SEGMENTATION D'IMAGES PAR COUPE DE GRAPHE

JEMMALI Fadi
n° 54214

TIPE Informatique

Juin 2022

Définition et méthode de segmentation

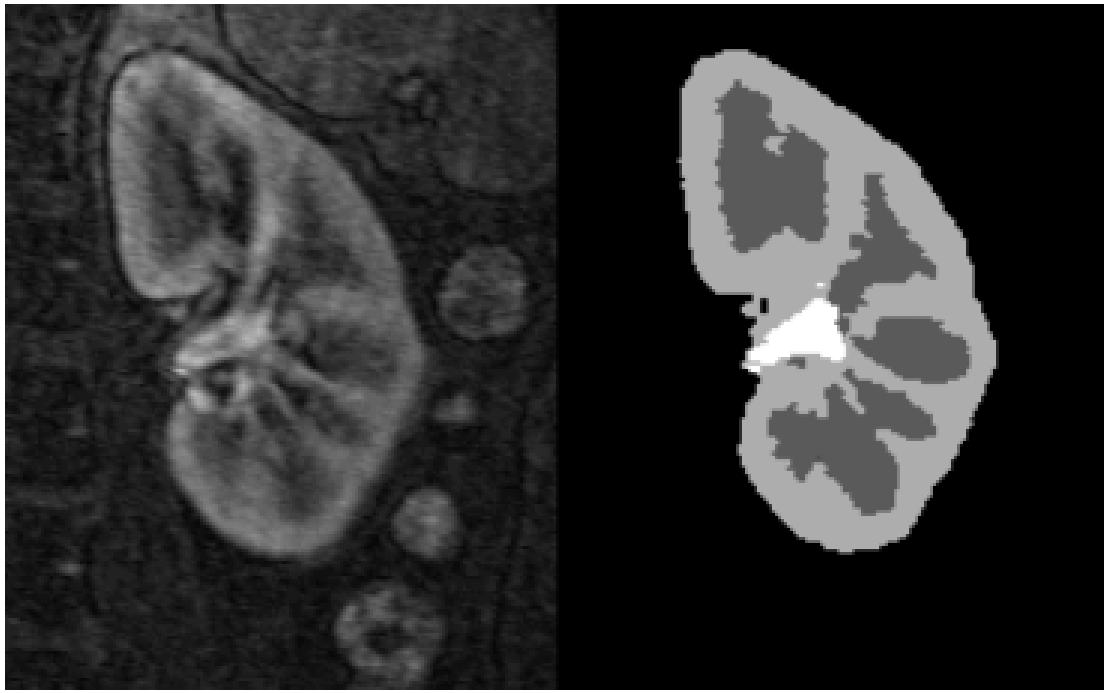


Figure 1 – Segmentation par coupe de graphe d'un rein [4]

- Division de l'image en zones basées sur une description spécifiée.
- Séparer l'objet et l'arrière-plan de l'image

Réseau de transport

Définition 2.1

Un réseau de transport est un graphe orienté $G = (S, A, c)$.
$\forall (u, v) \in A$, (u, v) est pondérée par $c(u, v) > 0$.
$\exists s, t \in S : d^-(s) = 0$ et $d^+(t) = 0$

Réseau de transport

Définition 2.1

Un réseau de transport est un graphe orienté $G = (S, A, c)$.
$\forall (u, v) \in A$, (u, v) est pondérée par $c(u, v) > 0$.
$\exists s, t \in S : d^-(s) = 0$ et $d^+(t) = 0$

- Si $(u, v) \notin A$, $c(u, v) = 0$.
- $\forall u \in S \setminus \{s, t\}$, il existe un chemin $s \longrightarrow u \longrightarrow t$.

Flot

Définition 2.2

Un flot dans le réseau G est une fonction $f : S^2 \longrightarrow \mathbb{R}$ qui vérifie :

- **Contrainte de capacité** : $\forall u, v \in S, f(u, v) \leq c(u, v)$
- **Anti-symétrie** : $\forall u, v \in S, f(u, v) = -f(v, u)$
- **Conservation du flot** : $\forall u \in S \setminus \{s, t\}, \sum_{v \in S} f(u, v) = 0$

Flot

Définition 2.2

Un flot dans le réseau G est une fonction $f : S^2 \longrightarrow \mathbb{R}$ qui vérifie :

- **Contrainte de capacité** : $\forall u, v \in S, f(u,v) \leq c(u,v)$
- **Anti-symétrie** : $\forall u, v \in S, f(u,v) = -f(v,u)$
- **Conservation du flot** : $\forall u \in S \setminus \{s,t\}, \sum_{v \in S} f(u,v) = 0$

Définition 2.3

La valeur d'un flot f est $|f| = \sum_{v \in S} f(s,v)$.

Un flot maximal est un flot f de valeur $|f|$ maximale.

Exemple de réseau de transport

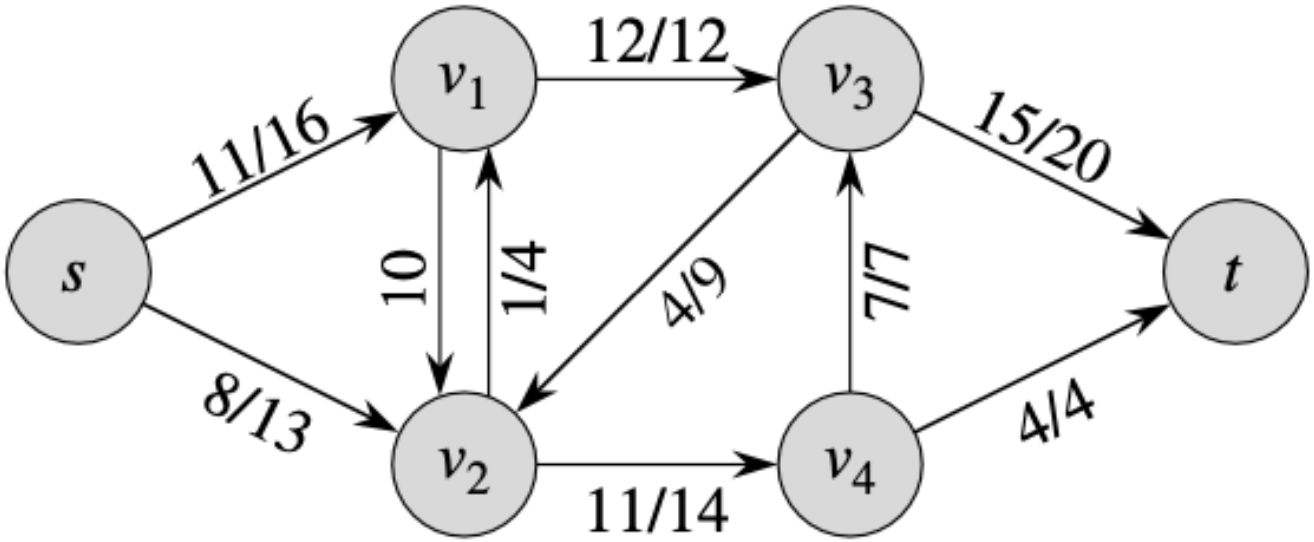


Figure 2 – Un réseau de transport G muni d'un flot f [2]

- Le flot et la capacité sont notés f/c.
- Seuls les flots positifs sont représentés.
- $|f| = 19$

Coupe dans un réseau de transport

Définition 2.4

Une coupe (E,T) de G est une partition de S t.q $s \in E$ et $t \in T$.

Coupe dans un réseau de transport

Définition 2.4

Une coupe (E,T) de G est une partition de S t.q $s \in E$ et $t \in T$.

Définition 2.5

La capacité d'une coupe (E,T) est :

$$c(E,T) = \sum_{u \in U} \sum_{v \in V} c(u,v).$$

Une coupe minimale est une coupe de capacité minimale.

Le flot net à travers (E,T) est :

$$f(E,T) = \sum_{u \in E} \sum_{v \in T} f(u,v).$$

Exemple de coupe

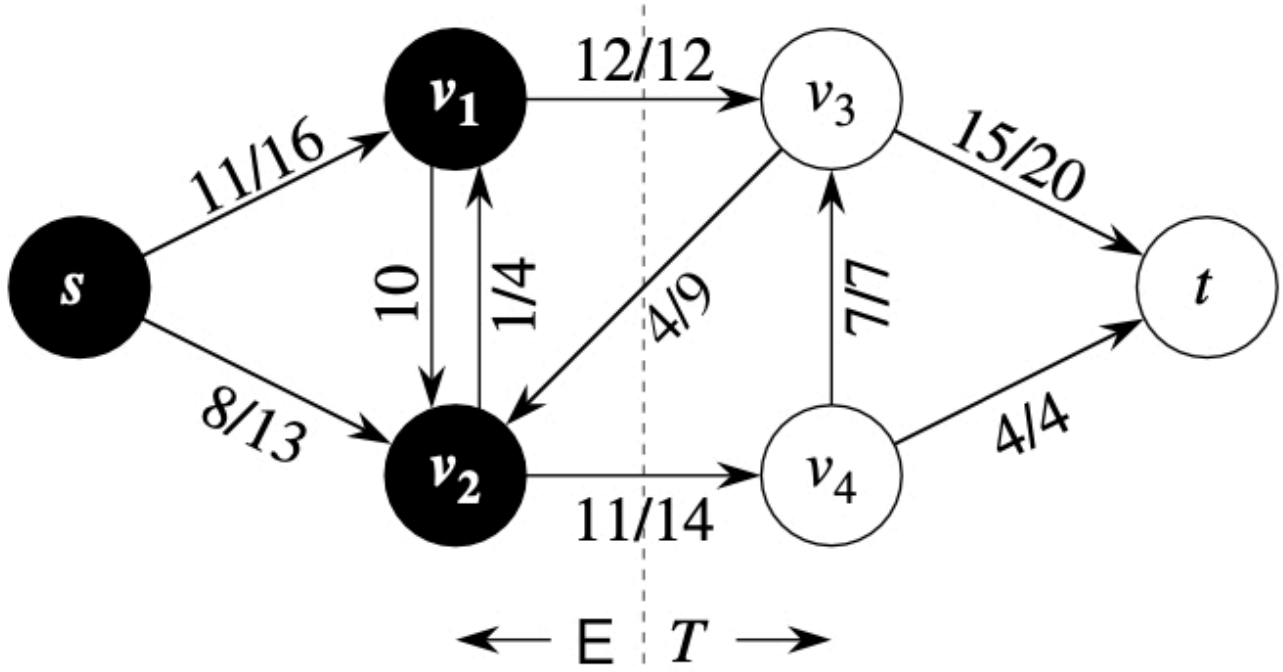


Figure 3 – Coupe (E,T) dans G [2]

- $c(E, T) = 26$.
- $f(E, T) = 19 = |f|$.

Lien entre flot et coupe

Proposition 2.1

$\forall$ coupe (E, T), $f(E, T) = |f|$.

Démonstration en annexe

Réseau résiduel

Définition 2.6

$\forall u, v \in S$, La capacité résiduelle de (u,v) est :

$$c_f(u, v) = c(u, v) - f(u, v)$$

Réseau résiduel

Définition 2.6

$\forall u, v \in S$, La capacité résiduelle de (u,v) est :

$$c_f(u, v) = c(u, v) - f(u, v)$$

Définition 2.7

Le réseau résiduel de G induit par f est $G_f = (S, A_f)$ où

$$A_f = \{(u, v) \in S^2 : c_f(u, v) > 0\}$$

Réseau résiduel

Définition 2.6

$\forall u, v \in S$, La capacité résiduelle de (u,v) est :

$$c_f(u, v) = c(u, v) - f(u, v)$$

Définition 2.7

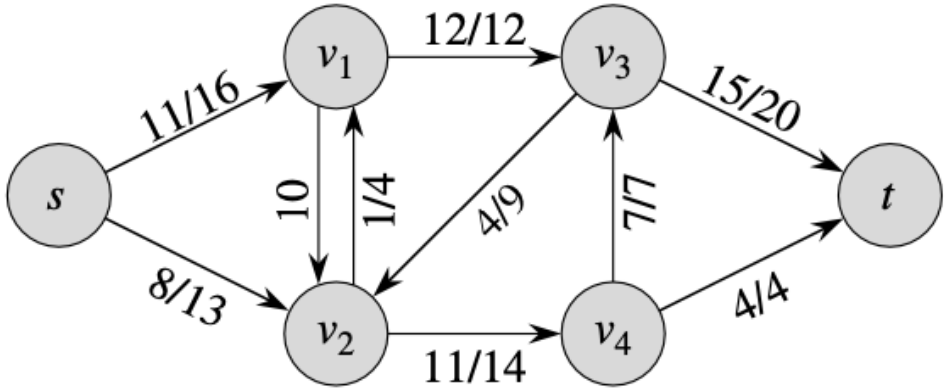
Le réseau résiduel de G induit par f est $G_f = (S, A_f)$ où

$$A_f = \{(u, v) \in S^2 : c_f(u, v) > 0\}$$

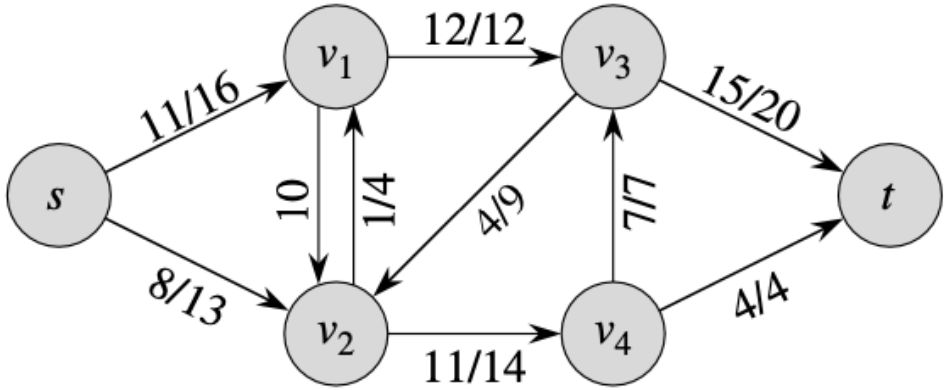
Définition 2.8

Un chemin améliorant p est un chemin simple de s vers t dans G_f.
La capacité résiduelle de p est :

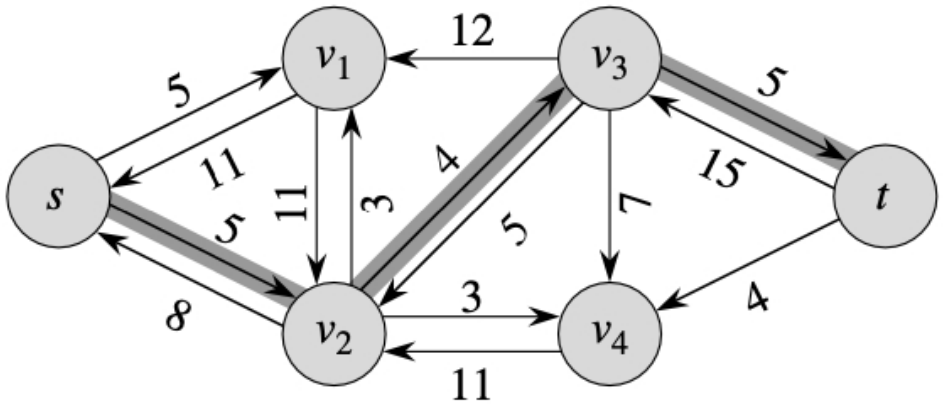
$$c_f(p) = min\{c_f(u, v) : (u, v) \text{ appartient à p}\}$$



(a) Le réseau de transport G de la figure 1 [2]



(a) Le réseau de transport G de la figure 1 [2]



(b) Le réseau résiduel G_f avec un chemin améliorant p en gris de capacité residuelle $c_f(p) = min(5, 4, 5) = 4$ [2]

Théorème Maxflow/Mincut

Théorème Maxflow/Mincut

Les conditions suivantes sont équivalentes :

1. f est un flot maximal dans G
2. Le réseau résiduel G_f ne contient aucun chemin améliorant
3. $|f| = c(E, T)$ pour une certaine coupe (E, T) de G

Démonstration en annexe

Identification d'une coupe minimale

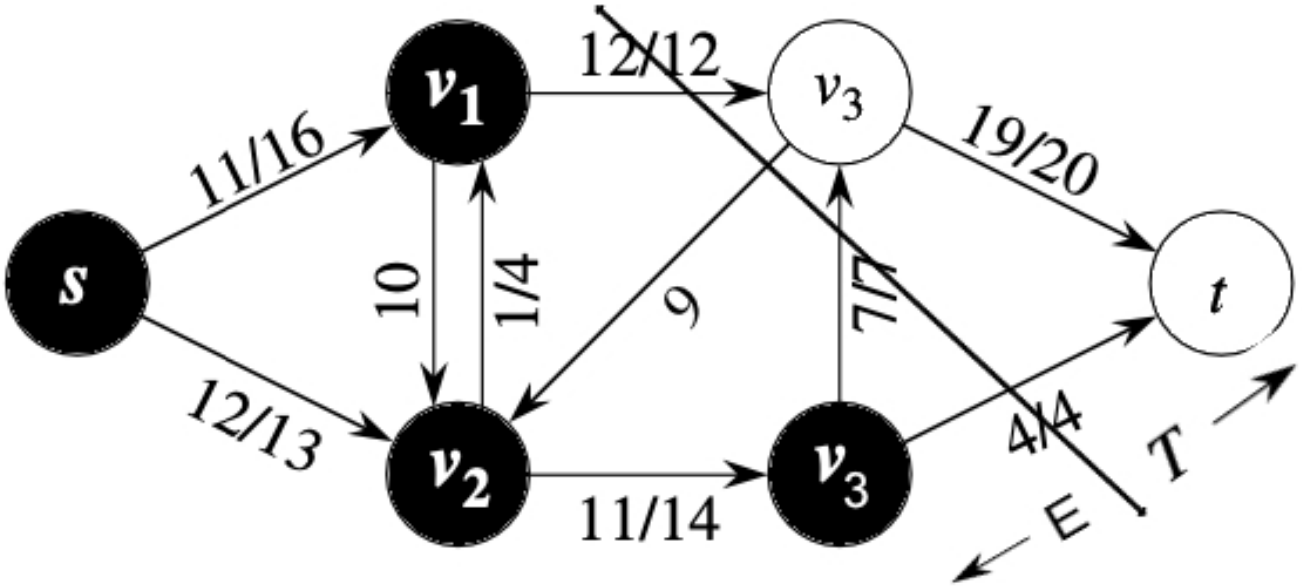


Figure 5 – flot maximal et coupe minimale [2]

- Arrêtes saturées

Identification d'une coupe minimale

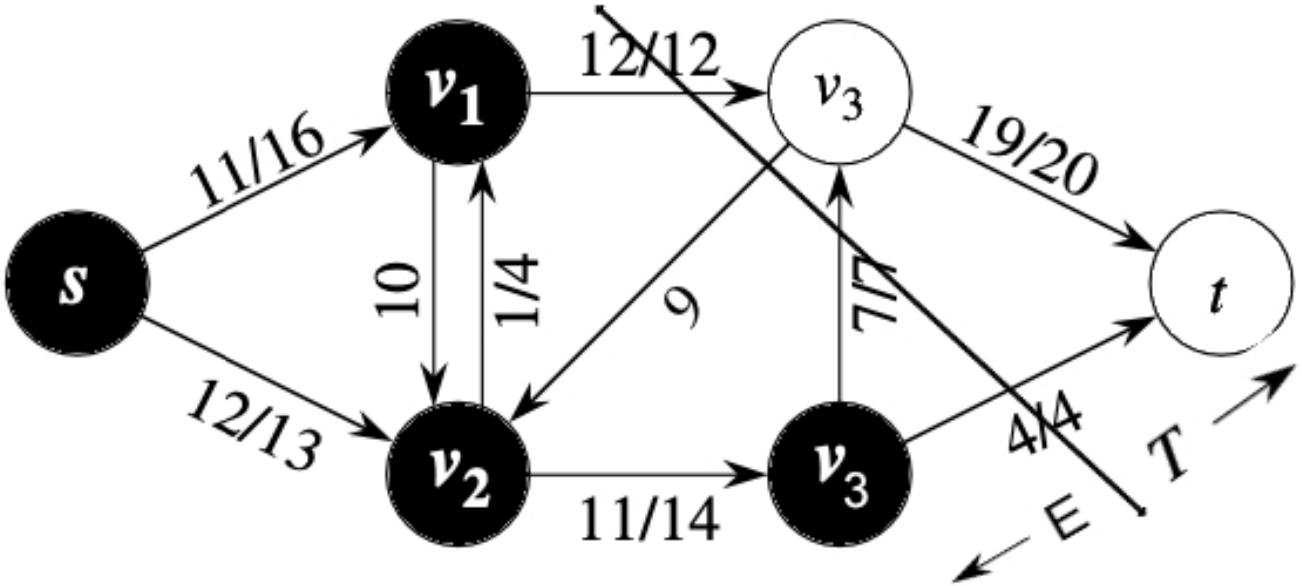


Figure 5 – flot maximal et coupe minimale [2]

- Arrêtes saturées
- E = composante connexe de s dans G_f
- T = composante connexe de t dans G_f

Méthode de Ford-Fulkerson

Algorithme 1 : Ford-Fulkerson(G,s,t)

pour $(u,v) \in A$ **faire**
  $f(u,v) \leftarrow 0$
fin
tant que *il existe un chemin améliorant* p **faire**
  pour (u,v) *de* p **faire**
    $f(u,v) \leftarrow f(u,v) + c_f(p)$;
    $f(v,u) \leftarrow -f(u,v)$;
  fin
fin

Méthode de Ford-Fulkerson

Algorithme 1 : Ford-Fulkerson(G,s,t)

pour $(u,v) \in A$ **faire**
  $f(u,v) \leftarrow 0$
fin
tant que *il existe un chemin améliorant* p **faire**
  pour (u,v) *de* p **faire**
    $f(u,v) \leftarrow f(u,v) + c_f(p)$;
    $f(v,u) \leftarrow -f(u,v)$;
  fin
fin

- Recherche du chemin ameliorant $\longrightarrow$ plusieurs variantes

Algorithme d'Edmonds-Karp

- Le chemin améliorant p est un plus court chemin de s vers t dans le réseau résiduel, où chaque arête possède une pondération unitaire
- Cette recherche se fait à l'aide d'un parcours en largeur

Algorithme d'Edmonds-Karp

- Le chemin améliorant p est un plus court chemin de s vers t dans le réseau résiduel, où chaque arête possède une pondération unitaire
- Cette recherche se fait à l'aide d'un parcours en largeur

Complexité de l'algorithme d'Edmonds-Karp

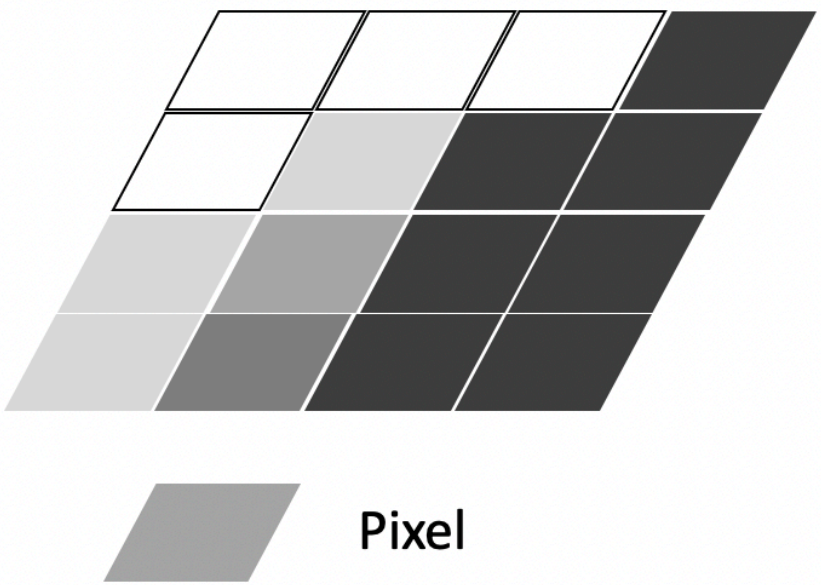
Le temps d'exécution total de l'algorithme d'Edmonds-Karp est $\mathcal{O}(|S||A|^2)$

- Démonstration en annexe
- Code en annexe

Méthode Pousser-réétiqueter

- Une autre approche (algorithme de Goldberg)
- $\mathcal{O}(|S|^2|A|)$ ou $\mathcal{O}(|S|^3)$
- Pas de gain vis-à-vis de notre objectif
- Code en annexe

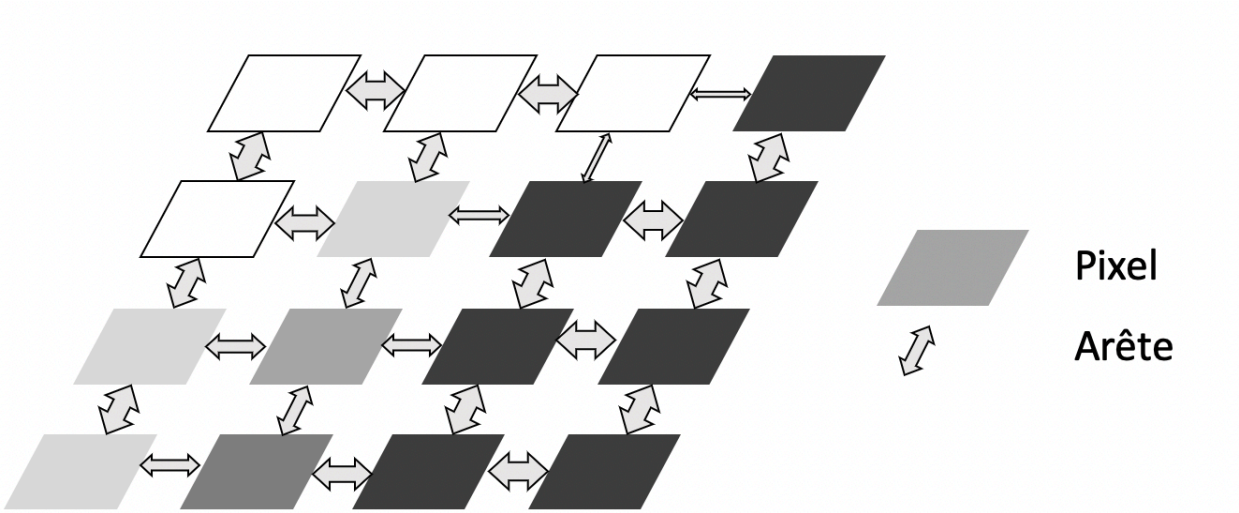
Comment transformer une image en graphe ?



- Donnée : image de taille $n \times n$
- But : Réseau de transport $G = (S, A, c)$

Comment transformer une image en un graphe ?

- Pixel dans l'image $\longrightarrow$ Sommet dans le graphe.
- Des capacités traduisant la similarité entre deux pixels.



Similarité entre deux pixels

- Pour un pixel i, on note $(f_{ik})_k$ ses caractéristiques.

Définition 4.1

L'affinité entre deux pixels i et j est :

$$A(i,j) = e^{\frac{-1}{2\sigma^2}\sum_k (f_{ik}-f_{jk})^2}$$

où σ est un paramètre de réglage.

Similarité entre deux pixels

- Pour un pixel i, on note $(f_{ik})_k$ ses caractéristiques.

Définition 4.1

L'affinité entre deux pixels i et j est :

$$A(i,j) = e^{\frac{-1}{2\sigma^2}\sum_k (f_{ik}-f_{jk})^2}$$

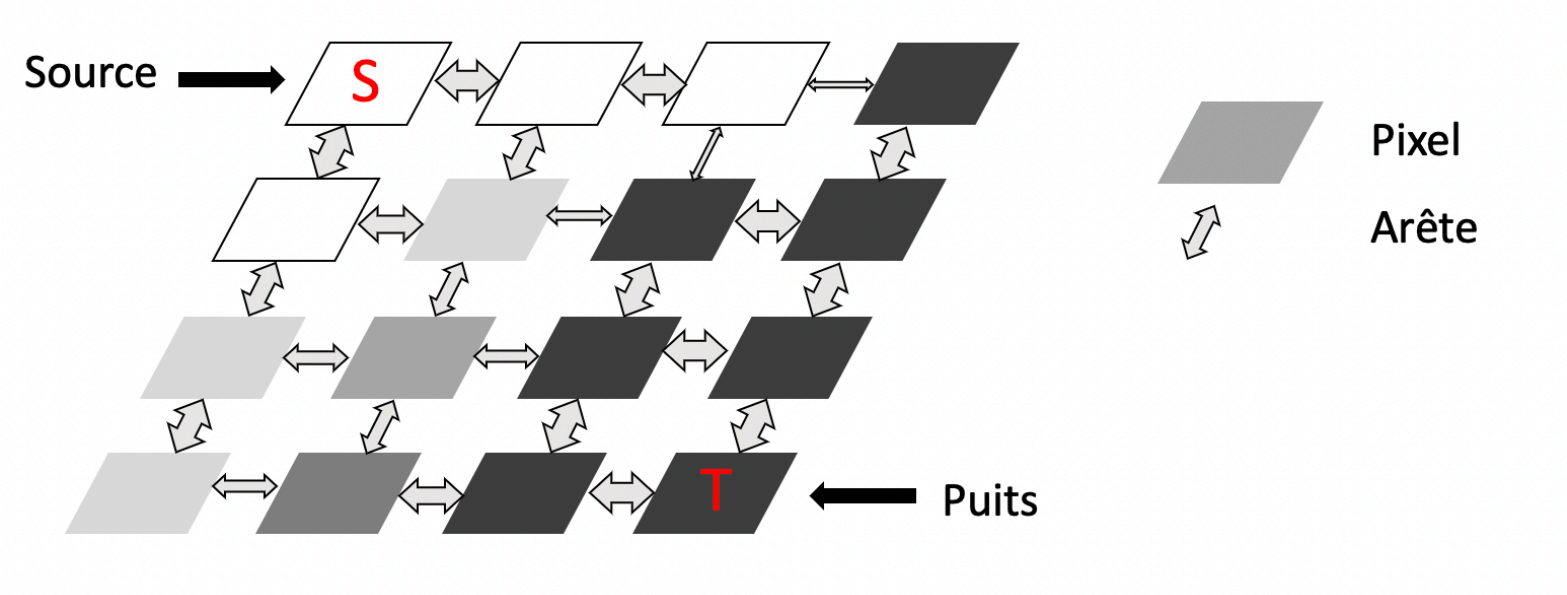
où σ est un paramètre de réglage.

- Pour i et j deux pixels voisins, deux arêtes anti-parallèles (u,v) et (v,u) sont ajoutées avec des capacités :

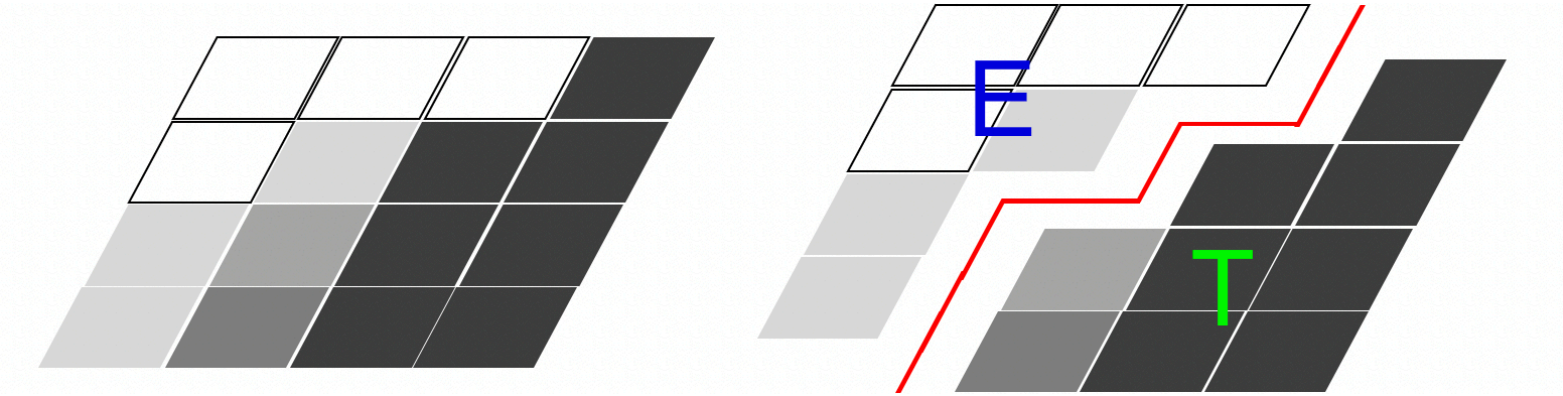
$$c(u,v) = c(v,u) = \lceil 100 \times A(i,j) \rceil$$

Source et puits ?

- Pointer un pixel de l'arrière-plan $\longrightarrow$ Source s
- Pointer un pixel de l'objet $\longrightarrow$ Puits t



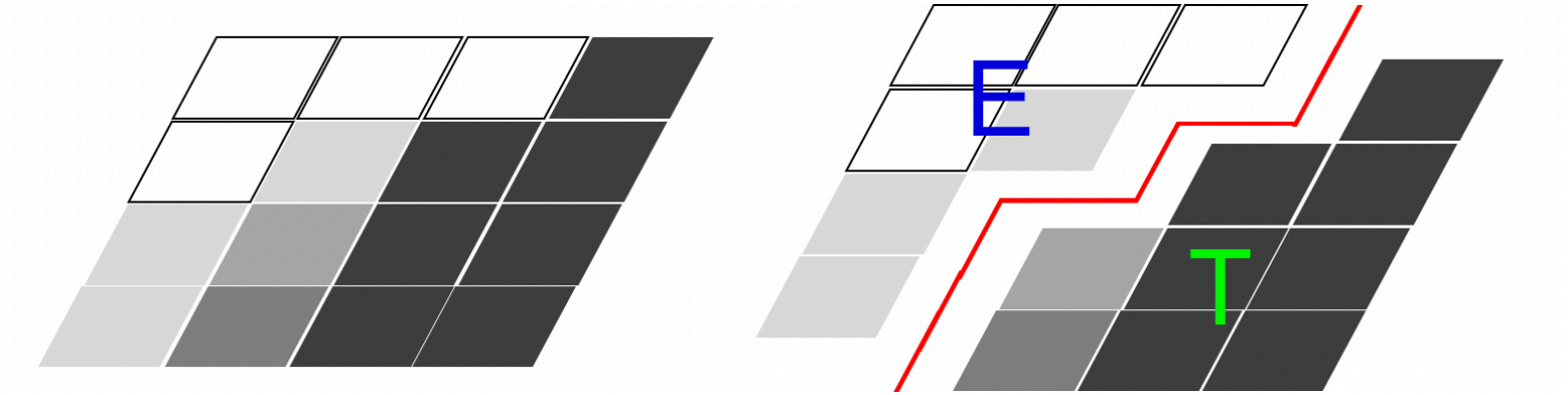
Coupe minimale



- Minimiser : $\sum_{i \in E, j \in T} A(i,j)$
- Maximiser :

$$\sum_{i,j \in E} A(i,j) + \sum_{i,j \in T} A(i,j)$$

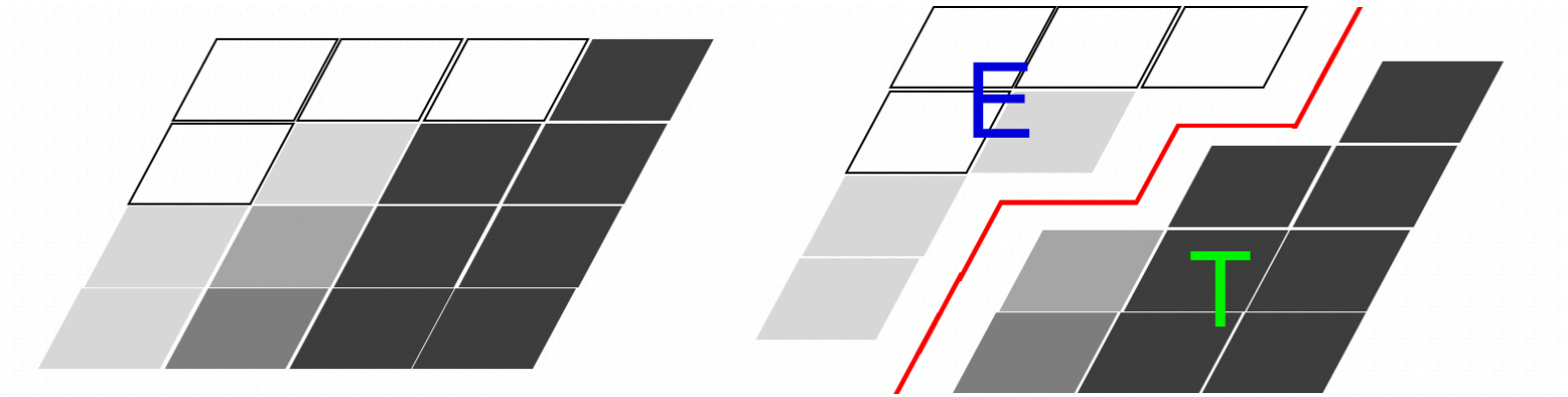
Coupe minimale



- Minimiser : $\sum_{i \in E, j \in T} A(i,j)$
- Maximiser :

$$\sum_{i,j \in E} A(i,j) + \sum_{i,j \in T} A(i,j) = \underbrace{\sum_{i,j \in S} A(i,j)}_{\text{Terme constant}} - 2 \times \sum_{i \in E, j \in T} A(i,j)$$

Coupe minimale



- Minimiser : $\sum_{i \in E, j \in T} A(i,j)$
- Maximiser :

$$\sum_{i,j \in E} A(i,j) + \sum_{i,j \in T} A(i,j) = \underbrace{\sum_{i,j \in S} A(i,j)}_{\text{Terme constant}} - 2 \times \sum_{i \in E, j \in T} A(i,j)$$

- Segmentation de l'image $\iff$ Recherche d'une coupe minimale

Problème

- Une tendance à couper des petits segments isolés

Problème

- Une tendance à couper des petits segments isolés

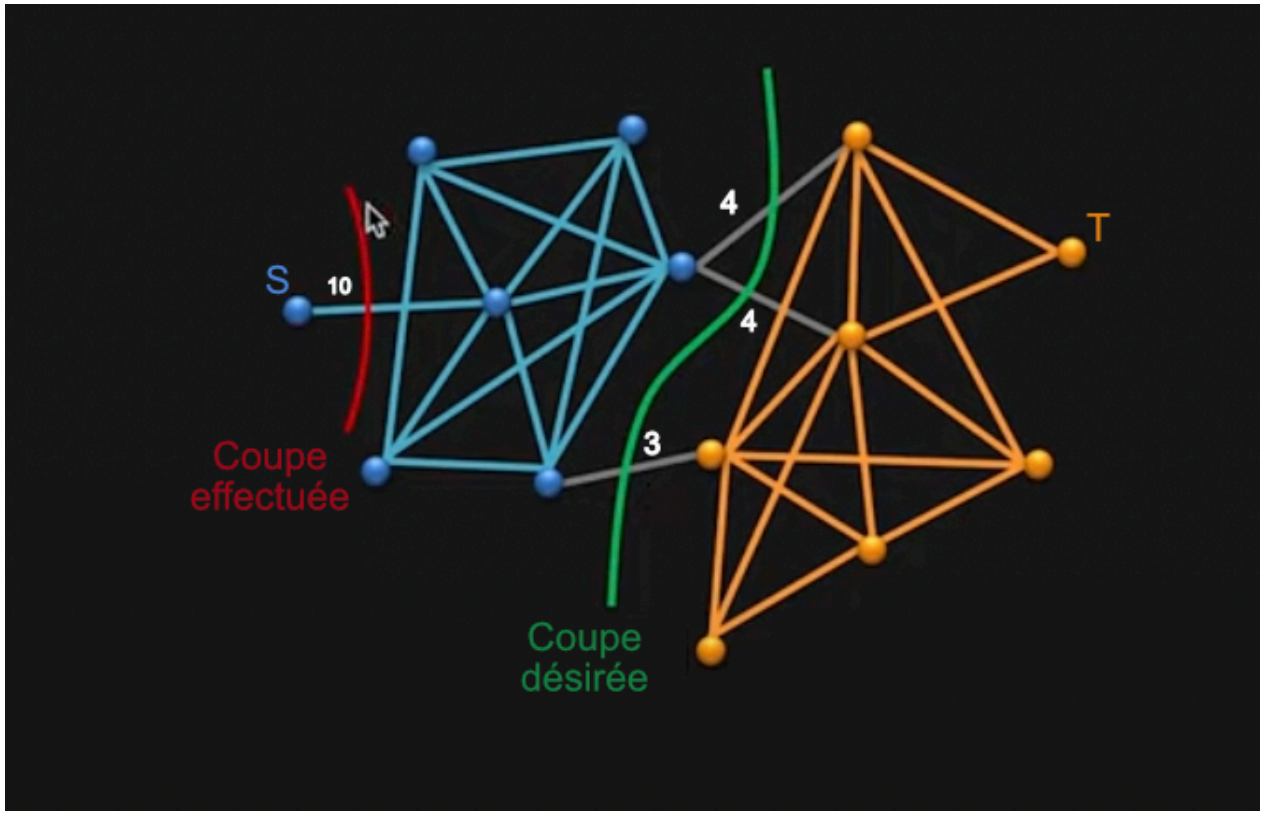
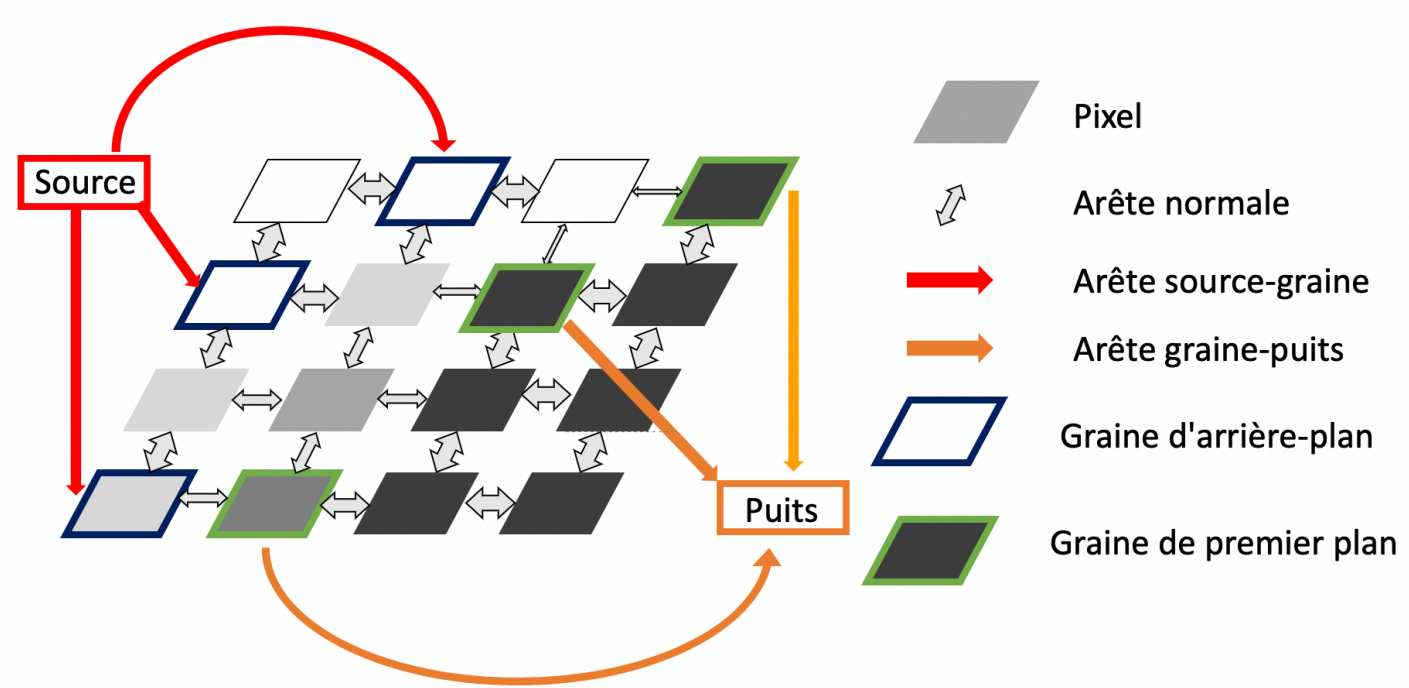


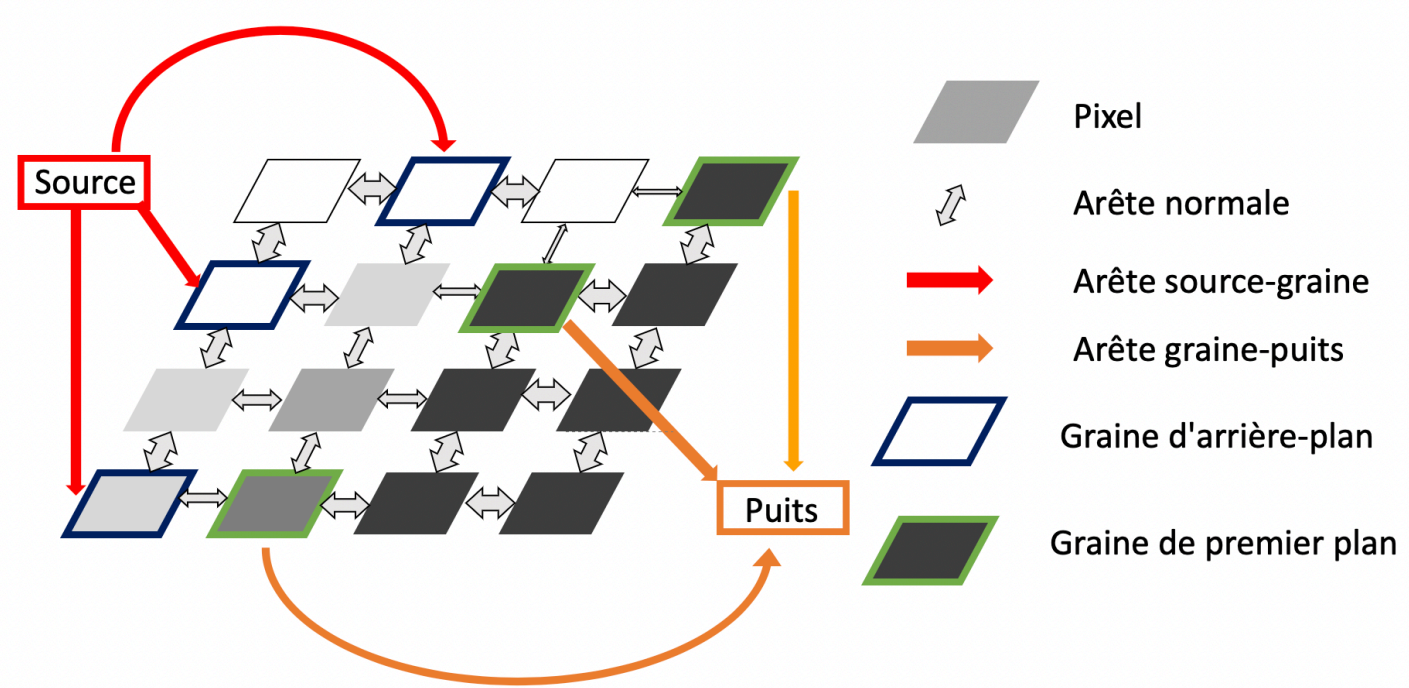
Figure 6 – [8]

Solution



- Marquer manuellement des pixels $\longrightarrow$ Les graines

Solution



- Marquer manuellement des pixels $\longrightarrow$ Les graines
- Source s fictive représentant l'arrière-plan
- Puits t fictif représentant l'objet

- $\mathcal{B}$: Ensemble des graines de l'arrière-plan
- $\mathcal{O}$: Ensemble des graines de l'objet

- $\mathcal{B}$: Ensemble des graines de l'arrière-plan
- $\mathcal{O}$: Ensemble des graines de l'objet
- $\mathcal{K} = \max_{u \in S} \sum_{v:(u,v) \in A} c(u, v)$
- $\forall b \in \mathcal{B},\ c(s, b) = 1 + \mathcal{K}$
- $\forall o \in \mathcal{O},\ c(o, t) = 1 + \mathcal{K}$

- $\mathcal{B}$: Ensemble des graines de l'arrière-plan
- $\mathcal{O}$: Ensemble des graines de l'objet
- $\mathcal{K} = \max_{u \in S} \sum_{v:(u,v) \in A} c(u, v)$
- $\forall b \in \mathcal{B},\ c(s, b) = 1 + \mathcal{K}$
- $\forall o \in \mathcal{O},\ c(o, t) = 1 + \mathcal{K}$

Validité de la segmentation

Pour (E,T) une coupe minimale :

$$\forall b \in \mathcal{B}, b \in E$$

$$\forall o \in \mathcal{O}, o \in T$$

- Démonstration en annexe

Complexité

- $|S| = n^2$
- $|A| = 4\,n^2$
- Edmonds-Karp $\mathcal{O}(|S||A|^2)$
- Pousser-réétiqueter $\mathcal{O}(|S|^2|A|)$ ou $\mathcal{O}(|S|^3)$

Complexité

- $|S| = n^2$
- $|A| = 4\,n^2$
- Edmonds-Karp $\mathcal{O}(|S||A|^2)$
- Pousser-réétiqueter $\mathcal{O}(|S|^2|A|)$ ou $\mathcal{O}(|S|^3)$

Complexité de la segmentation

La complexité de notre algorithme de segmentation d'images est $\mathcal{O}(n^6)$

Implémentation

- Images carrées $n \times n$ en niveaux de gris
- Critère : luminosité I

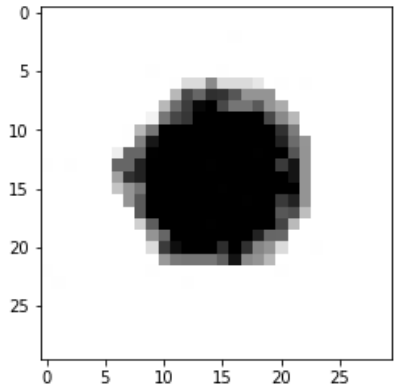
Implémentation

- Images carrées $n \times n$ en niveaux de gris
- Critère : luminosité I
- Sélection des graines
- Construction du graphe

Implémentation

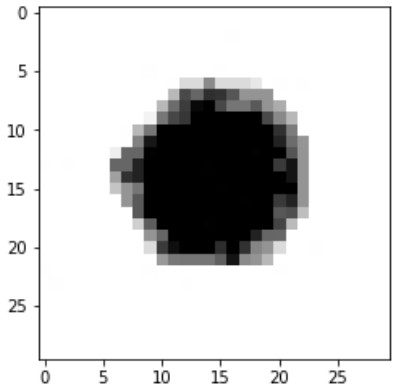
- Images carrées $n \times n$ en niveaux de gris
- Critère : luminosité I
- Sélection des graines
- Construction du graphe
- Recherche d'un flot maximal
- Identification des coupes via un parcours en largeur
- Code en annexe

Choix de σ

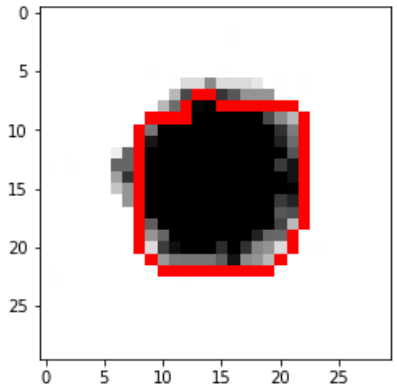


(a) Image à segmenter

Choix de σ

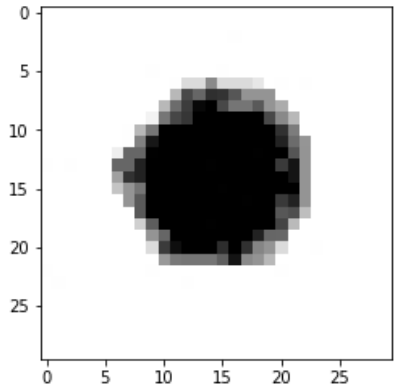


(a) Image à segmenter

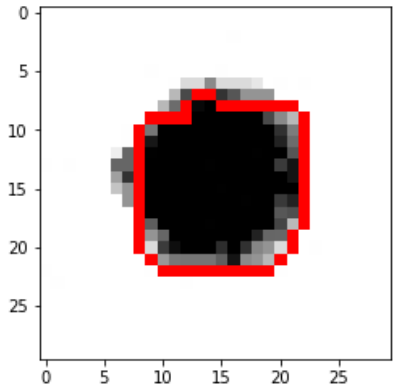


(b) $\sigma = 10$

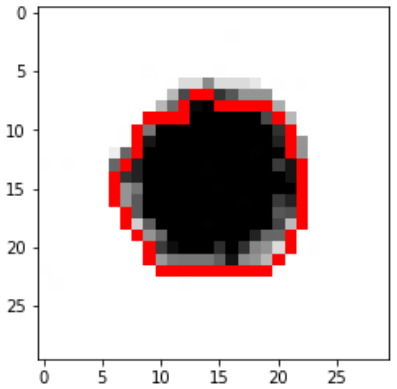
Choix de σ



(a) Image à segmenter

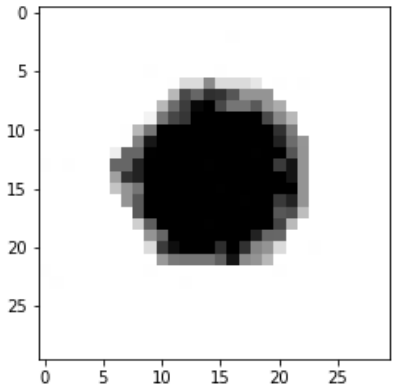


(b) $\sigma = 10$

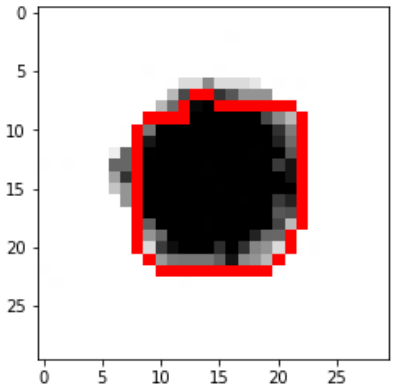


(c) $\sigma = 20$

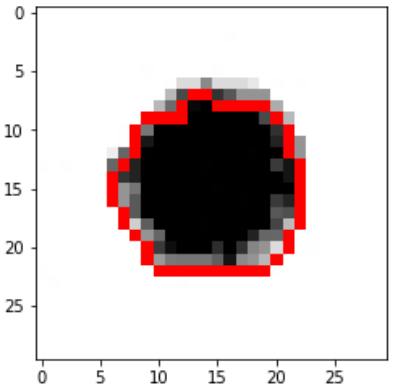
Choix de σ



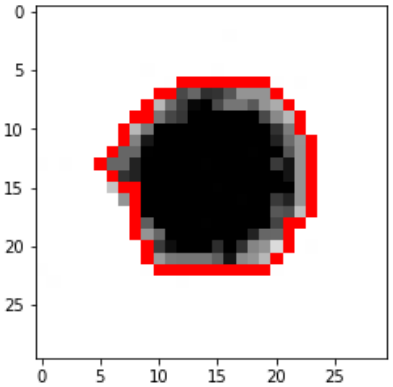
(a) Image à segmenter



(b) $\sigma = 10$

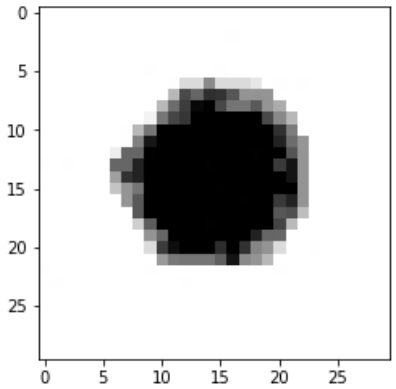


(c) $\sigma = 20$

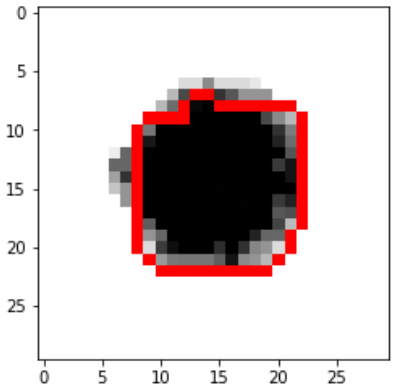


(d) $\sigma = 30/40$

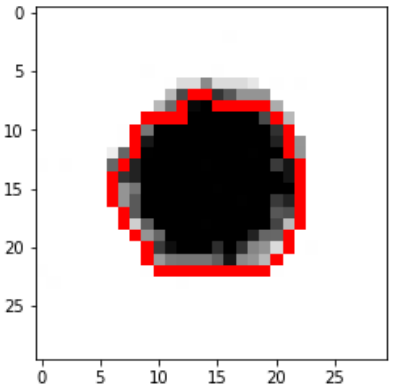
Choix de σ



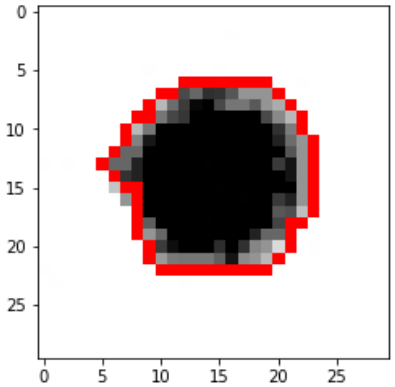
(a) Image à segmenter



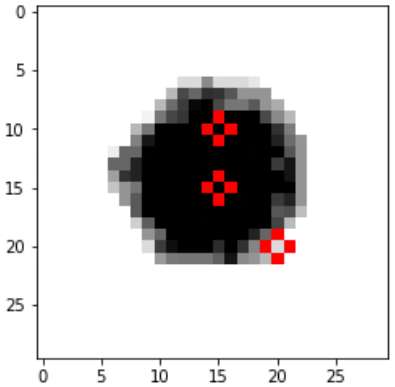
(b) $\sigma = 10$



(c) $\sigma = 20$



(d) $\sigma = 30/40$



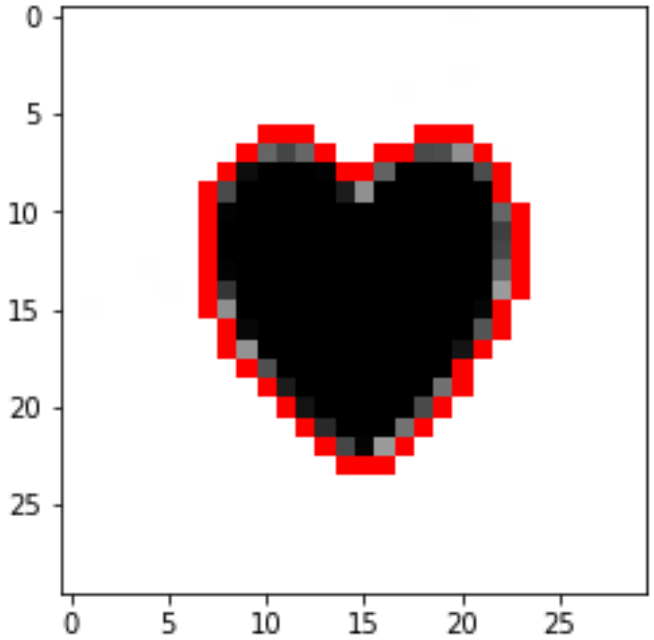
(e) $\sigma = 50$

Temps d'exécution

(a) $n = 30$

Temps d'exécution

(a) $n = 30$



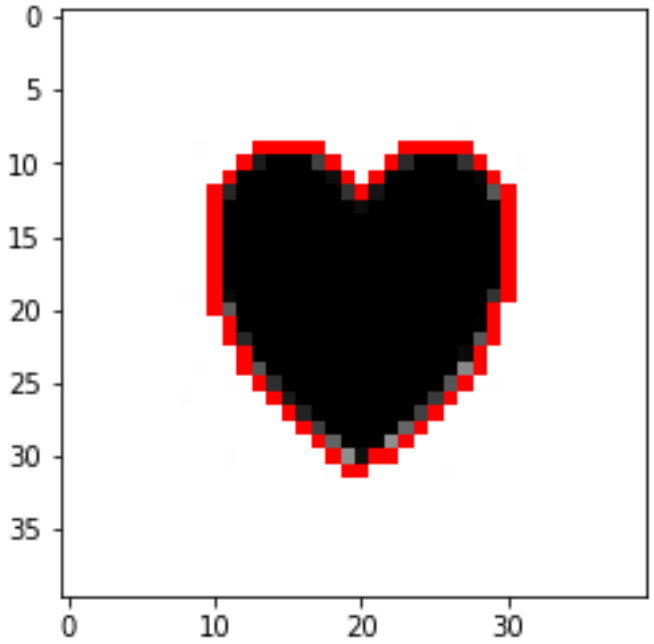
(b) $t = 9.22s$

Temps d'exécution

(a) $n = 40$

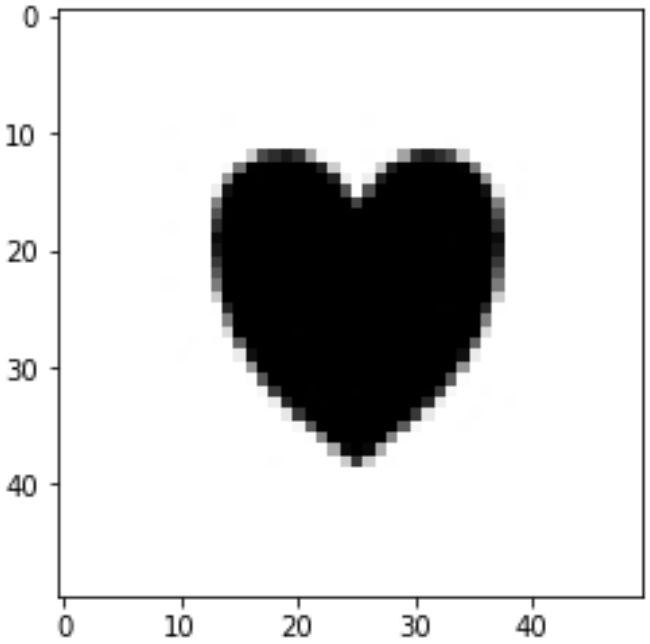
Temps d'exécution

(a) $n = 40$



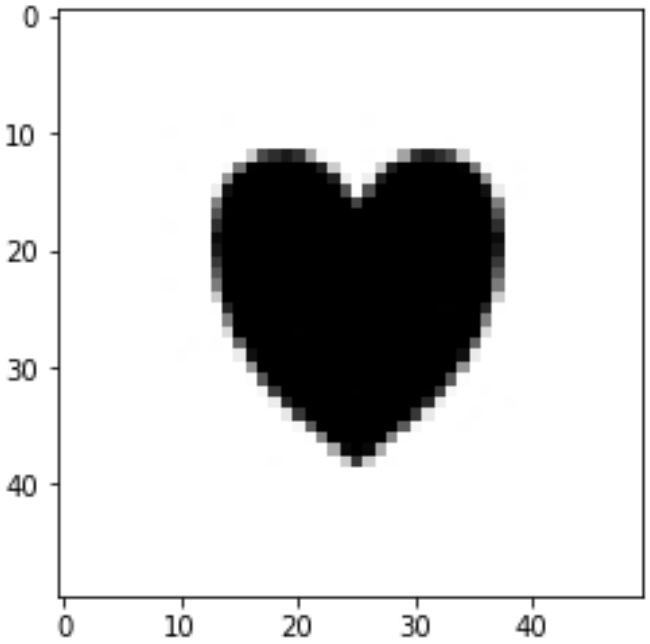
(b) $t = 36.26s$

Temps d'exécution

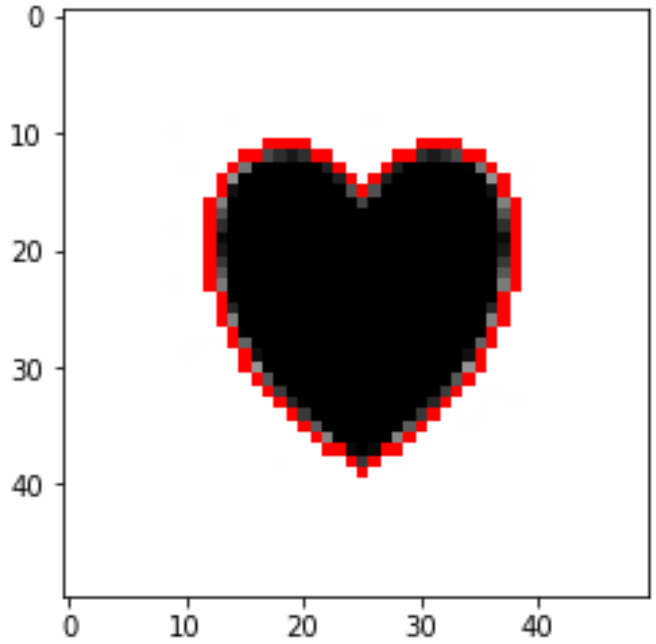


(a) $n = 50$

Temps d'exécution

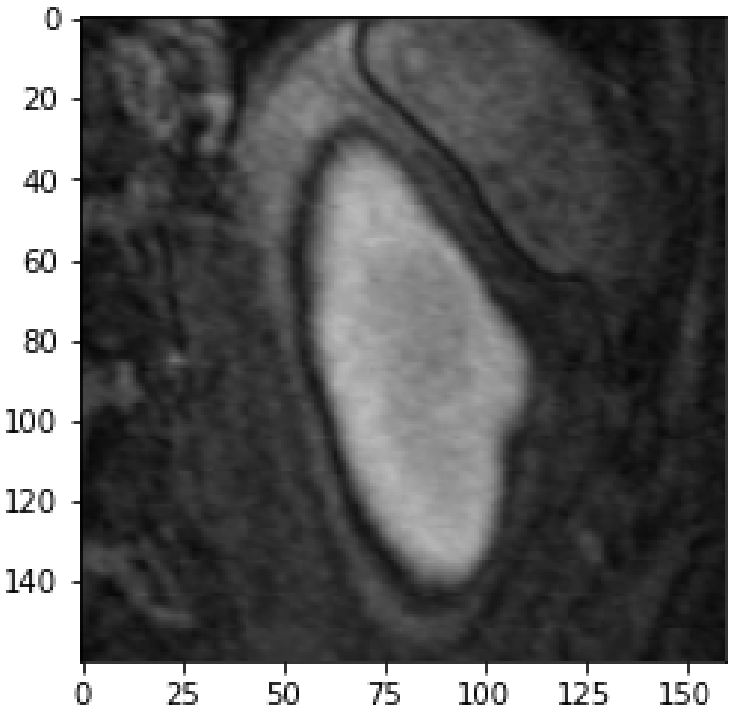


(a) $n = 50$



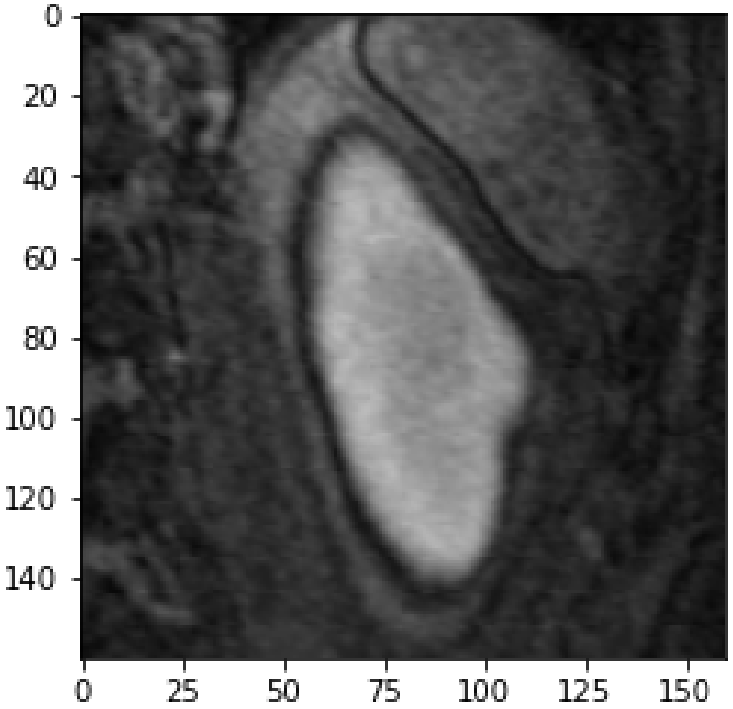
(b) $t = 117.53s$

Image médicale

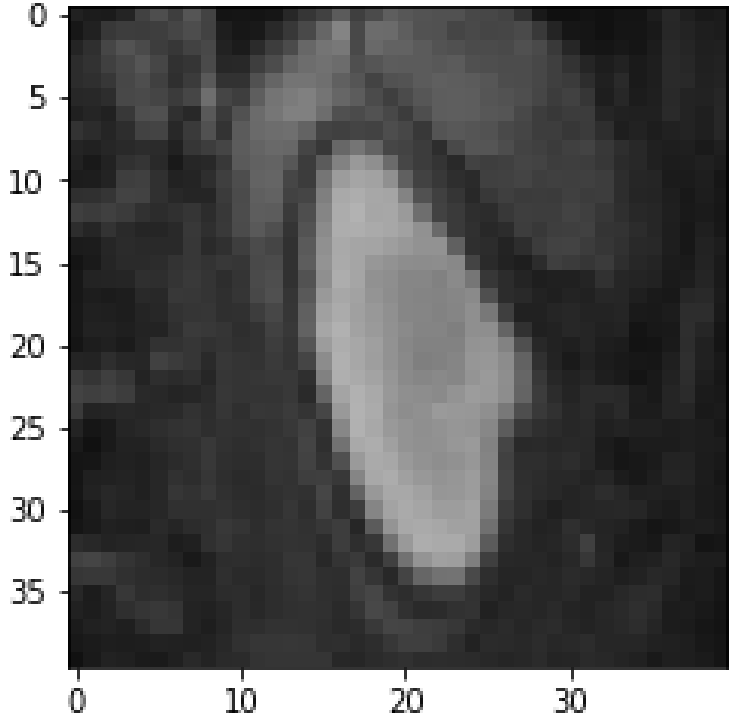


(a) IRM d'un rein $n = 160$

Image médicale



(a) IRM d'un rein $n = 160$



(b) Réduit à n=40

Image médicale

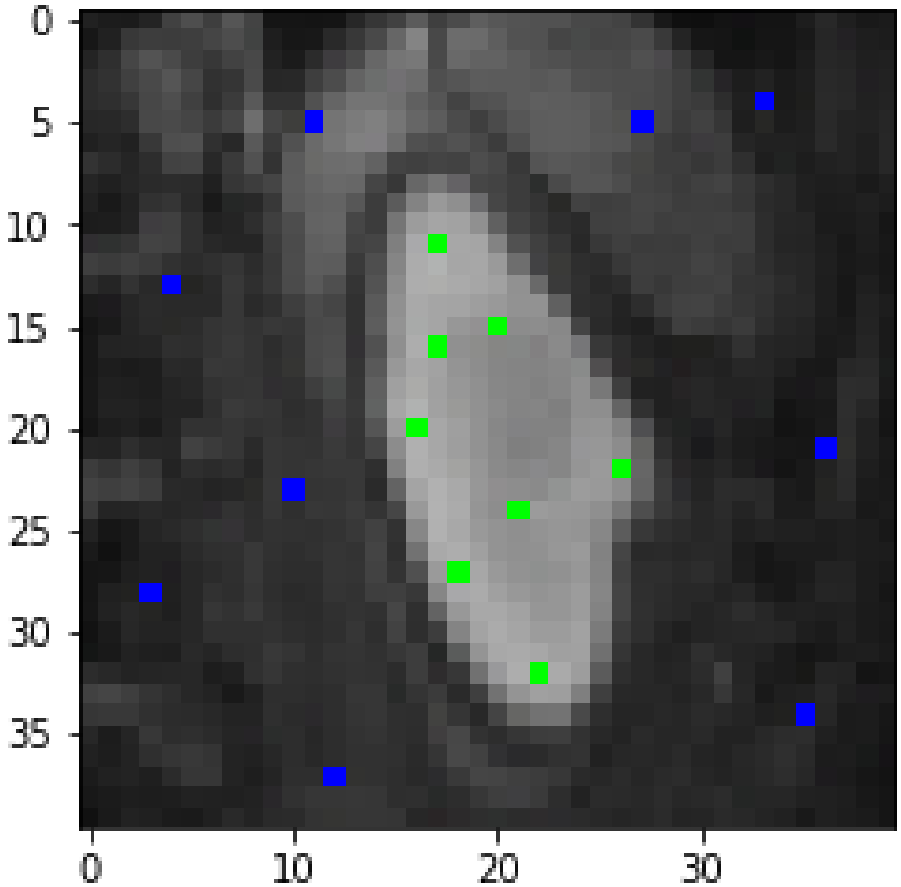
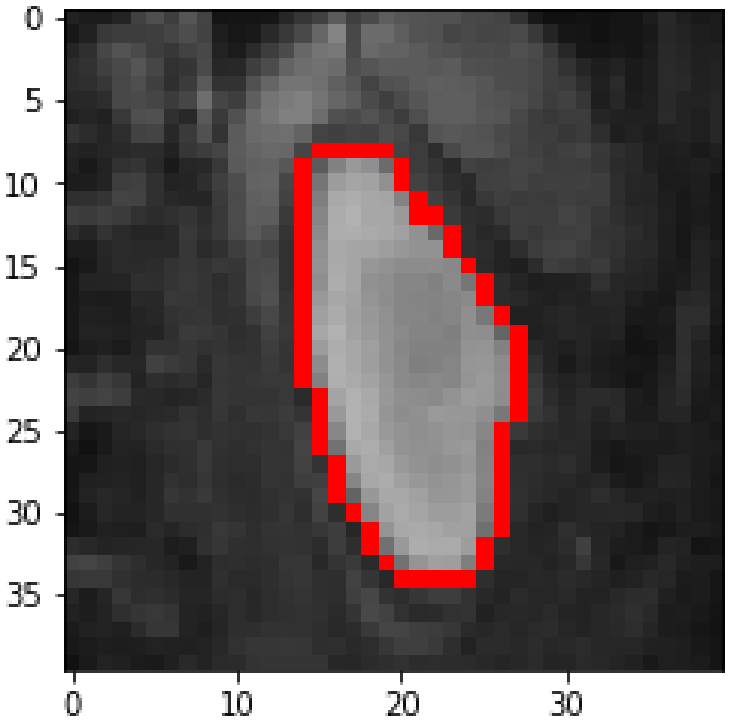
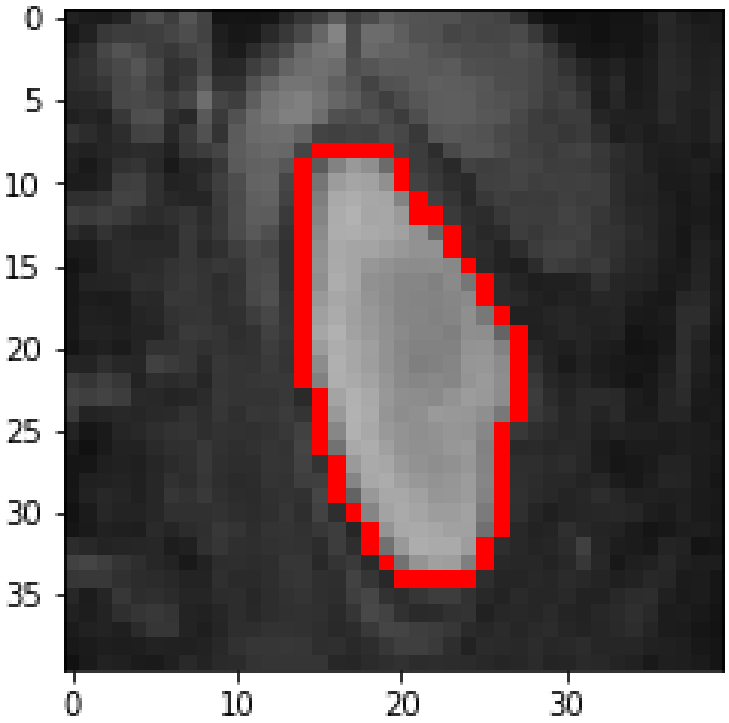


Image médicale

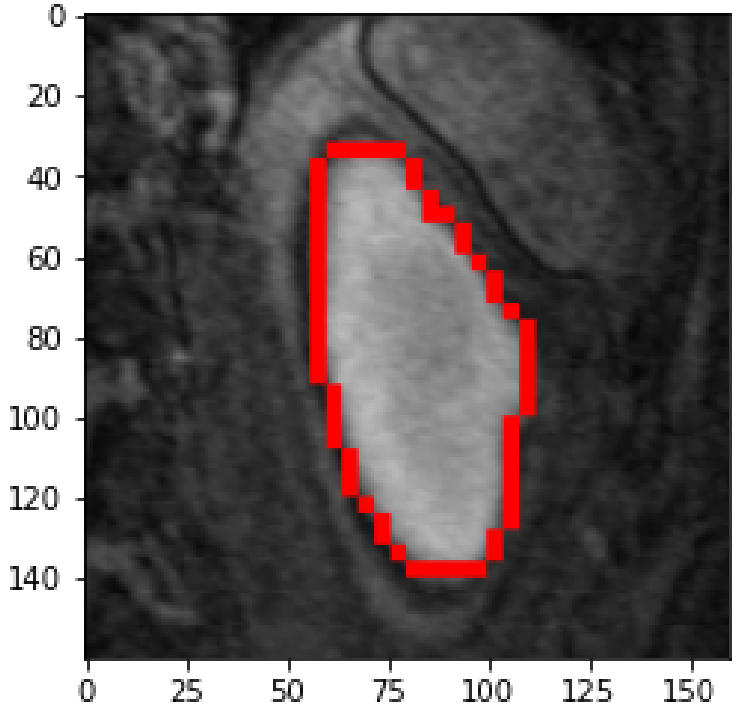


(a) Résultat de segmentation

Image médicale

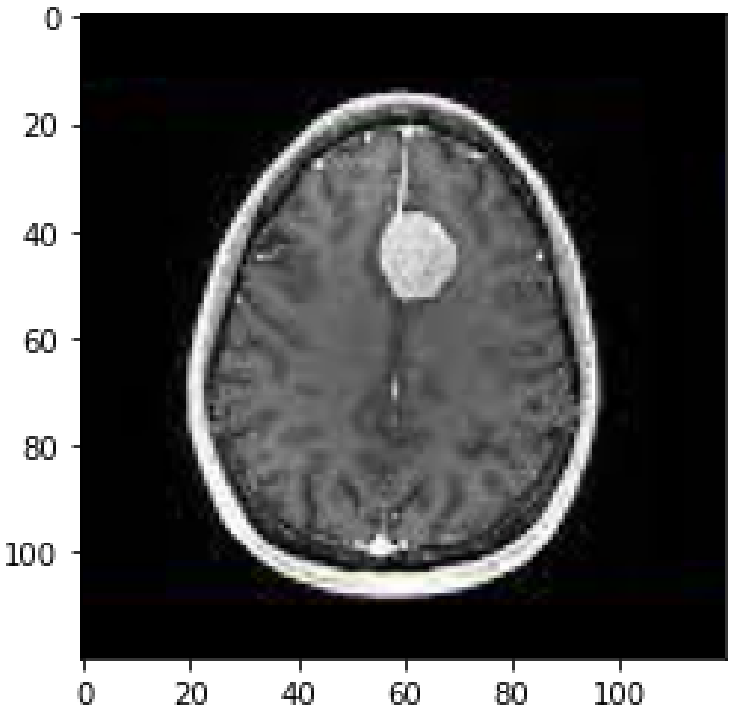


(a) Résultat de segmentation



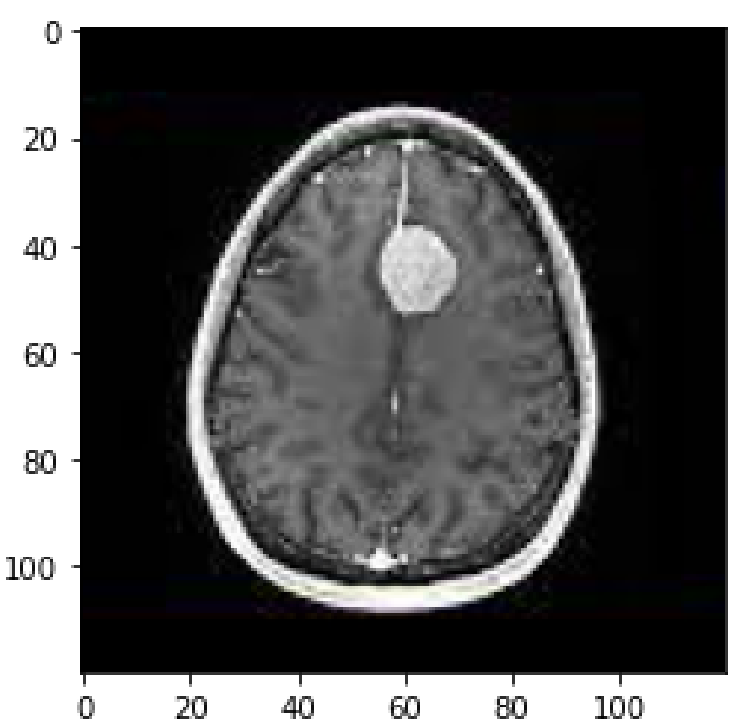
(b) Retour à $n = 160$

Image médicale

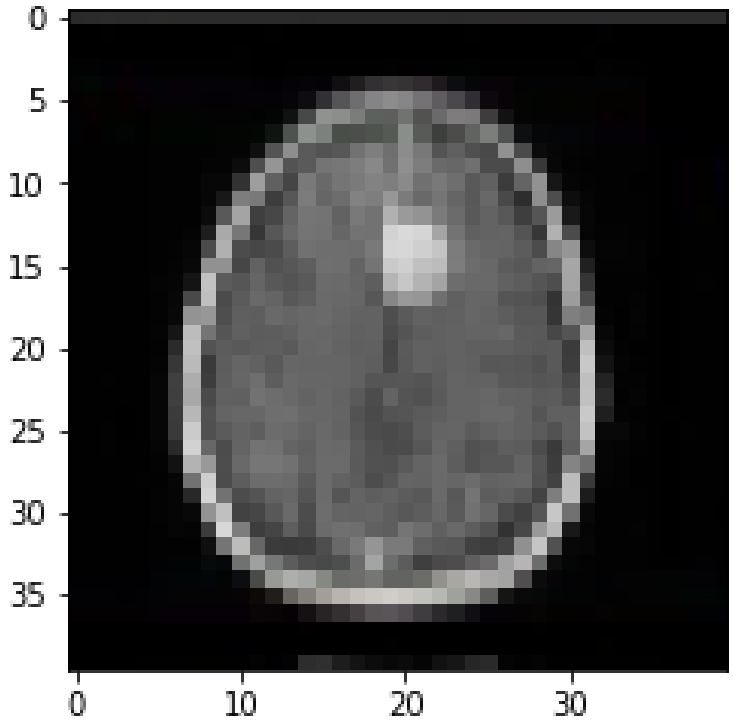


(a) IRM d'un cerveau $n = 120$

Image médicale



(a) IRM d'un cerveau $n = 120$



(b) Réduit à n=40

Image médicale

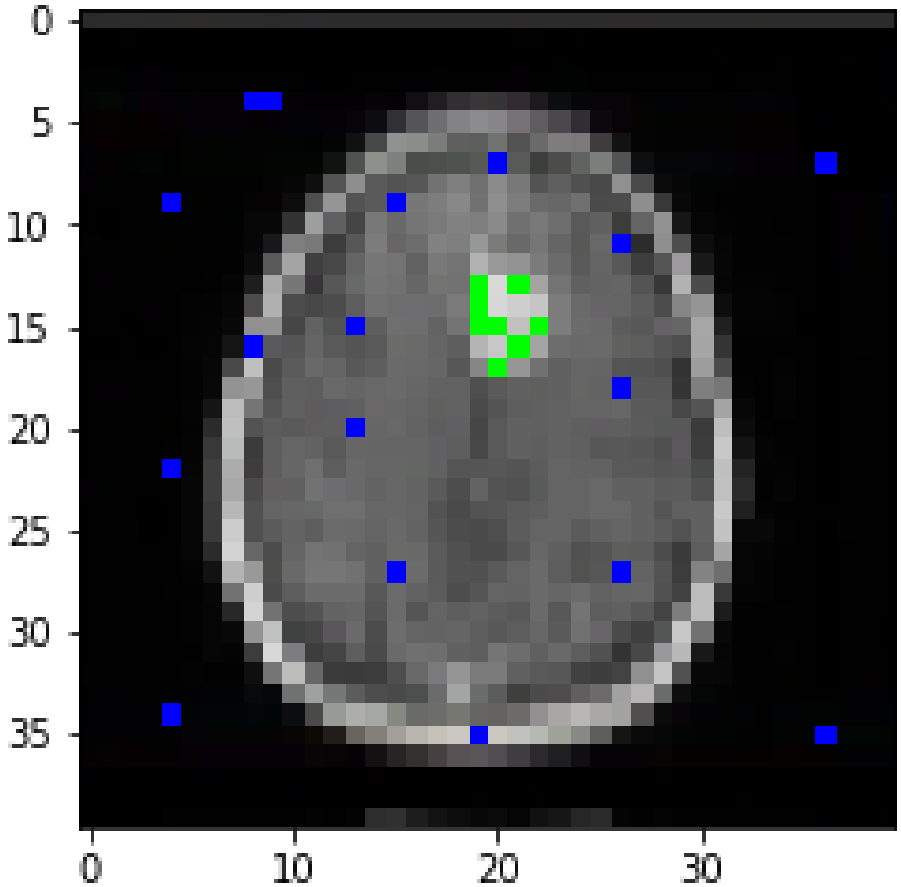
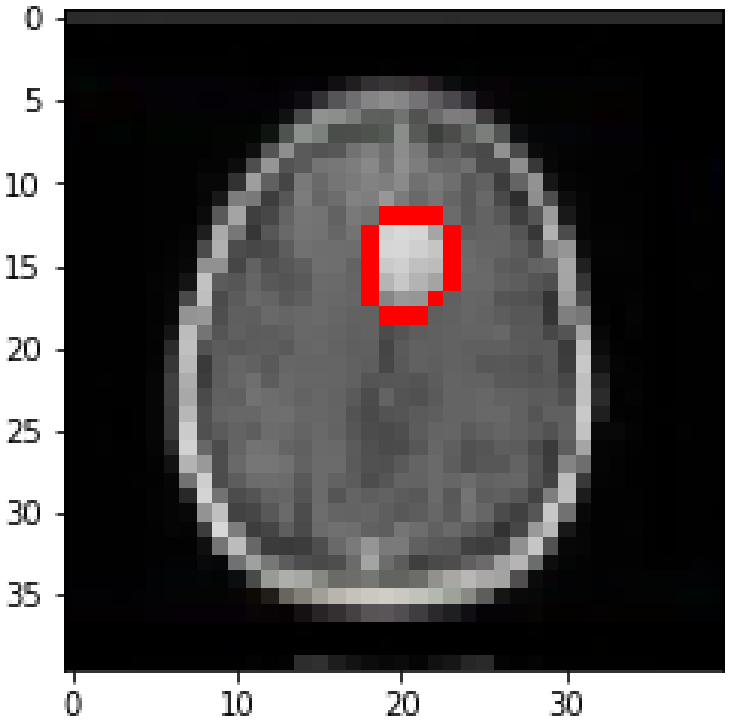
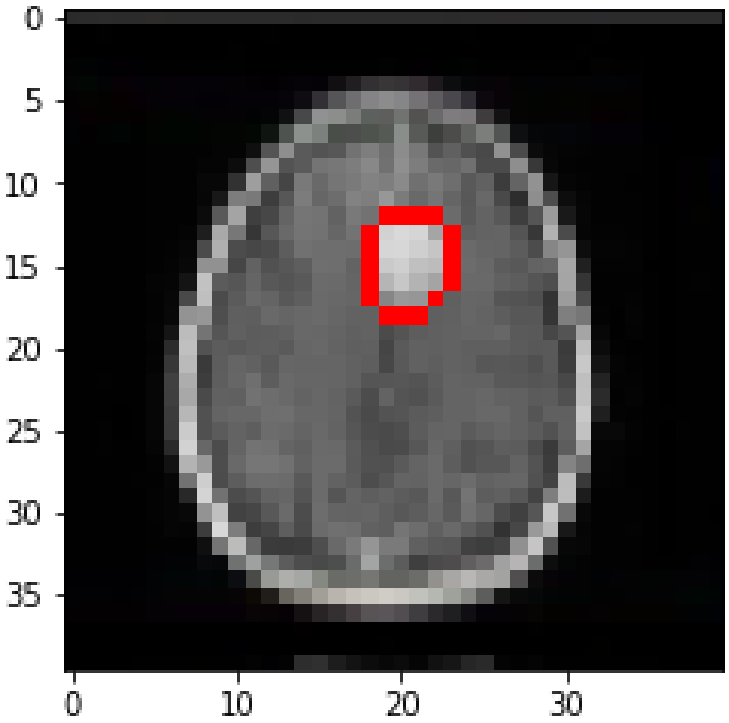


Image médicale

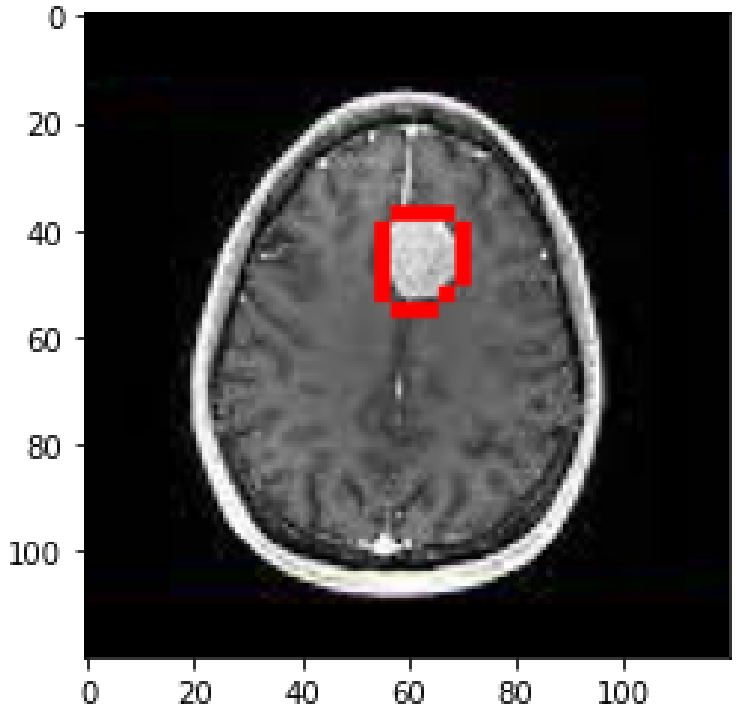


(a) Résultat de segmentation

Image médicale

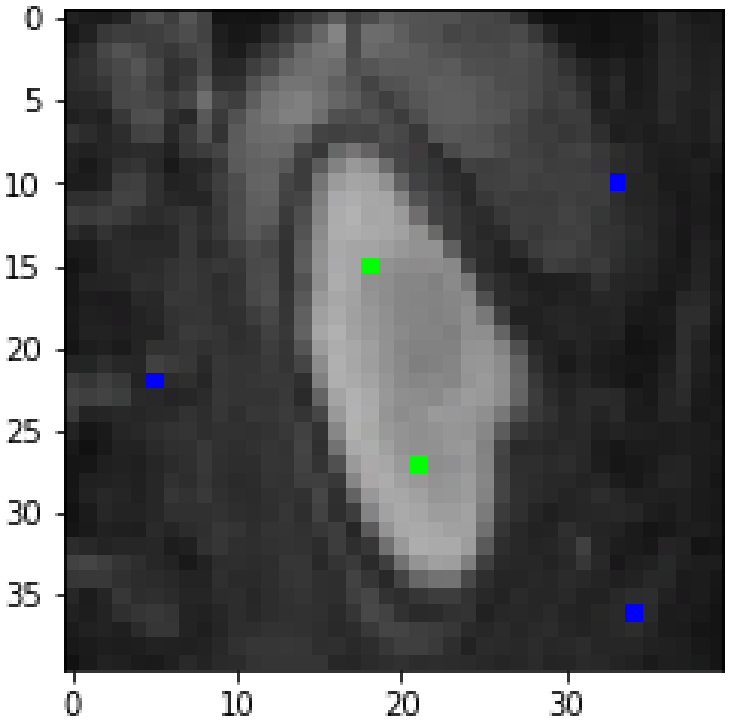


(a) Résultat de segmentation



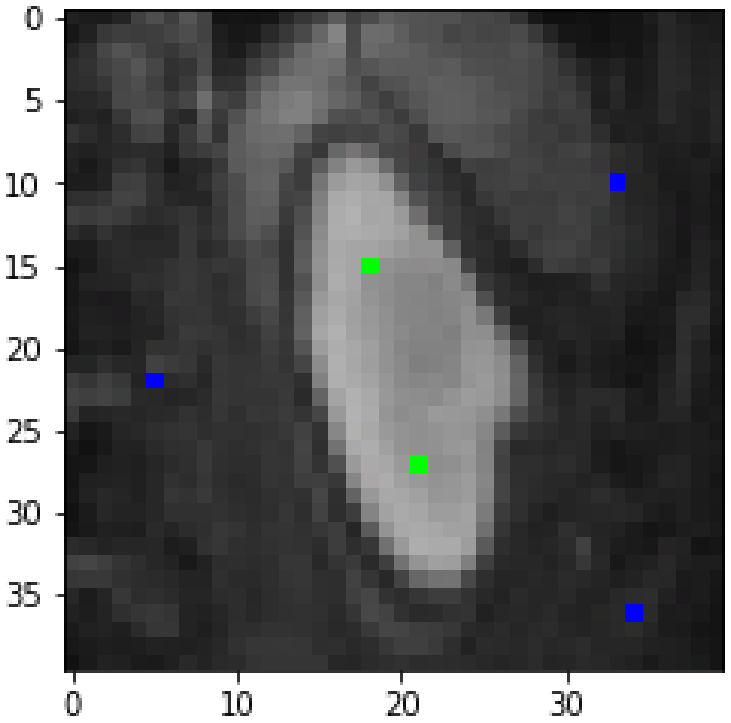
(b) Retour à $n = 120$

Effet du nombre de graines

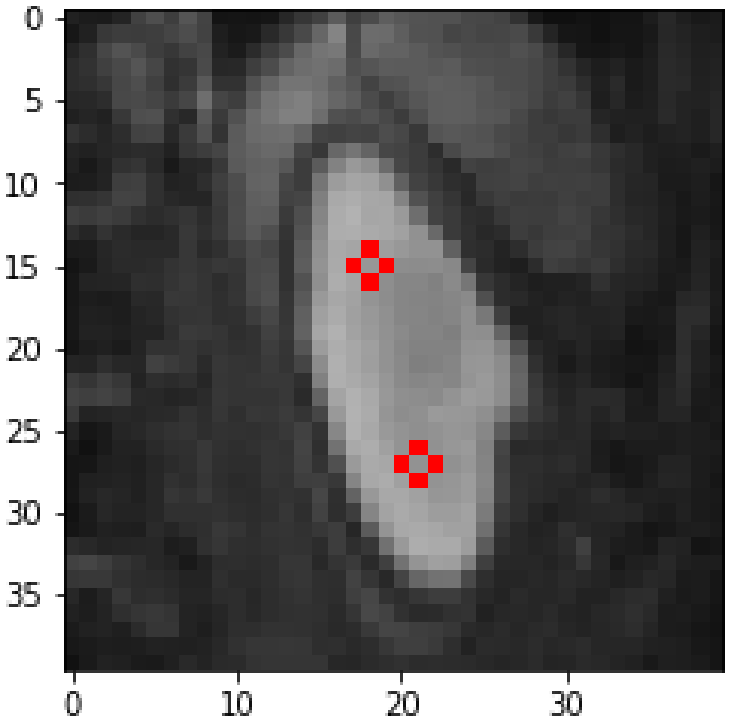


(a)

Effet du nombre de graines

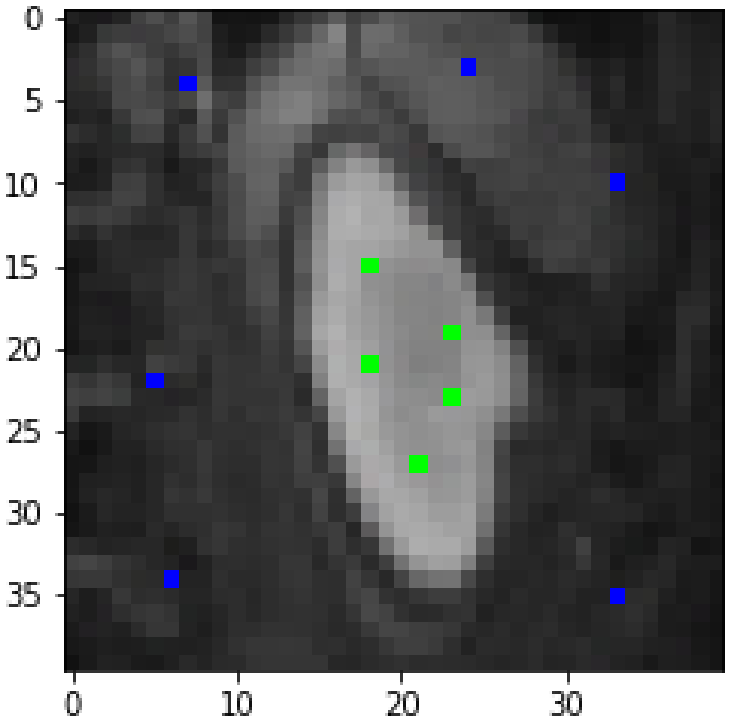


(a)



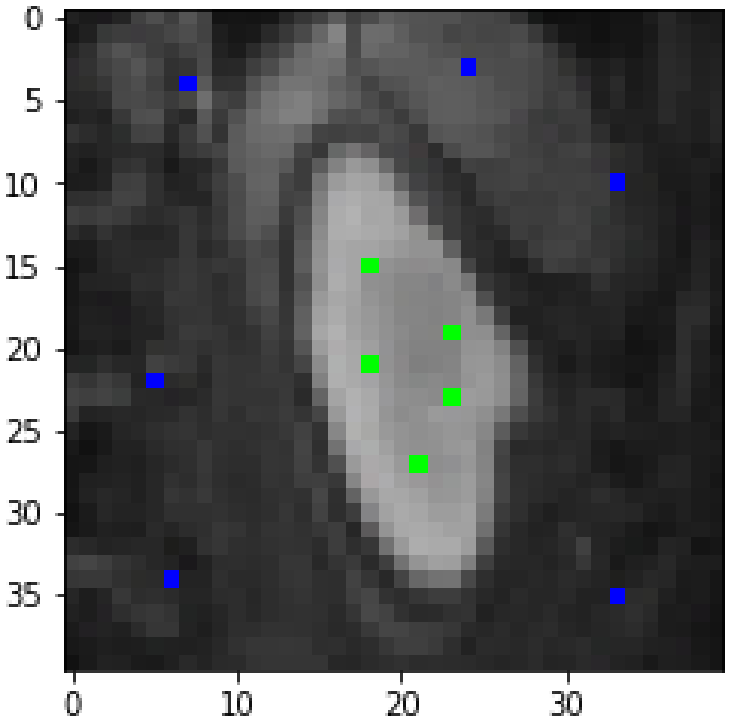
(b)

Effet du nombre de graines

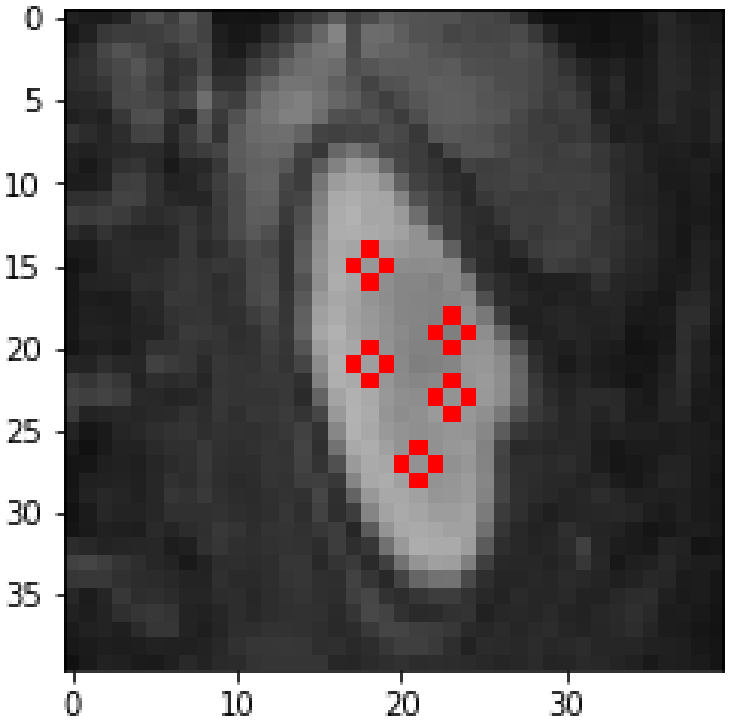


(a)

Effet du nombre de graines

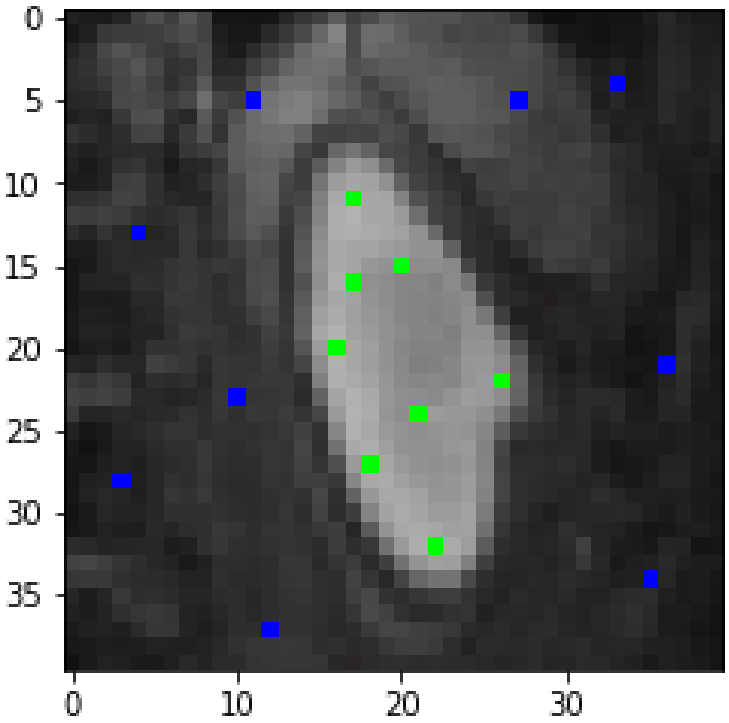


(a)



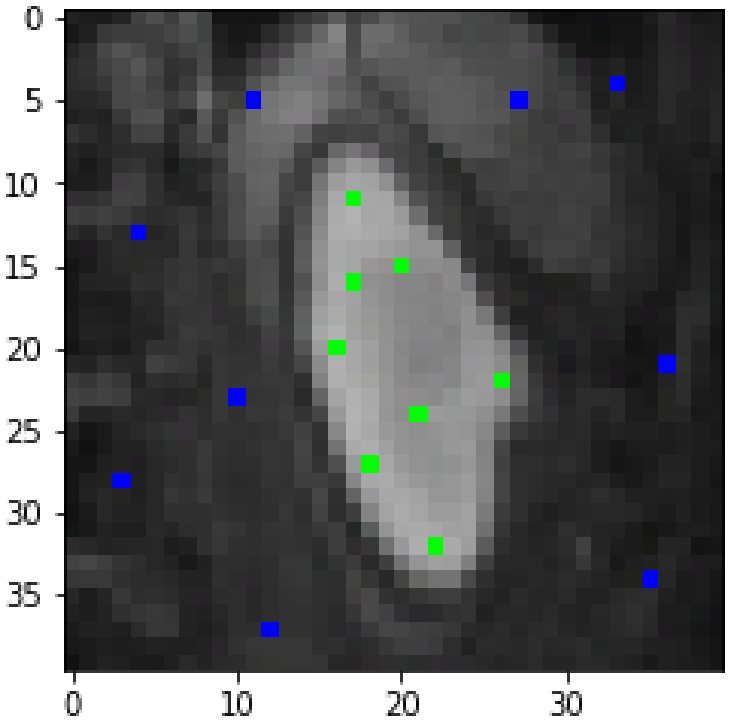
(b)

Effet du nombre de graines

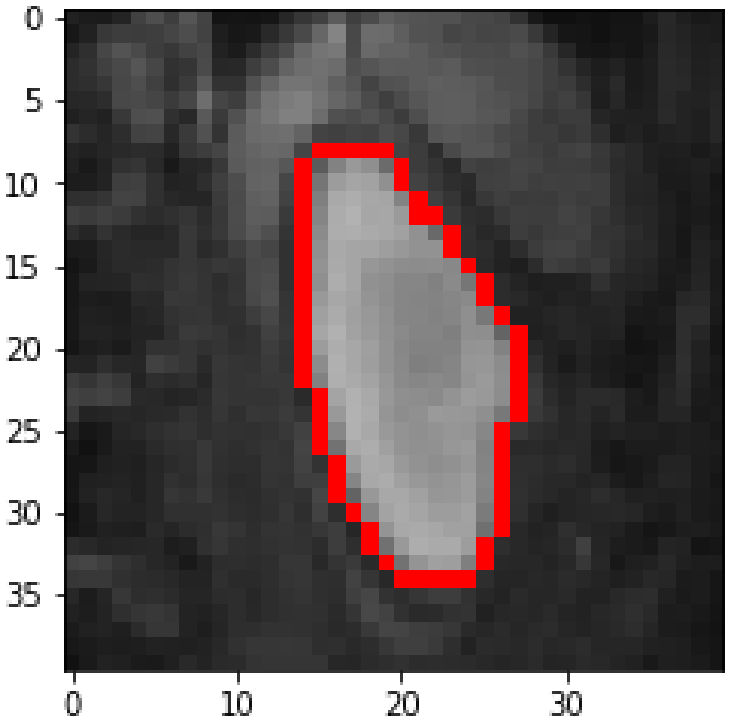


(a)

Effet du nombre de graines

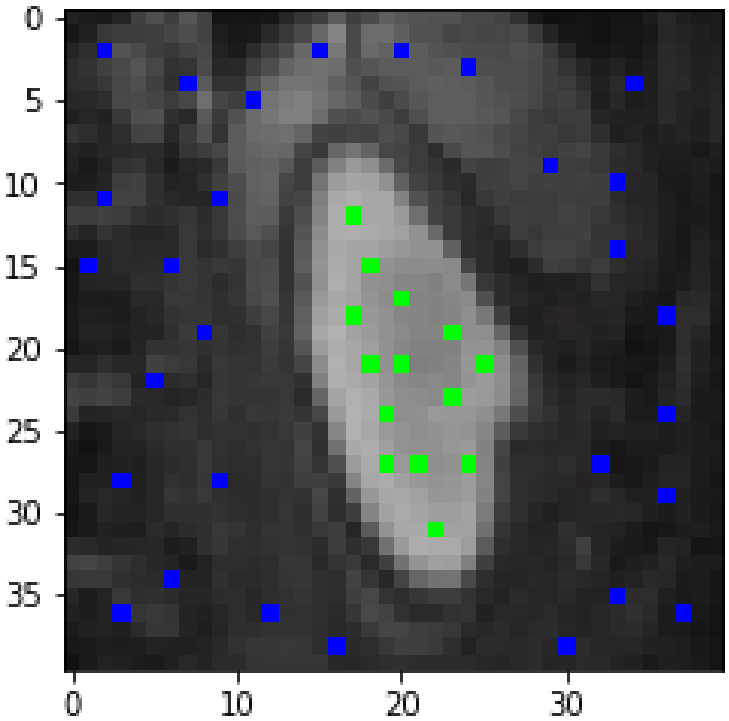


(a)



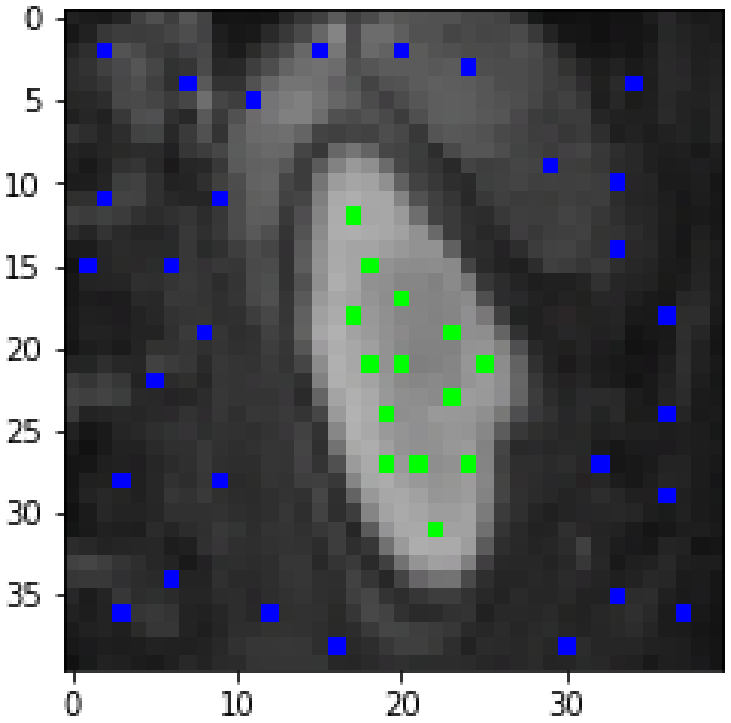
(b)

Effet du nombre de graines

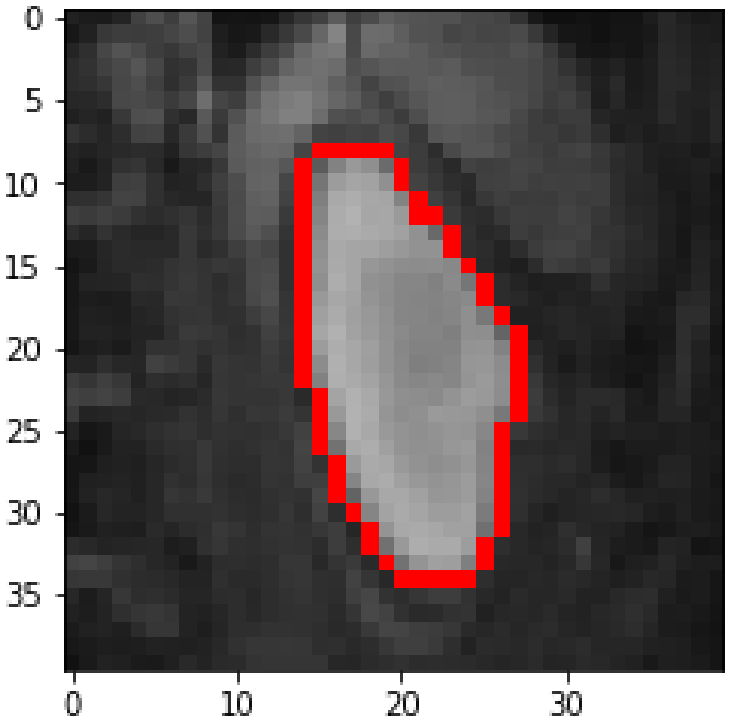


(a)

Effet du nombre de graines



(a)



(b)

Conclusion

- Première contrainte : Nombre de graines.
- (2006) [6] : Les coupes normalisées.

Conclusion

- Première contrainte : Nombre de graines.
- (2006) [6] : Les coupes normalisées.
- Deuxième contrainte : Résolution.
- Idée personnelle : $\mathcal{O}(\sqrt{n}^6 + \sqrt{n}^6 \times n) = \mathcal{O}(n^4)$ mais échec
- (2004) [7] : $\mathcal{O}(n^2 log(n))$

Conclusion

- Première contrainte : Nombre de graines.
- (2006) [6] : Les coupes normalisées.
- Deuxième contrainte : Résolution.
- Idée personnelle : $\mathcal{O}(\sqrt{n}^6 + \sqrt{n}^6 \times n) = \mathcal{O}(n^4)$ mais échec
- (2004) [7] : $\mathcal{O}(n^2 log(n))$
- Troisième contrainte : segmentation interactive.
- (2001) [3] : Termes régionales et automatisation de la sélection des graines

Conclusion

- Première contrainte : Nombre de graines.
- (2006) [6] : Les coupes normalisées.
- Deuxième contrainte : Résolution.
- Idée personnelle : $\mathcal{O}(\sqrt{n}^6 + \sqrt{n}^6 \times n) = \mathcal{O}(n^4)$ mais échec
- (2004) [7] : $\mathcal{O}(n^2 log(n))$
- Troisième contrainte : segmentation interactive.
- (2001) [3] : Termes régionales et automatisation de la sélection des graines
- (2006) [5] : Segmentation de volume

Merci de votre attention !

Références

[1] Alireza Norouzi, Mohd Shafry Mohd Rahim, Ayman Altameem, Tanzila Saba, Abdolvahab Ehsani Rad, Amjad Rehman Mueen Uddin : Medical Image Segmentation Methods, Algorithms, and Applications, : IETE Technical Review, 31 :3, 199-213.
[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein : Introduction to algorithms : Ch. 26 Maximum Flow 708.
[3] Yuri Y. Boykov Marie-Pierre Jolly : Interactive Graph Cuts for Optimal Boundary Region Segmentation of Objects in N-D Images : Proceedings of "Internation Conference on Computer Vision" , Vancouver, Canada, July 2001 vol.I, p.105.
[4] Julie Jiang : Image Segmentation with Graph Cuts : https ://julie-jiang.github.io/image- segmentation/image2graph.

Références

[5] Yuri Boykov, Gareth Funka-Lea : Graph Cuts and Efficient N-D Image Segmentation : International Journal of Computer Vision 70(2), 109–131, 2006.
[6] Shi, Jianbo, and Jitendra Malik. "Normalized cuts and image segmentation." IEEE Transactions on pattern analysis and machine intelligence 22, no. 8 (2000) : 888-905.
[7] Felzenszwalb, Pedro F., and Daniel P. Huttenlocher. "Efficient graph-based image segmentation." International journal of computer vision 59, no. 2 (2004) : 167-181.
[8] Images Google

Notations. Pour U et V deux ensembles de sommets, on note $f(U,V) = \sum_{u\in U}\sum_{v\in V} f(u,v)$ et pour u un sommet, on note $f(u,V) = f(\{u\},V)$ et $f(U-u,V) = f(U\setminus\{u\},V)$.

Lemme 4.1

1. $\forall U \subset S, f(U,U) = 0$
2. $\forall U,V \subset S, f(U,V) = -f(V,U)$
3. $\forall U,V,W \subset S$ tq $U\cap V = \emptyset$,
$$f(U\cup V, W) = f(U,W) + f(V,W)$$
$$f(W, U\cup V) = f(W,U) + f(W,V)$$

Démonstrations

Démonstration Lemme 4.1

1. En notant $U = \{u_1, u_2, ..., u_n\}$ on a :

$$\begin{aligned} f(U,U) &= \sum_{1 \leq i \leq j \leq n} f(u_i, u_j) + f(u_j + u_i) \\ &= \sum_{1 \leq i \leq j \leq n} f(u_i, u_j) - f(u_i + u_j) \\ &= 0 \end{aligned}$$

2. Conséquence immédiate de l'anti-symétrie de f
3. Sommation par paquets

Démonstration Proposition 2.1

D'après la conservation du flot,

$$\begin{aligned} f(E-s,S) &= \sum_{u\in E\setminus\{s\}} f(u,S) \\ &= \sum_{u\in E\setminus\{s\}} 0 \end{aligned}$$

D'après le Lemme 4.1 ,

$$\begin{aligned} f(E,T) &= f(E,S) - f(E,E) \\ &= f(E,S) \\ &= f(s,S) + f(E-s,S) \\ &= f(s,S) \\ &= |f| \end{aligned}$$

Lemme 4.2

Soit p un chemin améliorant de G_f. On définit la fonction

$$f_p : S^2 \longrightarrow \mathbb{R} \text{ par : } f(u,v) = \begin{cases} c_f(p) & \text{si } (u,v) \text{appartient à p} \\ -c_f(p) & \text{si } (v,u) \text{appartient à p} \\ 0 & \text{sinon} \end{cases}$$

Alors f_p est un flot de G_f de valeur $|f_p| = c_f(p)$ et $f' = f + f_p$ est un flot de valeur $|f'| = |f| + c_f(p) > |f|$.

Démonstration Lemme 4.2

On vérifie que f_p puis f' sont des flots par définition. On a évidemment $|f_p| = c_f(p)$ car p est un chemin simple puis on utilise la linéarité de la somme pour monter que : |f'|=|f|+|f_p|.

Démonstration Théorème Maxflow/Mincut

(1) => (2) : Raisonnons par l'absurde. Supposons que f soit un flot maximum mais que G_f contienne un chemin améliorant p. Alors d'après le Lemme 4.2, $f + f_p$ est un flot de G dont la valeur est strictement supérieure à $|f|$. Ce qui est absurde car f est un flot maximal.

(2) => (3) : On suppose que G_f n'ait pas de chemin améliorant. On définit alors $E = \{v \in S : \exists p = s \longrightarrow v \text{dans} G_f\}$ et $T = S - E$. (E, T) est alors une coupe. En effet, $s \in E$ et $t \in T$, car il n'existe aucun chemin de s vers t dans G_f. Pour chaque couple $(u, v) \in E \times T$, $f(u, v) = c(u, v)$ sinon en prenant $p = s \longrightarrow u$ un chemin dans G_f alors $q = s \longrightarrow u \rightarrow v$ est un chemin dans G_f ainsi $v \in E$ ce qui est absurde puisque T est non vide. Donc $|f| = c(E, T)$ car $|f| = f(E, T)$ d'après la Proposition 2.1.

Démonstration Théorème Maxflow/Mincu

(3) => (1) :

$$\begin{aligned} |f| &= f(E,T) \\ &= \sum_{u\in E}\sum_{v\in T} f(u,v) \\ &\leq \sum_{u\in U}\sum_{v\in V} c(u,v) \\ &= c(E,T) \end{aligned}$$

Or $|f| = c(E,T)$ donc f est un flot maximal.

Lemme 4.3

Lors de l'exécution de l'algorithme d'Edmonds-Karp, pour tout $v \in S \setminus \{s, t\}$, la longueur du plus court chemin de s à v dans G_f $\delta_f(s, v)$ augmente de façon monotone à mesure que le flot f augmente.

Démonstration Lemme 4.3

Raisonnons par l'absurde : on suppose que, pour un certain sommet $v \in S \setminus \{s, t\}$, il existe une augmentation de flot qui provoque la diminution de la distance de plus court chemin entre s et v. Soit f le flot juste avant la première augmentation qui diminue une certaine distance de plus court chemin, et soit f' le flot juste après. Soit v le sommet ayant $\delta_{f'}(s, v)$ minimale dont la distance a été diminuée par l'augmentation, de sorte que $\delta_{f'}(s, v) < \delta_f(s, v)$.

Démonstration Lemme 4.3

Soit $p = s \longrightarrow u \rightarrow v$ un plus court chemin de s à v dans $G_{f'}$, de sorte que $(u, v) \in A_{f'}$ et

$$\delta_{f'}(s, u) = \delta_{f'}(s, v) - 1$$

Compte tenu de la façon dont on a choisi v, on sait que l'étiquette distance du sommet$u$ n'a pas diminué. C'est-à-dire que :

$$\delta_{f'}(s, u) \geq \delta_f(s, u)$$

Nous avons $(u, v) \notin A_f$. En effet, si l'on avait $(u, v) \in A_f$, alors on aurait aussi

$$\begin{aligned} \delta_f(s, v) &= \delta_f(s, u) + 1 \\ &\leq \delta_{f'}(s, u) + 1 \\ &= \delta_{f'}(s, v) \end{aligned}$$

Démonstration Lemme 4.3

Ce qui contredit notre hypothèse que $\delta_{f'}(s, v) < \delta_f(s, v)$. Comment alors avoir $(u, v) \notin A_f$ et $(u, v) \in A_{f'}$? L'augmentation doit avoir accru le flot entre v et u. L'algorithme d'Edmonds-Karp augmente toujours le flot sur des plus courts chemins, et donc le plus court chemin de s à u dans G_f a (v, u) pour dernière arrête. Par conséquent,

$$\begin{aligned} \delta_f(s, v) &= \delta_f(s, u) - 1 \\ &\leq \delta_{f'}(s, u) - 1 \\ &= \delta_{f'}(s, v) - 2 \end{aligned}$$

Ce qui contredit notre hypothèse que $\delta_{f'}(s, v) < \delta_f(s, v)$. On en conclut que notre hypothèse selon laquelle il existe un tel sommet v est erronée.

Lemme 4.4

Lors de l'exécution de l'algorithme d'Edmonds-Karp, le nombre total d'augmentations de flot effectuées est $\mathcal{O}(SA)$.

Démonstration Lemme 4.4

On dit qu'un arc (u, v) d'un réseau résiduel G_f est critique sur un chemin améliorant p si $c_f(p) = c_f(u, v)$. Après que nous avons augmenté le flot le long d'un chemin améliorant, tout arc critique du chemin disparaît du réseau résiduel. De plus, il faut qu'un arc au moins soit critique dans chaque chemin améliorant. Nous allons montrer que chacun des $|A|$ arcs peut devenir critique au plus $\frac{|S|}{2} - 1$ fois. Soit u,$v \in S$ reliés par un arc de A. Puisque les chemins améliorants sont des plus courts chemins, quand (u, v) est critique pour la première fois, on a $\delta_f(s, v) = \delta_f(s, u) + 1$. Après augmentation du flot, l'arc (u, v) disparaît du réseau résiduel.

Démonstration Lemme 4.4

Il ne pourra pas réapparaître sur un autre chemin améliorant tant que le flux de u à v n'aura pas diminué, ce qui n'arrive que si (v, u) apparaît sur un chemin améliorant. Or d'après le lemme, dont en notant f' le flot au moment de cet évènement, on a $\delta_{f'}(s, u) \geq \delta_f(s, u) + 2$. En conséquence, entre le moment où (u, v) devient critique et celui où il devient à nouveau critique, la distance entre u et la source augmente au moins de 2. Donc, jusqu'à ce que u devienne éventuellement inaccessible à partir de la source, sa distance est au plus égale à $|S| - 2$. Donc, (u, v) peut devenir critique au plus $\frac{|S|-2}{2} = \frac{|S|}{2} - 1$ fois.

Démonstration Complexité Edmonds-Karp

Comme chaque itération peut s'implémenter en temps $\mathcal{O}(|A|)$ quand le chemin améliorant est trouvé par une recherche en largeur, le temps d'exécution total est $\mathcal{O}(|S||A|^2)$.

Code Edmonds-Karp

```
import heapq as hq
import numpy as np
def ParcoursLargeur(G,dep):
    n = len(G) ; sommets = [i for i in range(n)]
    L = {} #la longeur pour aller du départ au sommet considéré
    parent, statut = {} , {}
    for sommet in sommets:
        L[sommet], parent[sommet], statut[sommet] = np.inf, -1, "blanc"
    L[dep] = 0
    gris = [] #les sommets qu'il faudra considérer
    hq.heappush(gris, [0, dep])
    statut[dep] = "gris"
    while len(gris) > 0:
        courant = hq.heappop(gris)[1]
        statut[courant] = "noir"
        for i,l in enumerate(G[courant]) : # i le sommet, l la capacite
            if (not statut[i] == "noir") and (l != 0):
                chemin = L[courant] + 1
                if chemin < L[i]:
                    parent[i], L[i] = courant, chemin
                    hq.heappush(gris, [chemin, i])
    return parent
```

```
import copy
def EK(G,s,t):
    Gf = copy.deepcopy(G)
    n = len(G)
    f= [[0 for j in range(n)] for in range(n)]
    p = ParcoursLaGfeur(Gf,s)
    while p[t] != -1:
        i,j = p[t],t
        cap_res = np.inf
        while j != s:
            cap_res = min(cap_res, Gf[i][j])
            i,j = p[i],i
        i,j = p[t],t
        while j != s:
            f[i][j] += cap_res
            f[j][i] = -f[i][j]
            Gf[i][j] -= cap_res
            Gf[j][i] += cap_res
            i,j = p[i],i
        p = ParcoursLargeur(Gf,s)
    return f,Gf
```

Démonstration

Démonstration Validité de la segmentation

Soit $b \in \mathcal{B}$, on suppose par l'absurde que $b \notin E$ donc $b \in T$.
Puisque $(s, b) \in A$:

$$c(E, T) = c(s, b) + \sum_{u \in E:(u,b) \in A} c(u, b) + \sum_{u \in E, v \in T, v \neq b} c(u, v).$$

On sait que b admet un voisin w dans T. En posant $E' = E \cup \{b\}$ et $T' = S \setminus E$:

$$c(E', T') = \underbrace{c(b, w) + \sum_{u \in E:(u,w) \in A} c(u, w)}_{\leq \mathcal{K}} + \sum_{u \in E, v \in T, v \neq b} c(u, v)$$

Ainsi $c(E', T') < c(E, T)$ ce qui contredit la minimalité de la coupe (E,T).

Code segmentation d'images

```
from math import ceil, exp, pow
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
rouge = [255,0,0]
SIGMA = 30

#image importé sous forme d'une matrice n*n
im = mpimg.imread('/Users/fadijemmali/Downloads/test6.jpg').copy()
n=len(im)
b,o= seeds(im) #liste des graines obtenus grâce à la fonction seeds utilisant
#le module cv2

def affinite(ip, iq):
    A =exp(- pow(int(ip) - int(iq), 2) / (2 * pow(SIGMA, 2)))
    return A
```

```
def arete_normale(G, im):
    K = 0
    for i in range(n):
        for j in range(n):
            p = i*n+j
            if i+1<n: #il y a un voisin en bas
                q = (i+1)*n+j
                A = affinite(im[i][j],im[i+1][j])
                c = ceil(100 * A)
                G[p][q] = c
                G[q][p] = c
                K = max(K, c)
            if j+1<n: #il y a un voisin à droite
                q = i*n + j+1
                A = affinite(im[i][j],im[i][j+1])
                c = ceil(100 * A)
                G[p][q] = c
                G[q][p] = c
                K = max(K, c)
    return (K)
```

```
def arete_graine(G, o, b, K):
    for i in b:
        G[-2][i] = 10*K
    for j in o:
        G[j][-1] = 10*K
    #-2 c'est s et -1 c'est t

def graphe(image, o, b):
    m = len(image)*len(image) + 2 #on ajoute s et t
    G = np.array([[0 for j in range(n)] for i in range(n)])
    K = arete-normale(G,image)
    arete-graine(G, o, b, K)
    return G

def coupe(Gf, V, s, visite):
    pile = [s]
    while pile:
        v = pile.pop()
        if not visite[v]:
            visite[v] = True
            pile.extend([u for u in range(V) if (Gf[v][u] != 0)])
```

```
G = graphe(im,b ,o )
f,Gf = EK(G, n*n, n*n+1)

def segmentation(Gf):
    V = len(Gf)
    visite = [0 for i in range(V)]
    coupe(Gf, V, 0, visite)
    for i in range(V-2):
        if visite[i] :
            if (i+1<V-2 and not visite[i+1]) or (i+n<V-2 and not visite[i+n])
            or (0<=i-1 and not visite[i-1]) or (0<=i-n and not visite[i-n]):
                x,y = i//n, i%n
                im[x][y] = rouge

segmentation(Gf)
imgplot = plt.imshow(im)
plt.show()
```

Code Pousser-réétiqueter

```
def deplacer(t,x):
    def echanger(i,j):
        aux=t[i]
        t[i]=t[j]
        t[j]=aux
    for i in range(x,0,-1):
        echanger(i,i-1)
    return(t)

def preflot (G,s,t):
    n=len(G)
    A=[]
    for i in range(n):
        for j in range(n):
            if G[i][j]>0:
                A.append((i,j))
    S=[i for i in range(n)]
    L=[]
    for u in S:
        if (u!=s) and (u!=t):
            L.append(u)
    N={}
```

```
for u in S:
    N[u]=[]
    for v in S:
        if ((u,v) in A) or ((v,u) in A):
            N[u].append(v)
c={}
for u in S:
    for v in S:
        c[(u,v)]=G[u][v]
cf={}
for u in S:
   for v in S:
       cf[(u,v)]=G[u][v]
f , h, e= {}, {}, {}
for u in S:
    h[u], e[u] = 0, 0
for (u,v) in A:
    f[(u,v)], f[(v,u)]= 0, 0
h[s]=n
```

```
for u in N[s]:
        f[(s,u)]=c[(s,u)]
        cf[(s,u)]=c[(s,u)]-f[(s,u)]
        f[(u,s)]=-c[(s,u)]
        cf[(u,s)]=c[(u,s)]-f[(u,s)]
        e[u]=c[(s,u)]
        e[s]-=c[(s,u)]
    def pousser(u,v):
        df=min(e[u],cf[(u,v)])
        f[(u,v)]+=df
        cf[(u,v)]=c[(u,v)]-f[(u,v)]
        f[(v,u)]=-f[(u,v)]
        cf[(v,u)]=c[(v,u)]-f[(v,u)]
        e[u]-=df
        e[v]+=df
    def reetiqueter(u):
        m=infini
        for v in N[u]:
            if cf[(u,v)]>0:
                m=min(m,h[v])
        h[u]=1+m
courant={}
    for u in S:
        courant[u]=0 #ici courant[u] est l'indice
```

```
def decharger(u):
    while e[u]>0:
        i=courant[u]
        if i>(len(N[u])-1):
            reetiqueter(u)
            courant[u]=0
        else:
            v=(N[u])[i]
            if (cf[(u,v)]>0) and (h[u]==h[v]+1):
                pousser(u,v)
            else:
                courant[u]+=1
    return(False)
```

```
j=0
while (j<=(len(L)-1)):
    u, ah= L[j], h[u]
    decharger(u)
    if h[u]>ah:
        deplacer(L,j)
        j=1
    else:
        j=j+1
Gf = copy.deepcopy(G)
for i in range(n):
    for j in range(n):
       Gf[i][j]=cf[(i,j)]

return Gf
```