Algorithme de calcul de l'arbre des composantes avec applications à la reconnaissance des formes en imagerie satellitaire

Anthony Baillard[1], Christophe Berger[2], Emmanuel Bertin[1],
Thierry Géraud[2], Roland Levillain[2], Nicolas Widynski[2]

[1]Institut d'Astrophysique de Paris, UMR 7095
98 bis boulevard Arago, F-75014 Paris, France

[2]Laboratoire de Recherche et de Développement de l'EPITA (LRDE)
14-16 rue Voltaire, F-94276 Le Kremlin-Bicêtre Cedex, France

baillard@iap.fr, christophe.berger@lrde.epita.fr, bertin@iap.fr,
thierry.geraud@lrde.epita.fr, roland.levillain@lrde.epita.fr, nicolas.widynski@lrde.epita.fr

Résumé – Cet article présente un nouvel algorithme de calcul de l'arbre des composantes d'une image. Vis-à-vis de l'état de l'art, cet algorithme ne fait pas un usage excessif de la mémoire et travaille efficacement sur les images dont les valeurs ont une forte quantification et sur les images à valeurs à virgule flottante. Nous décrivons également une application de cet algorithme à l'identification d'objets pertinents dans des images d'astronomie.

Abstract – In this paper a new algorithm to compute the component tree is presented. As compared to the state-of-the-art, this algorithm does not use excessive memory and is able to work efficiently on images whose values are highly quantized or even with images having floating values. We also describe how it can be applied to astronomical data to identify relevant objects.

1 Introduction

L'arbre des composantes d'une image est une représentation pratique et polyvalente de son contenu. Dans cette représentation, un nœud de l'arbre fait référence à une composante connexe donnée parmi les ensembles de niveaux de l'image. La relation de parenté entre les nœuds traduit la relation d'inclusion spatiale entre les composantes à différents niveaux. De nombreuses applications s'appuient sur l'arbre des composantes : classification, filtrage d'images, segmentation, recalage et compression (pour plus de références, voir [5] par exemple). Cependant, leur utilisation majeure reste l'implémentation de nombreux opérateurs de morphologie mathématique [4], par exemple les ouvertures et fermetures algébriques et les nivellements, pour lesquels plusieurs algorithmes ont été conçus (voir [1] pour plus de références).

Le cadre de notre travail[1] est l'identification de certains objets astronomiques dans des images du ciel, à savoir des étoiles et des galaxies, tout en prenant en compte des effets d'optique (halos, croix de diffraction) et d'autres effets (traces de satellites, impacts de rayons cosmiques, traînées de saturation des CCD). L'une des meilleures applications est le masquage automatisé de défauts d'images. Cette étape est nécessaire aux astronomes pour leur permettre de calculer des statistiques fiables. Actuellement, le masquage est essentiellement fait à la main. Cela peut prendre des heures pour une zone large, voire devenir infaisable sur la prochaine génération de campagnes d'acquisition (plusieurs téraoctets d'images par nuit). Dans ce contexte, l'utilisation d'arbres de composantes pour représenter les images est immédiate : les objets à identifier peuvent être décrits en termes d'attributs. Les étoiles sont non-résolues et apparaissent donc comme des réalisations de la fonction instrumentale ou *Point-Spread Function* (PSF), avec une pleine largeur à la mi-hauteur.

Les images d'astronomie ont deux caractéristiques principales. La première est la nature des valeurs acquises : afin d'éviter les effets de la quantification après calibration et moyennage, celles-ci sont codées avec des valeurs à virgule flottante. D'autre part, avec les instruments grand-champ actuels, les images en mosaïque sont composées de plusieurs centaines de millions de pixels. Nous avons observé que les algorithmes proposés dans la littérature pour calculer les arbres de composantes n'étaient pas suffisamment appropriés. Soit les temps d'exécution étaient rédhibitoires, soit l'occupation en mémoire était trop coûteuse. Pour résoudre ces problèmes, nous avons conçu un algorithme performant, qui offre un bon compromis entre efficacité à l'exécution et utilisation de la mémoire. Par conséquent, celui-ci est également adapté à d'autres types d'images plus classiques.

Cet article est structuré comme suit. La section 2 introduit les spécificités des données d'astronomie ; notre proposition d'algorithme est décrite dans la section 3 puis comparée en section 4 ; plusieurs applications de l'identification d'objets dans des images d'astronomie sont données en section 5 ; enfin nous concluons en section 6.

2 Le cas des images d'astronomie

Les images d'astronomie contiennent des valeurs à virgule flottante avec un intervalle de dynamique élevée. La plupart des pixels correspondent au ciel (bruité) en arrière-plan, et ont donc des valeurs flottantes très basses (pixels sombres sur la figure 1) ; les pixels clairs sont des objets : étoiles, galaxies, et quelques motifs dus aux effets d'optique (halos, croix de diffraction, etc.).

[1]Le projet EFIGI (Extraction de Formes Idéalisées de Galaxies en Imagerie) est un projet ACI « masses de données » de 3 ans, financé par le Ministère de l'Enseignement Supérieur et de la Recherche.

[2]Basé sur des observations obtenues avec MegaPrime/MegaCam, un travail effectué en collaboration avec le CFHT et le CEA/DAPNIA, au Télescope Canada-France-Hawaii (CFHT), dirigé par le National Research Council (Canada), l'Institut National des Science de l'Univers du CNRS (France) et l'Université d'Hawaii. Ce travail est basé en partie

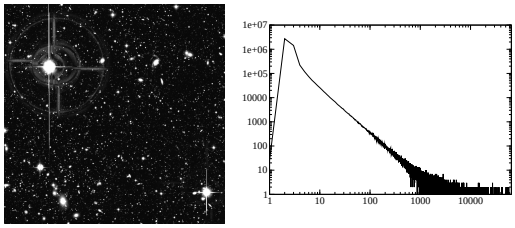

FIG. 1: À gauche : un fragment d'une image de 20 millions de pixels (5% de l'image initiale[2]). À droite : graphe log-log d'un histogramme d'image après une quantification linéaire sur 16 bits.

Pour appréhender la distribution des valeurs flottantes de telles images, nous avons quantifié d'une image codée sur 16 bits via une transformation linéaire des valeurs initiales. Un exemple d'histogramme résultant de cette transformation est reproduit sur la figure 1 (partie droite) avec un graphe $log - log$. Nous pouvons observer que la plupart des pixels sont quantifiés sur un intervalle compris entre 0 et 255 (moitié gauche du diagramme), tandis que les pixels restants sont quantifiés entre 256 et 65 535. La pente qui apparaît sur ce diagramme est due à la présence de flou. Pour être plus précis, l'image observée est le résultat d'une convolution par une fonction instrumentale (PSF), et les valeurs intermédiaires correspondent donc aux contours des objets. Par conséquent, pour éviter une perte de précision lors de l'identification des objets, nous n'avons pas d'autres choix que d'effectuer une quantification *optimale* de l'image ou mieux, de traiter directement l'image et les valeurs flottantes de ses pixels telles quelles.

Dans le cas d'une quantification optimale sur 16 bits, nous obtenons une image avec un histogramme plat et une erreur de quantification relative plutôt basse, pour chaque valeur quantifiée. Nous pouvons dès lors espérer pouvoir identifier correctement les objets, quel que soit leur intervalle de valeurs (bas, intermédiaire, haut). Malgré cela, une délimitation des contours plus précise nécessite de travailler avec les valeurs flottantes initiales de l'image.

3 L'algorithme proposé

3.1 Calcul de l'arbre de composantes

Notre proposition d'algorithme de calcul de l'arbre des composantes connexes s'appuie sur l'algorithme *union-find* [7]. Ce dernier est largement utilisé dans l'implémentation de filtres connectés [1]. Il est également à la base d'un autre algorithme récent [5] conçu pour calculer l'arbre des composantes (mais dont les besoins en quantité de mémoire utilisée sont prohibitifs dans le cas qui nous intéresse, cf. section 4).

Considérons l'échantillon d'image f de la figure 2 (en haut), auquel on adjoint une 4-connexité. Considérons ensuite la relation d'ordre total $\mathcal{R}$ entre les pixels de f basée sur les niveaux de gris décroissants et, pour deux pixels de même niveau, utilisant l'ordre de balayage vidéo usuel. $\mathcal{R}$ est représentée par une dénomination dans l'ordre lexicographique des points sur la figure 2 (en bas). Les points sont donc étiquetés de A à J suivant la relation $\mathcal{R}$.

Les ensembles de niveaux définis par $L_\lambda(f) = \{\ p \mid f(p) \geq \lambda\ \}$ pour des valeurs de λ décroissantes figurent dans la partie droite de la figure 2. Les éléments de $L_\lambda(f)$ sont affichés en caractères italiques gris s'ils font également partie de $L_{\lambda+1}(f)$. Les ensembles

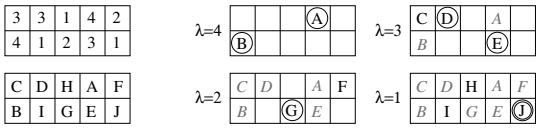

FIG. 2: À gauche : échantillon d'image (en haut) et relation d'ordre $\mathcal{R}$ entre les pixels (en bas). À droite : ensembles de niveaux pour différentes valeurs de λ (les points canoniques sont cerclés).

de niveaux de l'image peuvent être représentés par l'arbre de composantes présenté dans la figure 3 (à gauche) où chaque relation de parenté entre nœuds traduit la relation d'inclusion des composantes. Une représentation plus compacte de l'arbre de composantes est le *max-tree*, représenté sur la figure 3 (partie droite). Dans le max-tree, les nœuds ne stockent que les points appartenant à $L_\lambda(f) - L_{\lambda+1}(f)$, évitant ainsi la redondance de données.

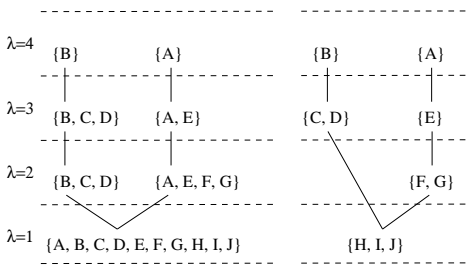

FIG. 3: Arbre des composantes (à gauche) et *max-tree* (à droite).

Pour construire le max-tree d'une image donnée, nous utilisons la représentation arborescente la plus compacte d'une image : un arbre enraciné défini par une fonction de relation de parenté, notée *parent*, et encodée sous forme d'une image 2D. Cette fonction est telle que $parent(p)$ est un point 2D. Lorsqu'un nœud du max-tree contient plusieurs points, nous choisissons son dernier point (au sens de la relation $\mathcal{R}$) comme représentant de ce nœud. Cet élément, dit canonique, est également appelé « racine du niveau » (*level root*) dans la littérature. Ces éléments sont désignés par des lettres cerclées sur les figures 2 et 4. L'élément canonique correspondant au nœud racine du max-tree est appelé « élément racine ». Cet élément – J dans notre exemple – est représenté dans un double cercle.

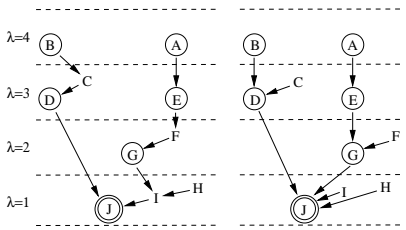

FIG. 4: Un arbre valide (à gauche) et sa forme canonique (à droite). La relation de parenté entre les points est figurée par des flèches ; par exemple $parent(C) = D$.

Soit Γ une composante correspondant à un nœud du max-tree, p_Γ son élément canonique et p_r l'élément racine. La fonction *parent* que nous avons construite doit vérifier les quatre propriétés suivantes :

1. $parent(p_r) = p_r$
2. $\forall\, p \neq p_r,\ p\ \mathcal{R}\ parent(p)$ donc $\forall\, p \neq p_r,\ p\ \mathcal{R}\ p_r$ et $\forall\, p,\ f(p_r) \leq f(parent(p)) \leq f(p)$
3. p est canonique ssi $p = p_r \ \vee\ f(parent(p)) < f(p)$
4. $\forall p,\ \ p \in \Gamma \ \Leftrightarrow\ f(p) = f(p_\Gamma) \ \wedge\ \exists\, i, parent^i(p) = p_\Gamma$ (où $parent^i$ est la i^e application de *parent*)

sur des données produites par TERAPIX et le Centre de Données Astronomiques du Canada comme participation à la campagne d'acquisition « CFHT Legacy Survey » du Télescope Canada-France-Hawaii, un projet mené en collaboration entre le NRC et le CNRS.

donc $\forall p \in \Gamma,\ p = p_\Gamma \ \vee\ p\ \mathcal{R}\ p_\Gamma$.

L'algorithme COMPUTE-TREE de la figure 5 produit une fonction *parent* « correcte », c'est-à-dire, une fonction qui satisfait les propriétés ci-dessus. Les deux arbres de la figure 4 sont des représentations valides de la fonction *parent* de l'image, mais celui de droite vérifie une propriété supplémentaire :

5. $\forall p$, $parent(p)$ est un élément canonique.

L'algorithme CANONIZE-TREE de la figure 5 transforme toute relation *parent* valide de telle sorte que la dernière propriété soit vérifiée. L'arbre résultant a désormais la forme la plus simple pouvant être attendue. De plus, nous avons un isomorphisme entre une image et sa représentation canonique.

```
FIND-ROOT(x)
1  if zpar(x) = x  then return  x
2  else { zpar(x) ← FIND-ROOT(zpar(x)) ; return zpar(x) }
COMPUTE-TREE(f)
1  for each p, zpar(p) ← undef
2  R ← REVERSE-SORT(f)    // traduit R comme un tableau
3  for  each p ∈ R in direct order
4    parent(p) ← p ; zpar(p) ← p
5    for  each n ∈ N(p) such as zpar(n) ≠ undef
6      r ← FIND-ROOT(n)
7      if  r ≠ p  then { parent(r) ← p ; zpar(r) ← p }
8  DEALLOCATE(zpar)
9  return pair(R, parent)    // une fonction « correcte »
CANONIZE-TREE(parent, f)
1  for  each p ∈ R in reverse order
2    q ← parent(p)
3    if  f(parent(q)) = f(q)  then  parent(p) ← parent(q)
4  return parent    // une fonction « canonisée »
```

FIG. 5: Proposition d'algorithme

La caractéristique-clef de notre proposition est de s'appuyer sur une version *non* compressée de l'algorithme d'union-find pour calculer la fonction *parent*. Cependant, la fonction issue de COMPUTE-TREE, étant correcte, inclut d'une certaine façon le max-tree, tel qu'on peut l'observer dans la partie gauche de la figure 4. L'obtention d'une représentation simple et compressée du max-tree est déléguée à la routine CANONIZE-TREE. Pour calculer *parent* efficacement, un accès rapide aux éléments racines temporaires (les éléments canoniques des composantes connectées de l'ensemble des points déjà traités) est nécessaire. Celui-ci est obtenu grâce une structure auxiliaire d'union-find compressée, à savoir *zpar*.

3.2 Calcul d'attributs et marquage

Le résultat de l'algorithme est constitué du tableau R contenant des points triés selon $\mathcal{R}$, et de l'image *parent*. Par rapport à d'autres implémentations d'arbres de composantes, nous ne stockons pas la relation de filiation. Malgré cela, il est possible de calculer des attributs sur les nœuds. Pour cela, il est nécessaire de « pousser » l'information des enfants vers les parents. La figure 6 illustre ce principe avec l'attribut classique « aire ».

```
COMPUTE-AREA(f, R, parent)
1  for  each p ∈ R, area(p) ← 1  // initialisation
2  for  each p ∈ R in direct order
3    area(parent(p)) ← area(parent(p)) + area(p)  // MÀJ
```

FIG. 6: Exemple de calcul d'attribut

À la fin, chaque élément canonique p_Γ dispose de la valeur d'attribut correcte. Notre approche est donc un processus en trois étapes : 1. calculer l'arbre des composantes ; 2. calculer des attributs pour chaque nœud de l'arbre ; 3. étiqueter les nœuds selon un critère calculé sur les attributs. La dissociation de ces trois étapes permet un processus interactif : séparer la construction de l'arbre des composantes du calcul des attributs permet à l'utilisateur de choisir l'ensemble des attributs à calculer, selon la forme du résultat attendu, pour un coût très faible. De même, une passe d'étiquetage séparée permet une modification interactive des paramètres utilisés dans les critères (par exemple, la taille de l'aire maximale lors de la recherche de petits objets), avec visualisation immédiate.

3.3 Gestion des valeurs à virgule flottante

Lorsque les valeurs ne sont pas quantifiées, mais sont des nombres à virgules flottantes, tels que dans les images astronomiques, la quasi-totalité des points sont des éléments canoniques. Autrement dit, il n'y a pratiquement pas de zone plates, et la plupart des nœuds du max-tree contiennent un unique point. Une propriété intéressante de l'algorithme donné dans la section précédente est qu'il s'applique *tel quel* sur des données à virgule flottantes. L'étape de canonisation devient alors inutile.

4 Travaux liés et comparaisons

Plusieurs approches ont été explorées pour calculer l'arbre de composantes d'une image, divisées en deux catégories principales : les méthodes à base de queue de priorité, calculant l'arbre par accumulation ; et celles à base d'union-find, générant des forêts d'ensembles disjoints.

Jones [2] propose un algorithme de la première catégorie, comme généralisation de l'algorithme de Vincent [8]. Une autre méthode est exposée par Salembier *et al.* [6]. Les deux sont pénalisées par des images ayant une haute quantification, à cause du coût de mise à jour de la structure de queue (insertion et suppression de pixels). Les approches à base d'union-find s'appuient sur le calcul de forêt d'ensembles discrets de Tarjan [7]. Meijster [3] a présenté la première méthode accélérant la procédure FIND-ROOT en utilisant une routine FIND-LEVEL-ROOT (compression de chemin par niveau). Cependant, cette approche est inutilisable avec des images à valeurs hautement quantifiées, car elles contiennent trop de niveaux différents pour que la compression apporte un gain. Najman et Couprie [5] ont également défini un « processus d'émergence » utilisant deux forêts d'ensembles disjoints pour éviter la création d'arbres dégénérés (déséquilibrés). Bien que cette méthode a des temps d'exécution comparables à ceux de notre proposition, elle nécessite trop d'espace mémoire (environ cinq fois la taille de l'entrée) pour pouvoir être utilisée raisonnablement sur des images en haute résolution sur les machines actuelles.

Les résultats suivants ont été obtenus sur un ordinateur à base de processeur Bi-Xeon à 3 Ghz, avec 2×1 Mo de mémoire cache et 4 Go de mémoire vive, fonctionnant avec GNU/Linux. La première série de tests a été conduite sur des images 16-bit, obtenues par échantillonnage et égalisation d'images flottantes (voir la figure 7). Tant l'algorithme de Najman et Couprie que le nôtre surpassent l'algorithme de Salembier *et al.*, dont la courbe croît plus vite que celle des deux premiers. Notre algorithme obtient les meilleurs résultats dans le cas 16-bit.

L'algorithme de Salembier *et al.* n'a pas été conçu pour fonctionner sur des images à valeurs hautement quantifiées. Bien que nous avons utilisé l'algorithme originel pour le cas 16-bits, nous avons écrit une version adaptée au cas flottant, pour éviter l'utilisation de structures de taille proportionelle au nombre de niveaux présents dans l'image.

Le cas flottant fait apparaître de nouveaux problèmes encore non observés dans les calculs d'arbres de composantes. En particulier, le fait qu'un niveau soit rarement présent plus d'une fois dans une image (naturelle) à va-

leurs flottantes affecte l'algorithme de Salembier *et al.* au point que ses temps d'exécution sont incomparables avec ceux des autres algorithmes sur des images à valeurs flottantes, par exemple plus de deux jours pour une image de 5 millions de pixels (résultat non représenté sur la figure 7). Le coût de mise à jour de la queue est vraisemblablement trop élevé lorsque l'algorithme doit stocker un nombre de niveaux beaucoup plus important que dans le cas d'images plus faiblement quantifiées. L'algorithme de Najman et Couprie est celui qui obtient les meilleurs résultats, ce qui démontre la pertinence de la technique de classement (*ranking*) sur des images à valeurs flottantes.

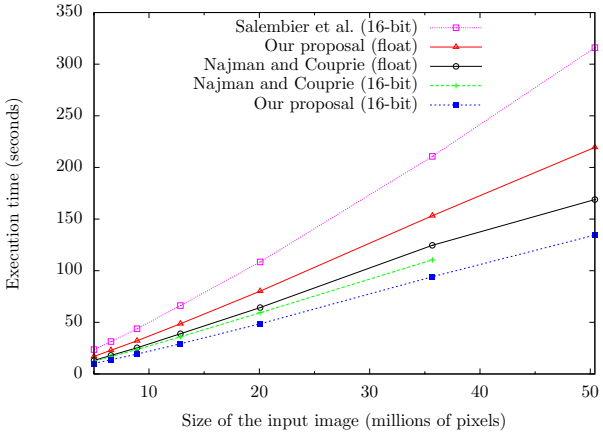

FIG. 7: Temps d'exécution sur des images 16-bit et flottantes.

La structure utilisée dans l'algorithme de Najman et Couprie et notre méthode code seulement des *points*. La mémoire utilisée dépend donc uniquement du nombre de points dans l'image en entrée (et la façon dont ces points sont représentés), et non de la taille des valeurs portées par chaque pixel – dans le cas présent, 2 octets (images sur 16 bits) ou 4 octets (image flottantes). Dans cette analyse de la complexité spatiale, nous considérons qu'un point est codé comme un index sur 32 bits (c'est-à-dire, un décalage depuis le début de l'image). L'algorithme de Najman et Couprie est celui qui utilise le plus de mémoire.

Une estimation grossière de l'empreinte mémoire pour une image de n points est de n points pour la relation $\mathcal{R}$, $2n$ points pour la relation $\mathcal{Q}_{\text{tree}}$ (comportant l'arbre lui-même et la structure de rang), $2n$ points pour $\mathcal{Q}_{\text{node}}$, et n pixels ou plus pour stocker l'arbre de composantes lui-même sous la forme de la structure *children*, ce qui correspond à une occupation mémoire de 6 fois la taille de l'entrée dans le cas flottant. Notre proposition a une empreinte mémoire de 3 fois la taille de l'entrée dans le cas flottant (pour les images $\mathcal{R}$, *parent* et *zpar*), c'est-à-dire deux fois moins que l'algorithme précédent. La quantité de mémoire utilisée par l'algorithme de Salembier *et al.* est plus difficile à évaluer, car la taille maximale atteinte par la queue hiérarchique dépend de la nature des données. Cependant, la complexité en temps de l'algorithme va de pair avec la complexité en espace, puisque celui-ci a saturé la mémoire du système en travaillant sur une image flottante de 8,9 millions de pixels.

5 Application aux données

Dans cette section, nous présentons des esquisses de critères pour reconnaître des objets dans des images d'astronomie. La représentation sous forme d'arbre des composantes est suffisamment flexible pour que les objets et les défauts puissent être identifiés. En premier lieu, pour restreindre cette identification à certaines parties de l'arbre, nous supprimons les branches non significatives. En pratique, les zones plates de l'arrière-plan sont détectées comme n'ayant pas leur isomagnitude à $\frac{f_{\max}}{2}$ deux fois plus élevée que celle de la région environnante $\{p|f(p) \in [\frac{f_{\max}}{4}; \frac{f_{\max}}{2}]\}$. Lors de l'identification, par exemple, les étoiles sont reconnues comme de très petite composantes suivant la fonction instrumentale (PSF).

Comme autre exemple, les traces de satellites sont des objets long et fins sans luminosité uniforme (à la différence des objets « centrés »). Elles sont identifiées comme de grandes composantes (largeur (l) + hauteur (h) supérieure à un seuil), de telle sorte que $\operatorname{var} x + \operatorname{var} y > \frac{(l+h)^2}{14}$, où $\operatorname{var} x$ et $\operatorname{var} y$ sont respectivement la variance spatiale de la composante selon l'axe X et l'axe Y.

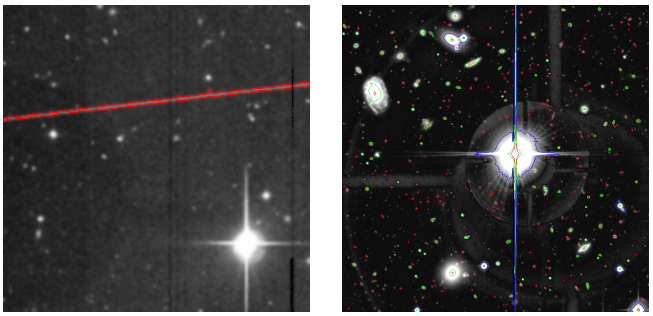

FIG. 8: À gauche : trace de satellite.
À droite : contours de la composante à $\frac{f_{\max}}{2}$, $\frac{f_{\max}}{4}$ et $\frac{f_{\max}}{10}$.

6 Conclusion

Dans cet article, nous avons présenté un nouvel algorithme de calcul de l'arbre des composantes. Ses avantages sont multiples ; tout d'abord, il se révèle performant sur les images à valeurs hautement quantifiées ou sans quantification. Son efficacité est comparable à celle de l'algorithme connu le plus rapide. Par ailleurs, il exige deux fois moins de mémoire que ce dernier, ce qui est capital lors de traitements sur des données volumineuses. Nous avons également présenté une utilisation de cet algorithme pour l'identification d'objets dans des images d'astronomie.

Références

[1] T. Géraud. Ruminations on Tarjan's Union-Find algorithm and connected operators. In *Mathematical Morphology : Proc. of the Intl. Symp. (ISMM)*, pages 105–116. Springer, 2005.

[2] R. Jones. Component trees for image filtering and segmentation. In E. Coyle, editor, *IEEE Workshop on Nonlinear Signal and Image Processing*, Mackinac Island, September 1997.

[3] A. Meijster. *Efficient sequential and parallel algorithms for morphological image processing.* PhD thesis, University of Groningen (NL), March 2004.

[4] A. Meijster and M. H. F. Wilkinson. A comparison of algorithms for connected set openings and closings. *IEEE Trans. Pattern Anal. Machine Intell.*, 24(4) :484–494, 2002.

[5] L. Najman and M. Couprie. Building the component tree in quasi-linear time. *IEEE Trans. Image Processing*, 15(11) :3531–3539, November 2006.

[6] P. Salembier, A. Oliveras, and L. Garrido. Antiextensive connected operators for image and sequence processing. *IEEE Trans. Image Processing*, 7(4) :555–570, 1998.

[7] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2) :215–225, 1975.

[8] L. Vincent. Grayscale area openings and closings : their applications and efficient implementation. In *Intl. Symp. on Mathematical Morphology*, pages 22–27, 1993.