Mini projet - Segmentation Mean Shift

Pierre-Antoine Augereau & Kévin Polisano

30/11/2012

1 Algorithme de segmentation par mean shift

Nous décrivons ici le principe de fonctionnement général de la procédure du mean shift. Etant donné n points (x_i) dans un espace de dimension d, $\mathbb{R}^d$. On souhaite pouvoir regrouper ces points en différents "clusters". Intuitivement on aimerait regrouper les points qui sont proches les uns des autres dans un même cluster. Plus formellement si on disposait de la fonction de densité de probabilité, qui décrit en quelque sorte à un endroit donné de l'espace si il y a une forte densité de points dans ce voisinage ou non, on pourrait repérer les *modes* de celle-ci, c'est-à-dire les maximum locaux qui feraient d'autant de clusters.

La fonction de densité de probabilité (p.d.f) f en un point x de l'espace $\mathbb{R}^d$ est définie par :

$$f_{h,K}(x) = \frac{c_{k,d}}{nh^d} \sum_{i=1}^{n} k\left(\left\| \frac{x - x_i}{h} \right\|^2 \right)$$

c'est-à-dire qui est calculée en examinant les voisins du point x, où k est un noyau statistique, qui va accorder plus ou moins d'importance à ces voisins suivant leur distance au point x : $\|x - x_i\|$, il faut donc au départ se fixer une métrique dans l'espace. La présence du paramètre h provient du fait qu'on examine les voisins dans une boule de rayon h autour de x. Par exemple le noyau gaussien défini par $k(u) = \frac{1}{\sqrt{2\pi}} e^{-u^2}$ indiquera une densité faible dès lors que les voisins x_i sont légèrement éloignés de x, tandis qu'un noyau d'Epanechnikov $k(u) = \frac{3}{4}(1 - u^2)\chi_{[-1,1]}(u)$ sera moins strict et indiquera une densité plus élevée.

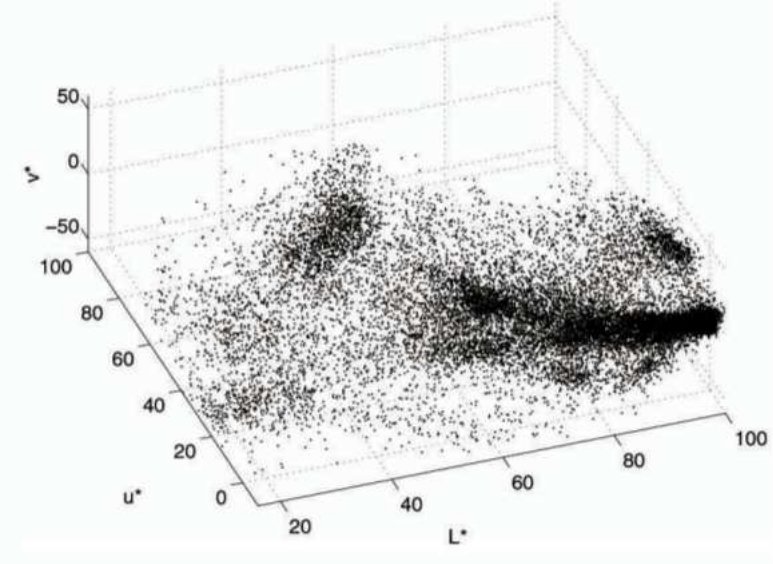

Figure 1: Nuage de points présentant des zones de densité différentes

Une fois que l'on s'est fixé une distance et un noyau on est donc en mesure de définir notre p.d.f. Déterminer ses maximum locaux revient à localiser les x tels que $\nabla f_{h,K}(x) = 0$. La puissance de la procédure du mean shift repose sur le fait qu'on est capable de localiser ces zéros sans avoir à estimer la p.d.f. En effet si on calcule formellement la gradient de $f_{h,K}$ on aboutit à l'expression suivante :

$$\nabla f_{h,K}(x) = f_{h,G}(x)\frac{2c_{k,d}}{h^2 c_{g,d}} m_{h,G}(x)$$

où G est le noyau dérivé de K ($g(x) = -k'(x)$) et $m_{h,G}(x)$ est un déplacement moyen du vecteur x (d'où le nom du mean shift) défini par :

$$m_{h,G}(x) = \underbrace{\frac{\sum_{i=1}^{n} x_i g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)}}_{\phi(x)} - x$$

Le "moyen" est à prendre au sens d'une moyenne pondérée des voisins de x pris dans une boule $\mathcal{B}(x,h)$. Comme $f_{h,G}$ ne s'annule pas (c'est une densité de probabilité), alors déterminer les zéros du gradient revient à déterminer les zéros de $m_{h,G}$, soit de déterminer un point fixe de ϕ. Définissons la suite

de points (y_j) suivants :

$$\begin{cases} y_1 = x \\ \\ y_{j+1} = \dfrac{\sum_{i=1}^{n} x_i g\left(\left\|\frac{y_j - x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} g\left(\left\|\frac{x - x_i}{h}\right\|^2\right)} \end{cases}$$

où à chaque itération les (x_i) sont les voisins du point y_j dans une boule $B(y_j, h)$. Alors il est démontré que cette suite (y_j) converge $\lim_{j \to +\infty} y_j = y_c$, et donc que : $m_{h,G}(y_j) = y_{j+1} - y_j \to 0$. Par conséquent cette limite y_c est le point fixe de ϕ, c'est-à-dire qu'à partir d'un certain rang la moyenne pondérée $\phi(y_{i_0})$ avec les voisins ne bouge plus $\phi(y_{i_0}) \simeq y_{i_0}$. Finalement cette procédure de mean shift est équivalente à une descente de gradient, et nous permet pour chaque point x de l'espace de déterminer vers quel mode il se déplace, et on l'ajoute au cluster correspondant à ce mode. En pratique on ne peut pas calculer un nombre infini d'itérées donc on s'arrête quand $\|m_{h,G}(y_{j_f})\| \leqslant \epsilon$, ϵ étant la condition d'arrêt. Le schéma ci-dessous décrit le calcul d'une itérée : on part de y_{j-1} en (a), on détermine ses voisins en (b) et on effectue la moyenne pondérée de ceux-ci, on obtient un nouveau point y_j en (c). En répétant cette étape tant que le critère d'arrêt n'est pas vérifié, on se déplace comme en (e) pour vers un point $y_{j_f} \equiv x_c$ en (f).

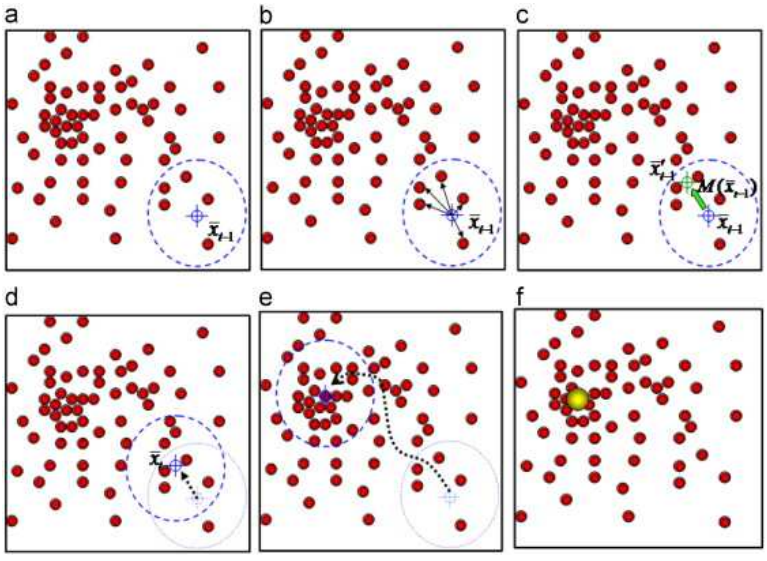

Figure 2: Illustration de la procédure du mean shift

La segmentation mean shift s'effectue donc en 2 étapes :

1) Pour chaque point x de l'espace, calculer vers quel point x_c il converge. On obtient un nouvel ensemble de points (x_c) "image".

2) La deuxième étape consiste à identifier ces groupes de points très proches (à distance au plus 2ϵ par inégalité triangulaire). C'est sur ce deuxième point que l'article reste complètement silencieux et auquel nous avons proposé *une* possibilité.

2 Implémentation et optimisation

Nous avons longuement réfléchis au diagramme de classes de notre programme, de façon à être le plus modulaire possible et transparent par rapport à l'utilisateur. En effet le programme principal `Vision.cpp` demande simplement les paramètres de segmentation et appelle sur l'image considérée la méthode de segmentation avec ces derniers, qui renvoie l'image segmentée

```
Image imgsegMS = img.MeanShift(K,b,tol).
```

Les paramètres en question utilisés dans la méthode de segmentation MeanShift sont :

```
double tol;
double b;
double hs;
double hr;
bool pos;
```

Les trois derniers paramètres sont intégrés dans l'objet `K` instance de la classe `Noyau` et construit par `Noyau(pos,color,hs,hr,d,uniforme)` où `d` est elle-même une instance de la classe `Distance` construit via `Distance(euclidienne)` ; et `uniforme` est une fonction k définissant le type de noyau statistique utilisé. Autrement dit la classe `Noyau` regroupe le noyau statistique k et la métrique $d(x, y)$ utilisée (la norme induite est $\|x\|_d = d(x, 0)$), qui sont les deux éléments requis pour calculer :

$$K(x) = \frac{C}{h_s^2 h_r^3} k\left(\left\|\frac{x^s}{h_s}\right\|_d\right) k\left(\left\|\frac{x^r}{h_r}\right\|_d\right)$$

qui dans notre implémentation s'obtient simplement en surchargeant l'opérateur des parenthèses `double Noyau::operator()(vector<double> & x)`

Rappelons que la première étape de la segmentation consiste pour chaque point de l'espace $X = (x, y, r, g, b)$ stocké dans un `\vector<double>` à calculer vers quel point il converge X_c par la procédure du mean shift. C'est le rôle de la fonction

```
void move(vector<double> & x, vector<double> & xc,
Noyau * K, double tol, double b)
```

qui prend en entrée le vecteur X et qui stocke le résultat dans X_c. Les itérées successives sont calculées dans la fonction `move` via la fonction

```
void moveOneStep(vector<double> & x, vector<double> & y,
Noyau * K, double b)
```

L'avantage de pouvoir écrire les vecteurs en place est qu'on ne créé pas de nouveaux vecteurs ni copies, seulement 2 vector sont nécessaires pour calculer les itérées en switchant le rôle entrée/sortie à chaque tour (voir la présence d'un compteur modulo 2 dans le code). De cette façon nous avons pu diviser par 3 le temps d'exécution de notre première version.

Rappelons qu'à chaque itération y_j (donc chaque déplacement du point) on a besoin de déterminer les voisins de y_j dans la boule $\mathcal{B}(y_j, h)$, c'est pourquoi dans la fonction `moveOneStep` nous faisons appel à la méthode

```
getVoisinsLocal(vector<double> & x, Noyau * K,double b)
```

A l'origine nous avions codé la recherche de voisins dans cette boule de manière naïve, c'est-à-dire en parcourant les n points de l'espace et en testant si ils sont dans cette boule, ce qui est beaucoup trop gourmand dès lors que l'image n'est plus une miniature. A titre informatif sur une image 256×256 le calcul de la première étape prenait plus d'une heure. Nous avons du coder un algorithme plus ingénieux, nommé "kNN search", décrit dans l'article "Fast exact k nearest neighbors search using an orthogonal search tree". Nous exposons le fonctionnement de notre implémentation en annexe.

La deuxième étape consiste ensuite à regrouper les points images (par la fonction `move`) en clusters. Pour se faire nous avons attribué à chaque point de l'espace x un entier représentant le cluster (ou la classe) auquel il appartient. Nous avons nommé cette structure `Pixel`.

En fait notre programme effectue les étapes 1 et 2 simultanément. On prend un point de l'espace x, on calcule son point image x_c. On détermine

les voisins de x_c, si au moins un d'entre eux est étiquetté d'un numéro de classe on choisit l'étiquette minimale parmi toutes celles des voisins et on met à jour leur étiquette par cette dernière, ainsi que celle de x. C'est ce qu'on appelle le "speed-up".

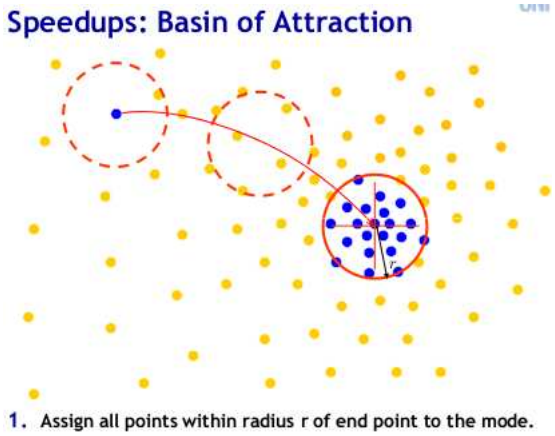

Figure 3: Illustration du speed up

Dans le cas où la boule $\mathcal{B}(x_c, b)$ est à cheval sur deux bassins relatifs à 2 modes A et C, la première version étiquettera tous les voisins de x_c avec l'étiquette du premier voisin trouvé (ici arbitrairement V_1) ce qui aura pour effet de fusionner certains modes comme nous le verrons dans les résultats obtenus.

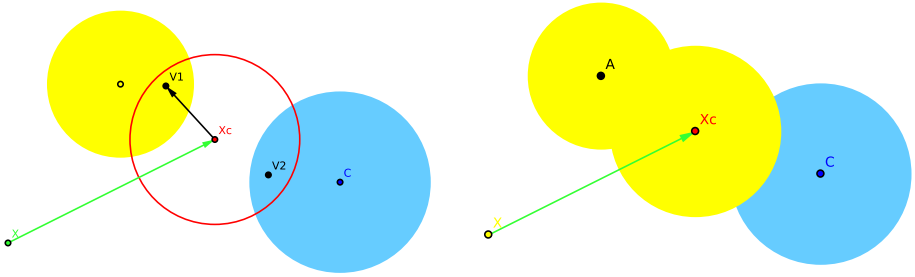

Figure 4: Illustration du cas pathologique où le voisinage de x_c est à cheval sur deux modes

Nous avons produis une deuxième version, qui contrairement à la précédente, dans le cas où x_c a plusieurs voisins, x reçoit la plus petite étiquette mais les voisins ne sont pas mis à jour. Bien sûr cette version sera plus longue à l'exécution (car il y a moins d'étiquettage à chaque tour), mais donnera

des résultats plus satisfaisants.

Enfin une troisième version implémentée en dernière minute qui consiste à prendre en compte la distance de x_c au "mode" (assimilé au premier point formant un cluster). Sur la figure 4 on calcule la distance $d_1 = \|x_c - A\|$ et $d_2 = \|x_c - C\|$, on compare ces distances à un seuil (typiquement ici on a pris $8 * d$) et si elles sont en dessous de ce seuil on conserve la plus petite distance, par exemple d_1 et on place x (ainsi que les voisins non marqués de x_c) dans le cluster de même étiquette que v_1. De cette façon on n'a plus du tout de phénomène de "bavure", les clusters sont bien délimités, mais cela a tendance à sur-segmenter.

remarque : signalons que pour repérer les fonctions chronophages dans notre programme (dans l'optique de les optimiser) nous avons utilisé un outil de profiling intéressant, à savoir `gprof`, qui permet de connaître le nombre d'appels et le temps passé dans chacune des fonctions et primitives. Nous avons pu ainsi réduire un nombre important d'allocations ou recopies superflues.

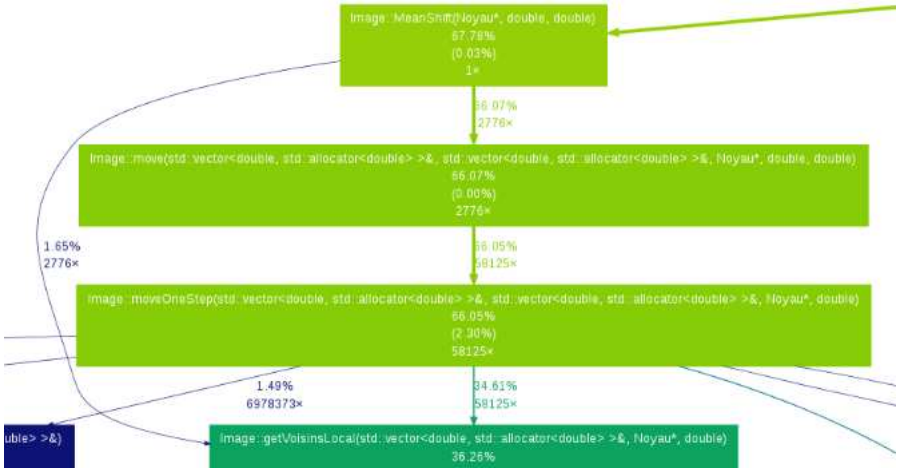

Figure 5: Capture d'écran d'une partie du profiling

3 Résultats obtenus

On considère l'image en couleur `grenouille.ppm` de taille 256×256 que l'on souhaite segmenter. On se place dans l'espace $\mathbb{R}^5$ regroupant la position et la couleur d'un pixel.

Figure 6: Image originale à segmenter

On prend les paramètres suivants, une distance euclidienne et le noyau d'Epanechnikov. Le temps d'exécution est sur notre machine personnelle de l'ordre de 30s (dont 7s sont vouées à la construction de l'arbre).

```
double tol = 0.1;
double b = 20;
double hs = b;
double hr = b;
```

On obtient l'image en niveau de gris suivante, où chaque pixel possède un niveau de gris égal au numéro de sa classe. Notre algorithme de segmentation renvoie un résultat cohérent.

Figure 7: Segmentation en classes

Pour une meilleure visualisation nous avons décidé d'affecter à un pixel la couleur correspondant à la moyenne des couleurs de tous les pixels de la même classe. Nous avons, comme dans l'algorithme du Kmean, calculé le barycentre à la volée pour limiter le nombre de calculs. Voici le résultat obtenu pour nos deux versions de l'étape de regroupement en clusters :

Figure 8: Clustering version 1

Figure 9: Clustering version 2

Bien que le résultat soit satisfaisant dès la première version, on constate des problèmes, notamment dans le fond, qui sont en partie résolus avec la version 2. En effet concernant l'assignation des étiquettes, dans la version 1, nous propageons l'étiquette minimale sur tous les voisins du point image, et nous avons probablement rencontré un cas pathologique comme illustré sur la figure 4.

La version 3 fonctionne encore davantage et résout complètement les problèmes cités précédemment :

Figure 10: Clustering version 3

On voit sur l'image en niveau de gris représentant les classes, que la prise en compte des positions influe beaucoup (trop), la segmentation quand nous limitons à l'espace RGB est préférable (voir figure 12 ci-dessous).

On peut ensuite jouer sur les paramètres, par exemple si on augmente la taille de la boule en prenant $b = 20$ on obtient une image sous-segmentée, car deux modes proches vont de nouveau être "fusionnés" puisque la rayon de la boule considérée chevauche les deux modes.

Figure 11: Segmentation avec un voisinage trop grand

Figure 12: Segmentation dans l'espace RGB

L'avantage du mean shift par rapport au Kmean est qu'il ne requiert le nombre de clusters (ce qui est assez contraignant car on ne sait pas *a priori* en combien de parties est composée une image), il en détecte un nombre donné en fonction de l'image et de la taille des boules considérées (qui est une contrainte également). C'est pourquoi le mean shift a tendance à sur-segmenter les images.

4 Annexe : kNN-search

Avant propos : Après avoir consulté différents articles sur le problème de la recherche des plus proches voisins, nous avons trouvé deux grandes familes d'algorithmes : ceux qui s'appuient sur la construction d'un arbre de recherche n-aire, et ceux qui tentent de réduire le nombre de calculs à effectuer en sélectionnant les points à l'aide de méthodes statistiques. Nous avons opté pour la première solution pour sa simplicité théorique et son côté intuitif. Dans cette classe, chaque algorithme diffère soit par la méthode de construction de l'arbre soit par la méthode de recherche à l'intérieur de celui-ci. Suite à la lecture d'un article présentant les avantages et les inconvénients de chaque type de méthode, nous avons choisi la méthode de construction par projection orthogonale, qui est une simplification de la méthode de construction par projection par axe de composante principale. Nous avons fait ce choix car cette méthode est relativement indépendante du jeu de données considéré et que le compromis entre temps de recherche et temps de calcul de l'arbre nous paraissait intéressant, sachant que généralement plus l'arbre est long à calculer plus les recherches sont ensuite efficaces.

Explication de la construction. Pour simplifier on considèrera la méthode de construction d'un arbre binaire. Étant donnée un ensemble de points d'un espace de dimension d, on considère que tous ses points sont stockés dans un noeud de l'arbre. On choisit alors un vecteur sur lequel on les projette. Ensuite on sépare ces points en deux groupes selon la valeur de leur projection, on assigne la moitié inférieure à un node enfant, et l'autre moitié à un deuxième node. On stocke pour chaque node ses extremums et on répète le processus itérativement sur chaque node en choisissant un nouveau vecteur de projection, jusqu'à ce que l'arbre soit profond de d niveau, ou que le noeud ne contienne plus assez de points pour être séparé.

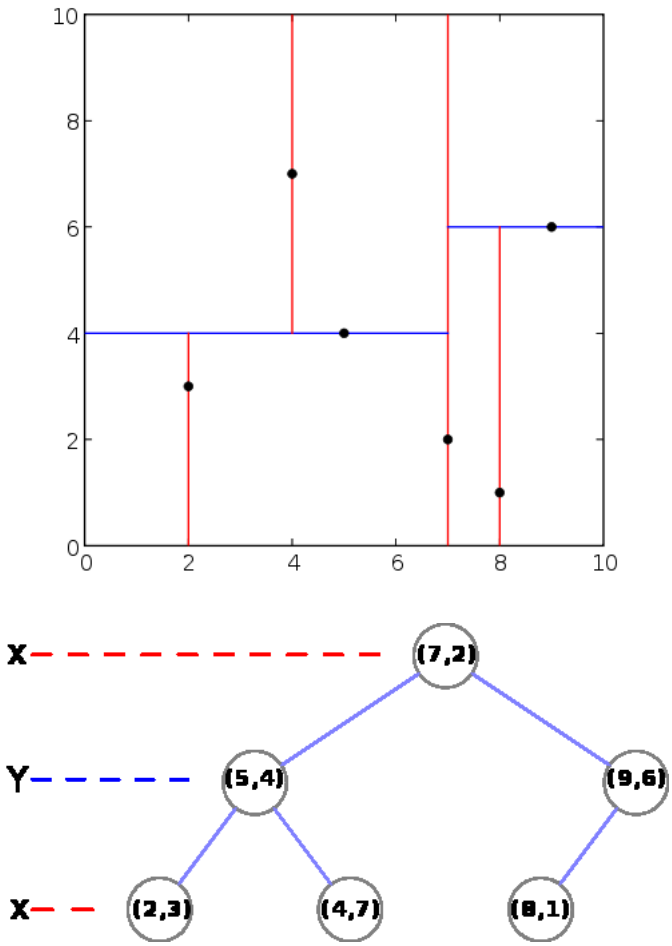

Figure 13: Découpage de l'espace et construction du *kd*-arbre

Optimisations de la construction. Elles se situent au niveau du choix de la séparation. On peut voir le résultat de la projection sur un vecteur d'un ensemble de points comme un histogramme. Ainsi, séparer intelligemment les points selon les modes de l'histogramme passerait par la détection de ces modes, autrement dit par l'application d'un mean shift, et le serpent se mords la queue si pour construire un arbre de recherche pour optimiser le mean shift on utilise le mean shift...

L'autre façon d'optimiser porte sur le choix du vecteur de projection. Une technique consiste à projetter les points sur les vecteurs composants les axes des composantes principales permettant une meilleure répartition des points lors des projections, alors que pour la construction à l'aide de projections or-

thogonales, on choisit une base orthogonale et on effetue la projection selon chaque axe dans un ordre donné. Pour une image comportant 2 dimensions spatiales et 3 dimensions de couleurs, il nous a semblé judicieux d'utiliser la base canonique de cet espace, car les deux axes principaux liés aux dimensions spatiales sont deux vecteurs de la base canonique et de plus cette base ne demande aucun calcul de projection ce qui simplifie bien les choses.

Une fois le calcul des vecteurs projecteurs et de leur projection éliminé, la partie coûteuse restante de la construction de l'arbre se situe au niveau de la séparation des vecteurs projetés en fonction de leurs valeurs de projection. Afin de réduire ce coût, on profite du fait que les points dans le domaine spatial de l'image sont une grille de pixels donc déjà rangés, et que dans l'espace de couleur il y a peu de valeurs possibles par rapport au nombre de points. On peut donc procéder à un tri par comptage qui se fait en $O(2n)$ comparé à un $O(n \ln n)$ pour la complexité optimale d'un tri par comparaison.

Recherche à l'intérieur de l'arbre. L'intérêt d'un arbre de recherche est que l'on peut facilement élaguer l'arbre en fonction de certains critères afin de ne pas visiter tout l'ensemble de points. L'idée pour la condition d'élagage est la suivante, étant donné un point $X = (x, y)$ du plan $\mathbb{R}^2$, on souhaite récupérer tous les points qui sont à une distance inférieure à 1 de ce point. On ne va certainement pas balayer tout l'espace afin de les trouver. On peut se limiter à l'ensemble $[x - 1, x + 1] \times [y - 1, y + 1]$. Et on peut facilement démontrer que tout point en dehors de cet ensemble est à une distance plus grande que 1 par rapport à X.

Autrement dit, tout node dont la distance entre le point considéré et le plus proche de ses extremums est plus grande que la valeur recherchée n'a pas besoin d'être visité. En pratique on ne considère qu'une distance partielle, (i.e. n'utilisant pas toutes les coordonnées du point) pour pratiquer l'élagage cela permet de gagner encore quelques cycles de calcul. De même on peut décider ou non de calculer la distance entre deux points en jouant sur les propriétés de la distance, en particulier l'inégalité triangulaire voir décrite au chapitre 3 de l'article étudié, ce qui permet, là encore, de gagner du temps lors de la recherche.