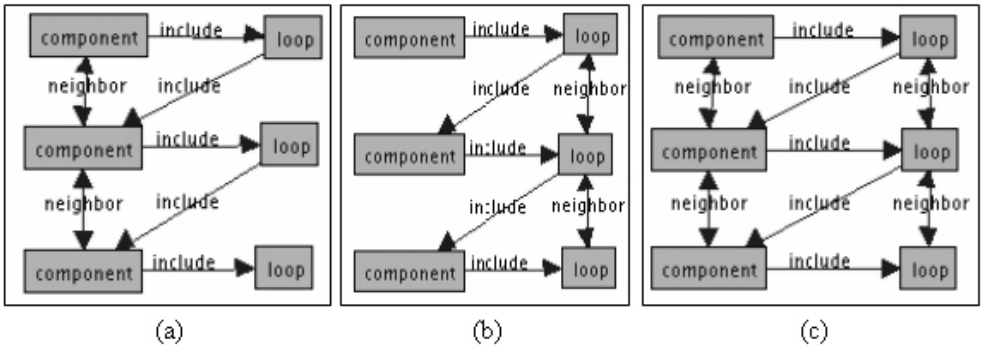


FIG. 1.28 – (a) basé composantes (b) basé occlusions (c) basé composantes & occlusions

Les graphes hybrides construits précédemment peuvent ensuite être traités par notre méthode de filtrage sous contraintes de distance (pseudo-algorithme (1.3.8)). En effet, les arcs de voisinage y sont attribués par une distance euclidienne (d), ils peuvent donc être filtrés. La figure (1.29) (b) donne un exemple de graphe sous contraintes de distance (b) du graphe hybride (a). Les relations de voisinage entre occlusions distantes y ont été filtrées permettant ainsi la segmentation des graphes de voisinage correspondant aux symboles.

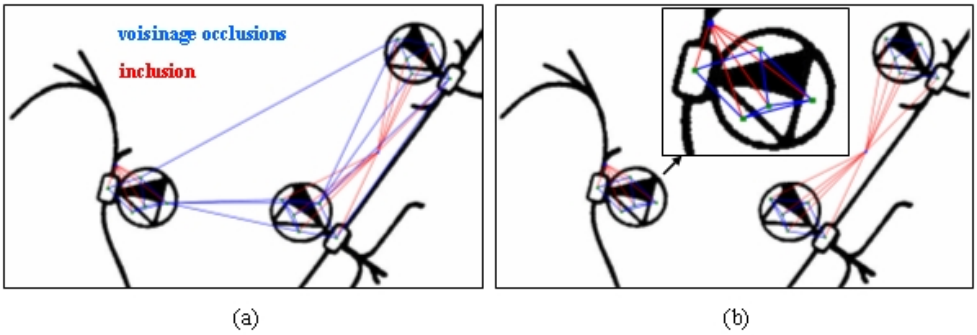


FIG. 1.29 – (a) graphe hybride (b) graphe hybride sous contraintes de distance

Dans cette sous-section, nous avons présenté notre méthode pour la construction de graphes hybrides. Celle-ci fusionne les graphes de voisinage et d'inclusion à partir de la comparaison des caractéristiques topologiques de leurs composantes et occlusions. Trois types de graphes hybrides peuvent être alors construits, basés sur : les composantes, les occlusions et les composantes & occlusions. Ces graphes hybrides peuvent ensuite être exploités par notre méthode de filtrage sous contraintes de distance. Ces contraintes s'appliquent alors aux relations de voisinage composant les graphes hybrides. Ainsi, via la combinaison de nos méthodes, il est possible de construire différents types de graphes. Ceci permet d'adapter la représentation employée en fonction du problème de reconnaissance envisagé.

1.3.7 Conclusion

Dans cette section nous avons présenté nos extracteurs de primitives graphiques par approche région. Ceux-ci utilisent tout d'abord des méthodes de marquage de composantes connexes et d'extraction d'occlusions. Basés sur les composantes et occlusions ils permettent la construction de différents types de graphes : d'inclusion, de voisinage et sous contraintes de distance. Nous combinons alors ces graphes via une méthode de construction de graphes hybrides. Ainsi, grâce à la combinaison de nos méthodes il est possible de construire différentes représentations à base de graphe de régions. Ceci permet d'adapter ces représentations au problème de reconnaissance envisagé. Dans la section suivante, nous présentons notre dernière approche pour l'extraction de primitives graphiques basée sur la squelettisation des formes.

1.4 Extraction par approche squelette

1.4.1 Introduction

Au cours des deux sections précédentes, nous avons présenté nos opérateurs d'extraction de primitives graphiques par approches contour et région. Nos contributions dans ce chapitre visent à préparer la mise en oeuvre de notre approche de reconstruction d'objets présentée dans le chapitre suivant. Dans cette optique, nous cherchons à réaliser des opérateurs permettant différents niveaux d'extraction. Nous présentons dans cette section des opérateurs basés sur une approche squelette. Ceux-ci sont ainsi complémentaires aux opérateurs par approche contour et région présentés précédemment. De cette façon, l'ensemble de nos opérateurs permettront à terme les différents niveaux de représentation région, contour et squelette.

Au cours du chapitre (**??**) nous avons présenté une étude bibliographique des différentes méthodes[27] d'extraction de primitives graphiques à base de squelettisation. Nous avons illustré leur décomposition en deux étapes : une de squelettisation et une de construction du graphe de squelette. Dans le cadre de nos contributions présentées dans cette section, nous nous positionnons sur l'étape de construction du graphe de squelette. En effet, nous employons en ce qui concerne l'étape squelettisation un algorithme[28] de la littérature : celui de [Baja 94]. Ce dernier est à base de transformée de distance, il est donc de complexité réduite et permet d'extraire une information d'épaisseur sur les formes. Il utilise également une transformée ($d_{3,4}$) qui permet une approximation de la distance euclidienne. De cette façon, il est adapté aux formes multi-orientées largement présentes dans les documents graphiques. Différents systèmes d'analyse des documents graphiques y ont d'ailleurs recours [Vossepoell 97] [Tombre 98] [Fränti 99]. La figure (1.30) donne un exemple de résultat de notre squelettiseur basé sur cet algorithme de [Baja 94].

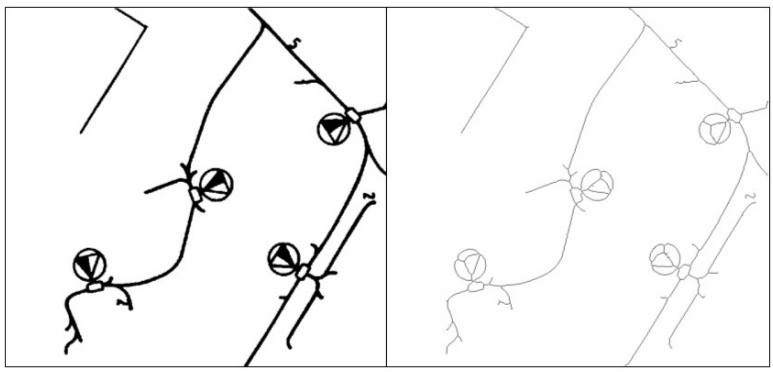


FIG. 1.30 – Résultat de notre squelettiseur basé sur l'algorithme de [Baja 94]

[27] Nous reportons le lecteur page **??** sur ces méthodes.
[28] Cet algorithme est exploité de la librairie PSI présentée en Annexe C.

Les images squelettisées sont ensuite traitées par des méthodes de construction de graphe de squelette. Au cours du chapitre (**??**) nous avons présenté une étude bibliographique sur ces méthodes[29]. Celles-ci extraient des squelettes différentes primitives graphiques : extrémités, chaînes et jonctions. Nous avons illustré que la problématique principale était le traitement des jonctions multiples (jonctions composées de multiples pixels). Ces méthodes procèdent alors par deux approches pour leur traitement (figure (1.31)) : détection (a) ou suppression/reconstruction (b). La première (a) extrait directement des graphes de squelette par analyse locale de chacun des pixels formes de l'image. La seconde (b) procède en deux étapes : une étape de suppression des pixels aux abords des jonctions suivie d'une étape de reconstruction des jonctions. La seconde étape calcule pour cela les distances euclidiennes entre les extrémités des chaînes. Elle utilise ainsi des informations globales aux jonctions pour leur reconstruction.

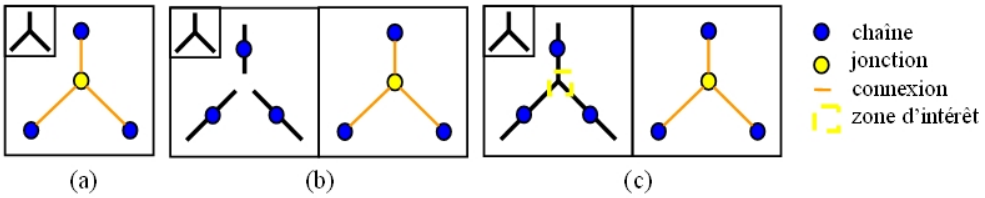


FIG. 1.31 – (a) détection (b) suppression/reconstruction (c) détection/reconstruction

Chacune de ces deux approches présentent des avantages et inconvénients. Celle par détection (a) respecte la connexité du squelette mais pose des problèmes de traitement des jonctions multiples. Ceci est dû au manque de prise en compte d'informations globales sur les jonctions durant le traitement des images de squelette. L'approche par suppression/reconstruction (b), en raison de son traitement en deux étapes, permet un meilleur traitement des jonctions multiples. En contre partie, elle peut entraîner des pertes de connexité des graphes construits vis à vis des images de squelette traitées.

Nous proposons ici une nouvelle approche (c) procédant par détection puis par reconstruction. Celle-ci procède donc en deux étapes au même titre que l'approche par suppression/reconstruction. Elle tire ainsi parti durant la seconde étape d'informations globales aux jonctions extraites durant la première. Cependant contrairement à l'approche par suppression/reconstruction, notre approche procède par détection durant la première étape. De cette façon, elle n'est pas sujette aux problèmes de ruptures de connexité. Pour ce faire, elle est basée sur un principe de catégorisation des pixels formes. Cette catégorisation permet d'extraire des zones d'intérêt sur les jonctions durant l'étape de détection. Ces zones sont alors utilisées durant la seconde étape de reconstruction des jonctions. Dans la suite de cette section, nous présentons tout d'abord dans la sous-section (1.4.2) notre principe de catégorisation des pixels. Nous présentons ensuite dans les sous-sections (1.4.3) et (1.4.4) nos étapes de détection puis de reconstruction des jonctions.

[29] Nous reportons le lecteur page **??** sur ces méthodes.

1.4.2 Catégorisation des pixels

La base de notre approche repose sur un principe de catégorisation des pixels formes du squelette. Nous employons pour cela six catégories de pixels : isolés, extrémités, listes, redondants, sur-connectés et jonctions. La figure (1.32) en donne des exemples.

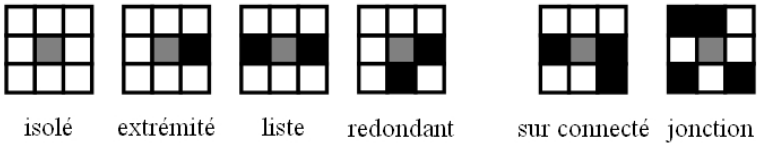


FIG. 1.32 – Exemples de pixels des différentes catégories

Les catégories de pixels isolés, extrémités, listes, et jonctions sont classiquement utilisées dans la littérature [Liu 99], [Tombre 00], [Lin 02], ... Les pixels isolés correspondent aux résultats de squelettisation de composantes de faibles tailles. Ils sont assimilables à du bruit et sont généralement effacés. Les pixels extrémités, listes, et jonctions représentent quant à eux les éléments structurants d'un squelette. Dans notre approche nous avons introduit deux catégories supplémentaires de pixels : redondants et sur-connectés. Les pixels redondants sont caractéristiques des sur-épaisseurs d'un squelette. Nous utilisons les pixels sur-connectés afin de détecter en amont les pixels jonctions et redondants. La figure (1.33) donne des exemples de pixels sur-connectés aux abords d'un pixel redondant (a) et d'un pixel jonction (b).

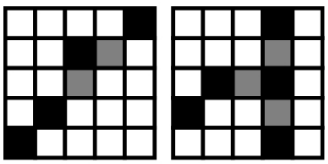


FIG. 1.33 – pixels sur-connectés[30](a) à un pixel redondant (b) à un pixel jonction

Nous avons basé notre principe de catégorisation des pixels sur l'analyse de leur voisinage. En effet une catégorisation basée sur cette analyse est exhaustive : si on considère le 8 voisinage d'un pixel (p) le nombre de configurations (C_i) maximum de ce voisinage est de 256. L'équation (1.3) suivante détaille ce calcul [Cowell 01].

$$C_i = \begin{array}{|c|c|c|}\hline p_3 & p_2 & p_1 \\ \hline p_4 & p & p_{0,8} \\ \hline p_5 & p_6 & p_7 \\ \hline \end{array} \qquad i_m = \sum_{k=0}^{8} C_8^k = 256 \tag{1.3}$$

[30] Les pixels sur-connectés sont représentés en grisés.

Le but de notre approche est de répartir ces 256 configurations (C_i) selon nos six catégories (isolé, extrémité, liste, redondant, sur-connecté et jonction). Pour cela nous avons attribué chaque (C_i) selon trois caractéristiques liées à son pixel central (p) : le nombre de voisins connexes (N_n), le nombre croisements (N_c) et le nombre de croisements connexes (N_{cc}). Le calcul du nombre de voisins connexes (N_n) détermine la connexité de (p). Le calcul du nombre de croisements détermine le nombre de séquences de pixels voisins (p_i) de (p) qui ne sont pas 4-connexes entre eux. Le calcul de (N_{cc}) permet alors de compléter le calcul de (N_c) en déterminant le nombre de pixels voisins (p_i) de (p) 8-connexe et non 4-connexe. La figure (1.34) donne des exemples de calculs de (N_n), (N_c) et (N_{cc}).

$$P_n = \begin{array}{|c|c|c|}\hline p_3 & p_2 & p_1 \\ \hline p_4 & p & p_{0,8} \\ \hline p_5 & p_6 & p_7 \\ \hline \end{array} \qquad N_n = \sum_{i=0}^{7} p_i \tag{1.4}$$

$$N_c = \frac{1}{2} \times \sum_{i=0}^{7} |p_{i+1} - p_i| \qquad N_{cc} = \sum_{i=0,2,4,6} (p_i \times p_{i+2}) \times (1 - p_{i+1}) \tag{1.5}$$

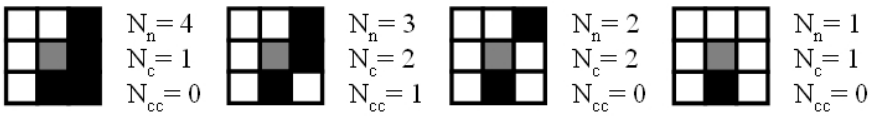


FIG. 1.34 – Exemples de calculs de N_n, N_c et N_{cc}

À partir des définitions de N_n,N_c, et N_{cc} nous avons réparti les 256 configurations (C_i) selon nos six catégories. Cette répartition s'est effectuée selon deux critères. Premièrement nous avons établi les définitions (1) et (2) de nos six catégories. Ensuite nous avons réparti les 256 (C_i) en différents ensembles (E_i). Ces ensembles (E_i) ont été à leur tour répartis selon nos six catégories. Nous détaillons cette répartition par la suite.

Définition 1 ***Un pixel est dit :***

isolé si il n'a aucun voisin.

extrémité si il a un unique voisin.

liste si il possède deux voisins non connexes.

redondant si il possède deux à trois voisins connexes.

Définition 2 ***Un pixel est dit :***

sur-connecté si ses voisins forment deux ensembles non connexes.

jonction si $N_c \geq 3$, ou si $N_n \geq 4$ et qu'il n'est pas sur-connecté.

Les valeurs $\{N_n,N_c,N_{cc}\}$ sont dépendantes les unes des autres. Ces dépendances sont illustrées dans le tableau (1.1), elles sont à deux niveaux. En premier lieu les valeurs de (N_c) sont bornées par celles de (N_n). En second lieu les plages de valeurs de (N_{cc}) sont spécifiques à chacun des couples de valeurs $\{N_n,N_c\}$.

N_n	N_c	N_{cc}	N_n	N_c	N_{cc}
0	0	0	5	{1,2,3}	{ 0, {0,1}, {0,2} }
1	1	0	6	{1,2}	{ 0, {0,2} }
2	{1,2}	{ 0, {0,1} }	7	1	{ 0, 1 }
3	{1,2,3}	{ 0, {0,1}, {0,1,2} }	8	1	0
4	{1,2,3,4}	{ 0, {0,1}, {0,1,2}, {0,4} }			

TAB. 1.1 – Dépendances N_n, N_c, et N_{cc}

Ces dépendances à deux niveaux nous ont orienté à formaliser la répartition des configurations (C_i) selon une arborescence[31]. Les figures (1.35) et (1.37) présentent l'arborescence que nous utilisons. Celle-ci représente trois niveaux de noeuds attribués. Ces noeuds y représentent alors les différentes valeurs de (N_n), (N_c) et (N_{cc}). Tout les noeuds appartenant aux niveaux (N_n) et (N_c) sont internes. Ceux appartenant au niveau (N_{cc}) sont des feuilles. Chacune de ces feuilles correspond alors à un ensemble (E_i) de configurations (C_i). Pour chaque (E_i) le triplet $\{N_n,N_c,N_{cc}\}$ est alors identique pour les (C_i) le composant. À travers cette arborescence nous avons donc réparti nos 256 configurations (C_i) en 30 ensembles (E_i). Ces 30 ensembles (E_i) sont à leur tour répartis en six catégories : isolé, extrémité, liste, redondant, sur-connecté et jonction.

L'attribution de notre arborescence par les valeurs entières de (N_n), (N_c) et (N_{cc}) est d'ordre lexicographique. En effet, pour chacun des niveaux les entiers attribuant les noeuds sont croissants et bornés. De cette façon, les triplets d'attributs ordonnés $\{N_n,N_c,N_{cc}\}$ adressent de façon unique les feuilles de notre arborescence, et donc les différents (E_i). Les figures (1.36) et (1.38) présentent pour chacun des triplets $\{N_n,N_c,N_{cc}\}$ et donc (E_i) des exemples de configurations (C_i).

Basé sur cet arborescence nous présentons dans les pseudo-algorithmes (1.4.1) et (1.4.2) nos méthodes employées pour la catégorisation des pixels. Tout d'abord nous utilisons une méthode (1.4.1) pour le calcul de (N_n), (N_c) et (N_{cc}). Celle-ci permet un calcul quasi-simultané de (N_n), (N_c) et (N_{cc}). En effet, seuls 12 accès au voisinage (P_n) de (p) sont nécessaires pour ce calcul. Ensuite, nous utilisons le triplet $\{N_n,N_c,N_{cc}\}$ calculé afin de catégoriser (p). Notre méthode de catégorisation est présentée dans le pseudo-algorithme (1.4.2). Celle-ci parcourt dans un ordre lexicographique notre arborescence afin d'en extraire les positions des noeuds ordonnés d'attributs $\{N_n,N_c,N_{cc}\}$. Ces positions permettent alors d'identifier la catégorie du pixel à partir d'un tableau à trois dimensions de structure identique à l'arborescence.

[31] Nous reportons le lecteur à l'Annexe A pour une introduction sur les graphes et arborescences.

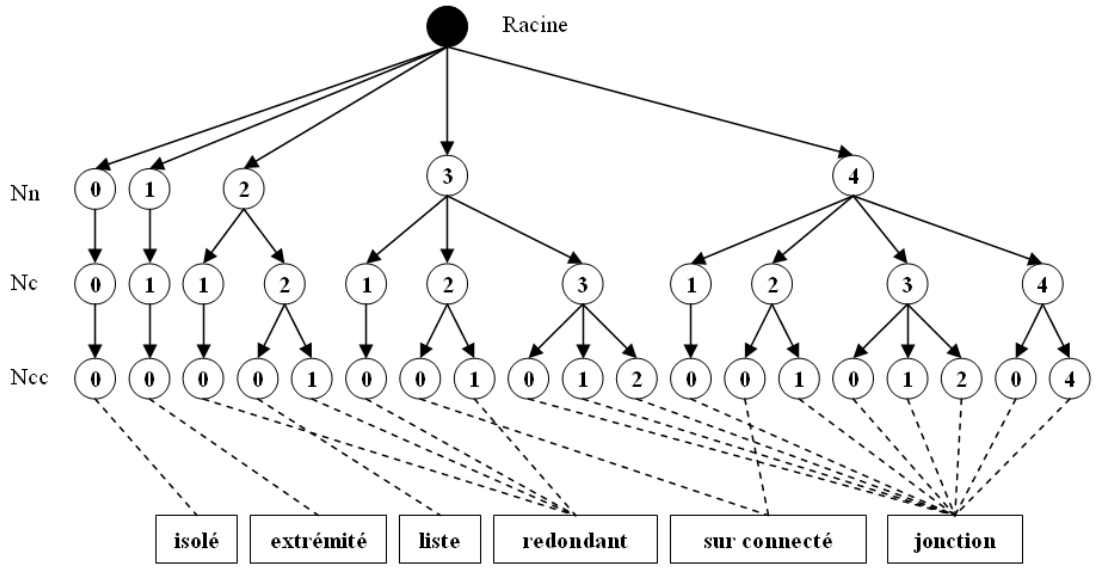


FIG. 1.35 – Arborescence de catégorisation des pixels formes du squelette (1/2)

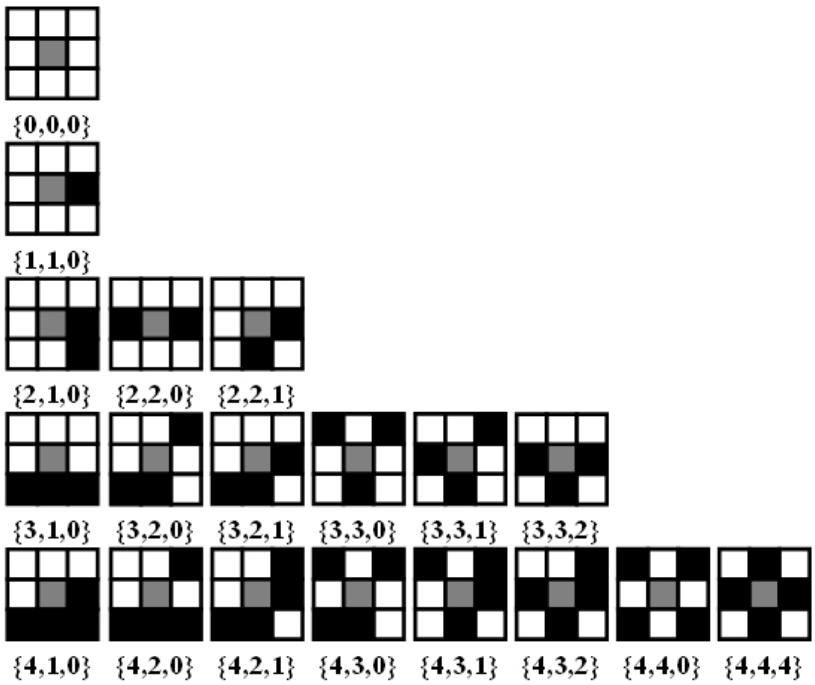


FIG. 1.36 – Exemples de configurations (C_i) pour chacun des triplets $\{N_n,N_c,N_{cc}\}$ (1/2)

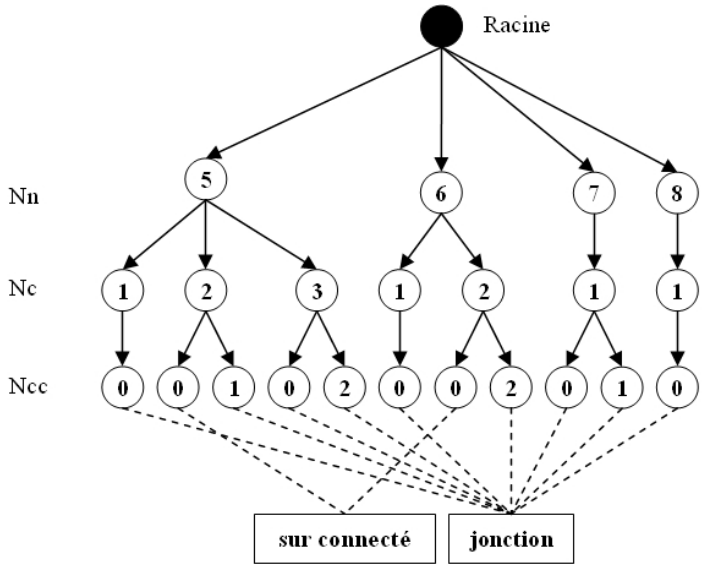


FIG. 1.37 – Arborescence de catégorisation des pixels formes du squelette (2/2)

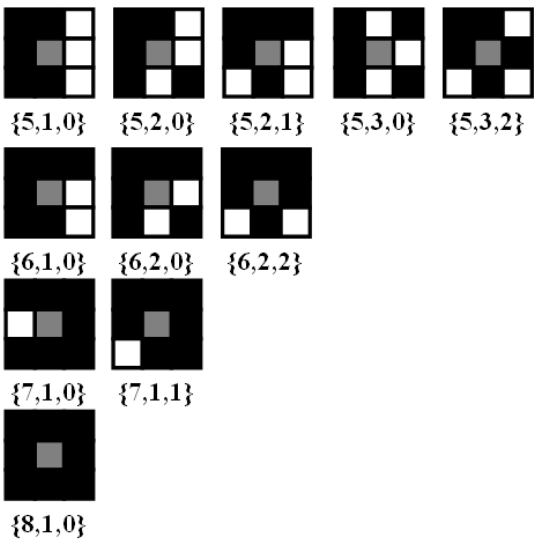


FIG. 1.38 – Exemples de configurations (C_i) pour chacun des triplets $\{N_n,N_c,N_{cc}\}$ (2/2)

Pseudo-algorithme 1.4.1: CONNEXITÉ($p = \{x, y\}$)

globale $squelette, p_e, chaîne, jonctions$ **locale** N_n, N_c, N_{cc}

fonction VALEUR(p)
si p est de $forme$ ou marqué dans $squelette$ **retourner** 1 **sinon retourner** 0

fonction CALCUL($\{x_1, y_1\}, \{x_2, y_2\}, \{x_3, y_3\}$)
$v_1 \leftarrow$ VALEUR(x_1, y_1)
$v_2 \leftarrow$ LIRE(x_2, y_2)
$v_2 \leftarrow$ LIRE(x_3, y_3)
$N_n \leftarrow N_n + v_1 + v_2$
$N_c \leftarrow N_c + (v_2 - v_1) + (v_3 - v_2)$
$N_{cc} \leftarrow N_{cc} + (v_1 \times v_3) \times (1 - v_2)$

principal
$N_n \leftarrow N_c \leftarrow N_{cc} \leftarrow 0$
CALCUL$(\{x+1, y\}, \{x+1, y-1\}, \{x, y-1\})$
CALCUL$(\{x, y-1\}, \{x-1, y-1\}, \{x-1, y\})$
CALCUL$(\{x-1, y\}, \{x-1, y+1\}, \{x, y+1\})$
CALCUL$(\{x, y+1\}, \{x+1, y+1\}, \{x+1, y\})$
$N_c \leftarrow \frac{N_c}{2}$
retourner (N_n, N_c, N_{cc})

Pseudo-algorithme 1.4.2: CATÉGORISATION(N_n, N_c, N_{cc})

globale $arborescence, types$
principal
$noeud \leftarrow$ racine de $arborescence$
$i \leftarrow 1$
tant que $attribut$ du fils i de $noeud \neq N_n$
  faire $i \leftarrow i + 1$
$noeud \leftarrow$ fils i de noeud
$j \leftarrow 1$
tant que $attribut$ du fils j de $noeud \neq N_c$
  faire $j \leftarrow j + 1$
$noeud \leftarrow$ fils j de noeud
$k \leftarrow 1$
tant que $attribut$ du fils k de $noeud \neq N_{cc}$
  faire $k \leftarrow k + 1$
retourner $(types[i][j][k])$

Dans cette sous-section nous avons présenté notre approche de catégorisation des pixels. Cette dernière est basée sur le calcul des trois caractéristiques $\{N_n,N_c,N_{cc}\}$. Celles-ci sont alors utilisées pour répartir les 256 configurations (C_i) possibles du voisinage d'un pixel en 30 ensembles (E_i). Ces 30 ensembles (E_i) sont à leur tour répartis selon 6 catégories de pixel : isolé, extrémité, liste, redondant, sur-connecté et jonction. Ces répartitions sont formalisées sous la forme d'une arborescence dans laquelle chaque triplet de caractéristiques ordonnées $\{N_n,N_c,N_{cc}\}$ adresse de façon unique chacune des configurations (E_i). Cette approche par catégorisation des pixels est ainsi exhaustive : aucune des configurations (C_i) rencontrées sur l'image du squelette ne peut être inconnue. Dans la sous-section (1.4.3) suivante nous présentons comment nous utilisons cette approche par catégorisation des pixels dans notre première étape de détection pour la construction du graphe de squelette.

1.4.3 Détection

Dans cette sous-section nous présentons notre première des deux étapes pour la construction du graphe de squelette : celle de détection. Celle-ci est basée sur notre approche par catégorisation des pixels présentée dans la sous-section précédente. Elle a pour but d'extraire les différentes chaînes des squelettes mais aussi des zones d'intérêt en ce qui concerne les jonctions. Ces zones d'intérêt correspondent alors aux différents pixels de catégorie jonction détectés sur l'image. Les chaînes et zones d'intérêt sont utilisées ultérieurement durant notre seconde étape de reconstruction des jonctions présentée dans la sous-section (1.4.4) suivante.

Notre méthode (*détection*) est présentée dans le pseudo-algorithme (1.4.3). Elle repose sur deux méthodes : (*entrée*) et (*chaîner*). Elle parcourt l'image (*squelette*) via la méthode (*entrée*) à la recherche de pixel d'entrée (p_e). Ces derniers peuvent être de catégories extrémité ou liste. La rencontre d'un pixel d'entrée (p_e) déclenche la méthode (*chaîner*) à partir de ce pixel. Selon que (p_e) soit un pixel extrémité ou liste cette méthode est déclenchée dans un seul ou en double sens. La méthode (*détection*) s'exécute tant que des pixels d'entrée (p_e) subsistent dans l'image (*squelette*).

La méthode (*entrée*) est détaillée dans le pseudo-algorithme (1.4.4). Celle-ci parcourt l'image (*squelette*) à la recherche des pixels formes. Le parcours est initialisé à partir du dernier point (p_e) rencontré. Durant ce parcours chaque pixel forme (p) rencontré est catégorisé. Cette catégorisation s'effectue par appel de la méthode (*catégoriser*) présentée dans le pseudo-algorithme (1.4.2) précédent. Notre méthode exploite alors les résultats de cette catégorisation afin de traiter les pixels extrémités, listes, isolés et jonctions. La rencontre d'un pixel extrémité ou liste initialise le point d'entrée (p_e) et retourne la catégorie détecté. Les pixels isolés eux sont effacés de l'image (*squelette*). Les pixels jonctions sont marqués sur l'image (*squelette*) et ajoutés dans une liste globale (*jonctions*). Cette liste globale (*jonctions*) correspond alors aux zones d'intérêt détectées par notre méthode. Elle est ensuite utilisée durant notre seconde étape de reconstruction des jonctions présentée dans la sous-section (1.4.4) suivante.

Pseudo-algorithme 1.4.3: DÉTECTION()

globale $squelette, p_e, chaîne, jonctions$
principal
initialiser les bords de $squelette$
$chaînage \leftarrow \emptyset$
$p_e \leftarrow \{1, 2\}$ $type \leftarrow$ ENTRÉE()
tant que $p_e \neq \emptyset$
faire
- **si** $type = extrémité$
  - **alors**
    - $chaîne \leftarrow \emptyset$
    - CHAÎNER($type, fin$)
    - ajouter $chaîne$ à $chaînage$
    - $type \leftarrow$ ENTRÉE()
  - **sinon si** $type = liste$
    - **alors**
      - $chaîne \leftarrow \emptyset$
      - CHAÎNER($type, fin$)
      - CHAÎNER($type, début$)
      - ajouter $chaîne$ à $chaînage$
      - $type \leftarrow$ ENTRÉE()

retourner ($chaînage$)

Pseudo-algorithme 1.4.4: ENTRÉE()

globale $squelette, p_e, chaîne, jonctions$
principal
$\{x_d, y_d\} \leftarrow$ LIRE(p_e) $p_e \leftarrow \emptyset$
pour y variant de y_d à $hauteur - 1$ de $squelette$
faire
- **pour** x variant de $x_d + 1$ à $largeur - 1$ de $squelette$
  - **faire si** p=$\{x$,$y\}$ est un pixel $forme$ dans $squelette$
    - **alors**
      - $\{N_n, N_c, N_{cc}\} \leftarrow$ CONNEXITÉ($squelette, p$)
      - $type \leftarrow$ CATÉGORISATION(N_n, N_c, N_{cc})
      - **si** type = $extrémité$
        - **alors**
          - $p_e \leftarrow p$
          - **retourner** ($extrémité$)
      - **sinon si** type = $liste$
        - **alors**
          - $p_e \leftarrow p$
          - **retourner** ($liste$)
      - **sinon si** $type$=$isolé$
        - **alors** effacer p sur $squelette$
      - **sinon si** type = $jonction$
        - **alors**
          - ajouter p à $jonctions$
          - marquer p sur $squelette$ comme $jonction$
- $x_d \leftarrow 1$

retourner ($\emptyset$)

Parallèlement à la méthode (*entrée*) nous utilisons la méthode (*chaîner*). Celle-ci est présentée dans le pseudo-algorithme (1.4.5). Elle a pour objectif de traiter individuellement chacune des chaînes de l'image (*squelette*). Elle chaîne et marque pour cela itérativement les pixels extrémités, listes, redondants, et sur-connectés d'une chaîne donnée. La catégorisation de ces pixels s'effectue à l'aide des méthodes (*connexité*) et (*catégorisation*) présentées dans les pseudo-algorithmes précédents (1.4.1) et (1.4.2). Les différents pixels sont ensuite ajoutés à une variable (*chaîne*) et marqués sur l'image (*squelette*). Le premier pixel chaîné et marqué est celui d'entrée (p_e) initialisé par la méthode (*entrée*). Les pixels suivants (p_s) sont ensuite trouvés par analyse du 8-voisinage du dernier pixel chaîné (p_c). Cette analyse recherche alors les pixels de couleur de la ($forme$) et non marqués dans l'image (*squelette*). Les pixels sont chaînés en fonction d'une variable (*sens*) paramètre de la méthode et définie depuis la méthode (*détection*). Selon que le pixel d'entrée soit extrémité ou liste le chaînage peut s'effectuer dans un seul ou en double sens. Les pixels chaînés sont alors respectivement ajoutés en fin et en début de la liste (*chaîne*) pour chaque sens du chaînage.

Le marquage et le chaînage d'un pixel (p_s) est fonction de sa catégorisation. En fonction du résultat de cette catégorisation le chaînage se déroule de différentes façons. Un pixel liste est normalement chaîné et marqué. La rencontre d'un pixel extrémité provoque son chaînage et marquage mais stoppe la méthode (*chaîner*). La rencontre d'un pixel jonction provoque l'arrêt de la méthode (*chaîner*) sans chaînage ni marquage de celui-ci. En effet les pixels jonctions sont traités par notre méthode (*entrée*). La rencontre d'un pixel sur-connecté provoque des traitements particuliers qui se déclinent en deux cas de figure principaux. En effet un pixel sur-connecté est annonciateur que soit des pixels redondants et/ou soit des pixels jonctions sont présents dans son voisinage. Premièrement la présence d'un pixel jonction dans le voisinage provoque le marquage du pixel sur-connecté et l'arrêt de la procédure de chaînage. L'absence de pixels jonctions dans le voisinage implique systématiquement que le voisinage du pixel sur-connecté est composé de deux pixels redondants et/ou sur-connectés. Deux configurations sont alors possibles, celles-ci sont présentées sur la figure (1.39). Notre méthode (*chaîner*) adopte alors un traitement adéquate pour chacune de ces configurations. Dans un cas $\{r \ \& \ s\}$ le pixel redondant (r) est effacé et celui sur-connecté (s) chaîné. Dans un cas $\{r \ \& \ r\}$ l'un des deux pixels redondants (r) est alors ignoré.

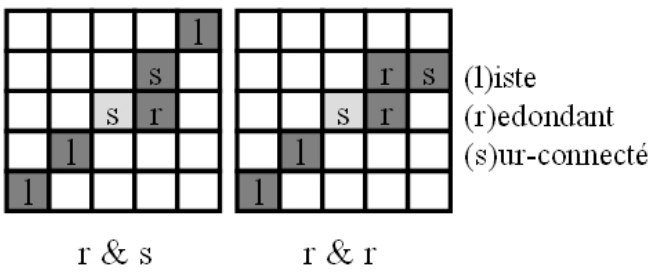


FIG. 1.39 – Cas de pixels sur-connectés 2-connexe dans une chaîne

Pseudo-algorithme 1.4.5: CHAÎNER($type, sens$)

globale $squelette, p_e, chaîne, jonctions$

fonction TRAITER($p, type, sens$)
marquer p sur $squelette$ comme $type$
ajouter p en $sens$ de $chaîne$

principal
$p_c \leftarrow p_e$
si $sens = fin$
  alors TRAITER($p_c, type, sens$)
$continue \leftarrow$ **vrai**
tant que $continue$
faire {
  $p_s \leftarrow$ voisins suivant de p_c de $forme$ dans $squelette$
  si $p_s \neq \emptyset$
  alors {
    $\{N_n, N_c, N_{cc}\} \leftarrow$ CONNEXITÉ(p_s)
    $type \leftarrow$ CATÉGORISATION(N_n, N_c, N_{cc})
    si $type = liste$
      alors { $p_c \leftarrow p_s$; TRAITER($p_c, liste, sens$) }
    sinon si $type = extrémité$
      alors { $p_c \leftarrow p_s$; TRAITER($p_c, extrémité, sens$) ; $continue \leftarrow$ **faux** }
    sinon si $type = jonction$
      alors $continue \leftarrow$ **faux**
    sinon si $type = sur\text{-}connecté$
      alors {
        $p_c \leftarrow p_s$
        TRAITER($p_c, liste, sens$)
        si p_c a un voisin non marqué $jonction$
          alors { ajouter p_c à jonctions ; $continue \leftarrow$ **faux** }
        sinon si les voisins non marqués de p_c forme $\{s,r\}$
          alors { effacer r sur $squelette$; TRAITER($s, liste, sens$) ; $p_c \leftarrow s$ }
        sinon si les voisins non marqué de p_c forme $\{r_1,r_2\}$
          alors { effacer r_1 sur $squelette$; TRAITER($r_2, liste, sens$) ; $p_c \leftarrow r_2$ }
      }
  sinon $continue \leftarrow$ **faux**
}

Notre méthode de détection permet ainsi l'extraction des chaînes du squelette en seule passe sur l'image. Basée sur notre principe de catégorisation des pixels elle permet durant cette extraction le filtrage des pixels peu informatifs comme ceux de catégories isolé et redondant. Elle permet également la détection de zones d'intérêt en ce qui concerne les jonctions. Cette détection ne s'effectue pas ici par analyse des extrémités des chaînes (comme c'est généralement le cas dans les méthodes de la littérature)[32] mais durant le parcours global de l'image. Ceci rend notre méthode particulièrement adaptée au traitement des jonctions multiples. La figure (1.40) donne des exemples de résultats de détection de pixels jonctions par notre méthode. Cependant les pixels jonctions détectés ne sont pas structurés : leurs relations de connexion avec les chaînes et les autres pixels jonctions ne sont pas construites. Nous présentons dans la sous-section suivante (1.4.4) la méthode que nous employons pour reconstruire ces jonctions.

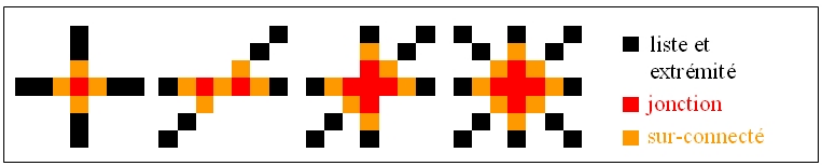


FIG. 1.40 – Détection de pixels jonctions

1.4.4 Reconstruction des jonctions

Notre méthode de reconstruction des jonctions exploite les différentes chaînes extraites au cours de l'étape de détection ainsi que les zones d'intérêt en ce qui concerne les jonctions. Elle se déroule en deux étapes : une de fusion des jonctions et une structuration du graphe de squelette.

La méthode que nous utilisons pour l'étape de fusion des jonctions est présentée dans le pseudo-algorithme (1.4.6). Celle-ci exploite la liste (*jonctions*) construite durant l'étape de détection. Elle parcourt cette liste de façon à fusionner des ensembles de pixels jonctions en jonctions multiples (jm). Elle utilise pour cela une méthode (*appartient*) qui teste pour un pixel donné (p) si celui-ci est 8-connexe à l'un des pixels (q) d'une jonction multiple (jm). Les différents pixels jonctions sont alors regroupés au sein de différents ensembles (jm) correspondant aux jonctions multiples. Selon la nature des jonctions présentes sur l'image ces jonctions multiples peuvent correspondre à un seul pixel jonction ou à un ensemble de pixels jonctions. Notre liste (*jonctions*) contient les pixels jonctions détectés par la méthode (*entrée*) mais aussi les pixels sur-connectés à ces jonctions détectés durant le chaînage (pseudo-algorithme (1.4.5)). De cette façon ces pixels sur-connectés sont pris en compte durant la reconstruction des jonctions. Cette prise en compte permet la fusion des jonctions multiples proches comme l'illustre la figure (1.41).

[32] Nous reportons le lecteur page **??** sur ces aspects.

Pseudo-algorithme 1.4.6: FUSION(*jonctions*)

fonction APPARTIENT($jm, p = \{x_j, y_j\}$)
 pour tout $q = \{x, y\}$ de jm
  faire { **si** $|x - x_j| \leq 1$ **et** $|y - y_j| \leq 1$
   alors retourner (**vrai**)
 retourner (**faux**)

principal
 $multiple \leftarrow \emptyset$
 pour $i = 1$ **à** TAILLE(*jonctions*)
  faire {
   $jm \leftarrow \emptyset$
   ajouter $jonctions[i]$ à jm
   supprimer $jonctions[i]$
   décrémenter i
   pour $j = i + 1$ **à** TAILLE(*jonctions*)
    faire si APPARTIENT($jm, jonctions[j]$)
     alors {
      ajouter $jonctions[j]$ à jm
      supprimer $jonctions[j]$
      décrémenter j
   ajouter jm à $multiple$
 retourner (*multiple*)

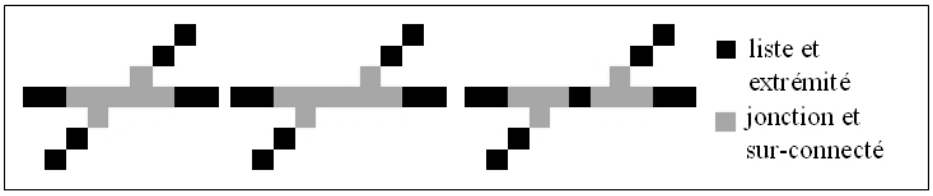


FIG. 1.41 – Reconstruction de jonctions

Dans une deuxième étape nous exploitons les jonctions multiples fusionnées et les chaînes extraites afin de structurer le graphe de squelette. Notre méthode est présentée dans le pseudo-algorithme (1.4.7). Elle initialise tout d'abord les noeuds d'un graphe de squelette (g_s) à partir de chacune des chaînes (c) et des jonctions multiples (jm). Elle parcourt ensuite chacune des chaînes (c) attribuant les premiers noeuds de (g_s) afin d'en extraire les extrémités (p_d) et (p_f). Les relations d'appartenance entre ces extrémités (p_d) et (p_f) et les jonctions multiples (jm) sont alors recherchées via la méthode (*appartient*) présentée précédemment dans le pseudo-algorithme (1.4.6). Dans un cas d'appartenance un arc (a) est construit entre chaque couple de noeuds {n_1,n_2} attribués respectivement par (c) et (jm). La figure (1.42) donne un exemple de résultat de construction de graphe de squelette (d) de l'image (a) avec les étapes de squelettisation (b) et de chaînage (c).

Pseudo-algorithme 1.4.7: STRUCTURATION(*chaînage*, *multiple*)

$g_s \leftarrow \emptyset$

pour tout chaîne c_i de chaînage
  faire ajouter un noeud attribué c_i à g_s

pour tout jonction multiple jm_i de multiple
  faire ajouter un noeud attribué jm_i à g_s

$t_c \leftarrow$ taille de *chaînage*
$t_m \leftarrow$ taille de *multiple*
pour $i = 1$ **à** t_c
  faire
    $n_1 \leftarrow$ noeud i de g_s
    $c \leftarrow$ attribut de n_1
    $p_d \leftarrow$ début de c
    $p_f \leftarrow$ fin de c
    pour $j = t_c + 1$ **à** $t_c + t_m$
      faire
        $n_2 \leftarrow$ noeud j de g_s
        $jm \leftarrow$ attribut de n_2
        si APPARTIENT(jm_j, p_d) **ou** APPARTIENT(jm_j, p_f)
          alors ajouter un arc $a = \{n_1, n_2\}$ à g_s

retourner (g_s)

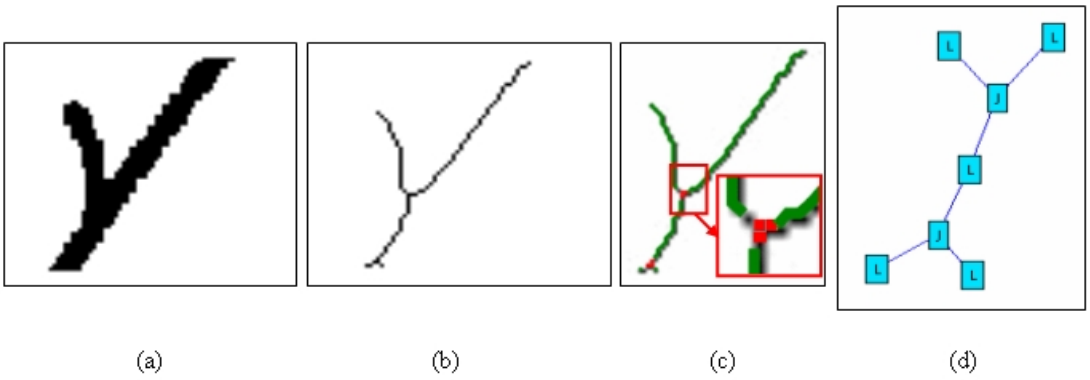


FIG. 1.42 – Construction du graphe de squelette
(a) image (b) squelette (c) détection (d) graphe

1.4.5 Conclusion

Dans cette section nous avons présenté nos opérateurs d'extraction de primitives graphiques par approche squelette. Ceux-ci sont basés deux méthodes : une de squelettisation puis une de construction du graphe de squelette. Celle de squelettisation repose sur un algorithme existant de la littérature. Concernant la seconde méthode nous avons proposé une approche originale procédant en deux étapes : une de détection puis une de reconstruction. Le coeur de cette approche repose sur un principe de catégorisation des pixels formes. Notre étape de détection exploite alors cette catégorisation afin d'extraire des squelettes des chaînes ainsi que des zones d'intérêt en ce qui concerne les jonctions. Ces zones d'intérêt sont détectées par parcours global de l'image ce qui rend notre approche particulièrement adaptée au traitement des jonctions multiples. Elles sont exploitées dans une seconde étape afin de reconstruire les jonctions du squelette. Ces dernières sont alors structurées avec les chaînes afin de construire les graphes de squelette.

1.5 Conclusion

Dans ce chapitre nous avons présenté nos différents opérateurs d'extraction de primitives graphiques. Parmi le nombre important de méthodes existantes dans la littérature nous avons dû faire des choix pour l'implémentation de ces opérateurs. Notre approche dans ce manuscrit vise à la reconstruction d'objets. Nous avons donc privilégié dans nos choix la diversité des représentations au détriment dans certains cas de la complexité et de la précision des méthodes.

Nous avons plus particulièrement développé des opérateurs basés sur les approches région, contour et squelette. Nos opérateurs par approche contour exploitent une méthode originale basée sur un paradigme marquage/suivi pour l'extraction robuste de contours. Ceux par approche région permettent l'extraction des composantes et/ou occlusions ainsi que la construction de différentes relations (de voisinage, d'inclusion et sous contraintes de distance). Le caractère original de cette approche réside dans la combinaison des relations afin de multiplier les modèles de représentation. Enfin ceux par approche squelette utilisent un principe de catégorisation des pixels. Cette catégorisation est alors particulièrement adaptée pour le traitement des jonctions multiples.

Nos opérateurs permettent ainsi les différentes représentations contour, région et squelette des objets graphiques. Ces représentations sont complémentaires en raison de leurs différences de niveaux d'extraction. De cette manière nos opérateurs permettent de multiples cas de reconstruction d'objets. Nous présentons dans le chapitre suivant notre formalisme & système pour la reconstruction d'objets basés sur ces opérateurs.

Bibliographie

[Baja 94] G.S. Di Baja. *Well-Shaped, Stable, and Reversible Skeletons from the (3,4)-Distance Transform.* Journal of Visual Communication and Image Representation, vol. 5, no. 1, pages 107–115, 1994.

[Black 81] W. Black & al. *A General Purpose Follower for Line Structured Data.* Pattern Recognition (PR), vol. 14, no. 1, pages 33–42, 1981.

[Cowell 01] J. Cowell & F. Hussain. *Thinning Arabic Characters for Feature Extraction.* In Conference on Information Visualisation (IV), pages 181–185, 2001.

[Freeman 61] H. Freeman. *On the Encoding of Arbitrary Geometric Configurations.* Transactions on Electronic and Computer, vol. 10, pages 260–268, 1961.

[Fränti 99] P. Fränti, E.I. Ageenko & A.N. Kolesnikov. *Vectorising and Feature-Based Filtering for Line-Drawing Image Compression.* Pattern Analysis and Applications (PAA), vol. 2, no. 4, pages 285–291, 1999.

[Kreher 05] D.L. Kreher & D.R. Stinson. *Pseudocode : A LATEX Style File for Displaying Algorithms.* Department of Mathematical Sciences, Michigan Technological University, Houghton, USA, 2005.

[Lin 02] F. Lin & X. Tang. *Off-Line Handwritten Chinese Character Stroke Extraction.* In International Conference on Pattern Recognition (ICPR), volume 3, pages 249–252, 2002.

[Liu 99] K. Liu, K., Y.S. Huang & C.Y. Suen. *Identification of Fork Points on the Skeletons of Handwritten Chinese Characters.* Pattern Analysis and Machine Intelligence (PAMI), vol. 21, no. 10, pages 1095–1100, 1999.

[Rosenfeld 82] A. Rosenfeld & A.C. Kak. Digital picture processing. Academic Press, 2 edition, ISBN : 0125973020, 1982.

[Rutovitz 66] D. Rutovitz. *Pattern recognition.* Journal of the Royal Statistical Society, vol. 129, pages 504–530, 1966.

[Tombre 98] K. Tombre, C. Ah Soon, P. Dosch, A. Habed & G. Masini. *Stable, Robust and Off-the-Shelf Methods for Graphics Recognition.* In International Conference on Pattern Recognition (ICPR), volume 1, pages 406–408, 1998.

[Tombre 00] K. Tombre, C. Ah Soon, P. Dosch, G. Masini & S.Tabbone. *Stable and Robust Vectorization : How to Make the Right Choices.* In Workshop on Graphics Recognition (GREC), volume 1941 of *Lecture Notes in Computer Science (LNCS)*, pages 3–18, 2000.

[Vossepoell 97] A.M. Vossepoell, K. Schutte & C.F.P. Delanghel. *Memory Efficient Skeletonization of Utility Maps.* In International Conference on Document Analysis and Recognition (ICDAR), pages 797–800, 1997.

Table des figures

Liste des tableaux

Liste des pseudo-algorithmes

Table des matières