IG 4 - Algorithmique avancée

TP : application des flots au problème de segmentation d'images

---

Avant de commencer

Télécharger les fichiers fournis dans "squelette_code_segmentation.zip". Ces fichiers contiennent un squelette de code (avec des trous) qui compile, mais qui ne fait rien pour l'instant. Ne regardez pas le code pour l'instant, le code vous sera présenté à travers les questions. Commencez donc par la lecture des sections ci-dessous qui présentent le sujet globalement. L'ordre pour compléter les morceaux de code sera l'ordre des questions.
Ce TP est à faire par groupe de 2 (si quelqu'un se retrouve seul il peut trouver un binôme dans l'autre groupe, mais privilégiez les binômes au sein du même groupe), et sera corrigé par des tests automatiques. Certains de ces tests (les tests "publics") sont déjà présents dans le fichier TestAutomatiques.java, et les autres sont gardés secrets. Votre note sera égale à la somme des points des tests passés, ou 0 si votre code ne compile pas. Les tests présents dans la classe TestsAutomatiques représentent 7 points en tout (et 16 points pour les secrets). Nous vous encourageons à tester votre code sur ces tests publics dès que cela est possible, et à aussi écrire d'autres tests.
Veillez aux choses suivantes pour éviter le 0 stupide :

- ne changez pas les noms des classes, noms/droits des attributs, et signatures des méthodes fournies dans le squelette de code (vous pouvez par contre ajouter vos méthodes auxiliaires)
- n'ajoutez pas de package au début des fichiers fournis (attention si vous utilisez un IDE qui en ajoute automatiquement), n'ajoutez pas non plus de nouveau fichier
- créez un dossier appelé TP-NOM1-NOM2, et placez-y tout le squelette de code fourni ainsi que le dossier images. Si par exemple Frederic Chopin et Sergeï Rachmaninov font le TP ensemble, ils devraient avoir un dossier "TP-CHOPIN-RACHMANINOV/" contenant les fichiers "Couple.java, Img.java, TestsAutomatiques.java, etc..." et un sous dossier "images/".
- juste avant de soumettre votre travail : vérifiez, dans votre répertoire TP-NOM1-NOM2, que l'on peut bien faire sans erreur (car c'est ce qui sera fait par le script) :
  - rm *.class
  - javac TestsAutomatiques.java
  - java TestsAutomatiques
- le travail doit être remis sur MOODLE dans Rendu_TP_flot, sous la forme **d'un fichier .zip contenant votre dossier principal du TP (par exemple "TP-CHOPIN-RACHMANINOV.zip"**. Veillez à ce qu'un seul membre du groupe seulement dépose le fichier.
- la date limite de rendu est le **dimanche 6 novembre à 23h59**, mais n'hésitez pas à déposer une première version avant! (vous aurez le droit de re-déposer plusieurs fois).
- je ferai tourner le script une première fois le **lundi 31 octobre à 12h**, si vous avez remis (même une version préliminaire!) votre travail à ce moment là, je pourrai vous prévenir si il y a un problème de compilation (sans pénalité à ce moment là).
- jouez le jeu : vous pouvez parler entre groupes, mais ne copiez collez pas de code (pour information, des scripts de détection de plagiat seront lancés)

Le non respect de ces consignes entraînera un retrait de 5 points.

1 Le problème de Segmentation d'images

Le problème du flot maximum (ou de la coupe minimum) a de nombreuses applications dans le traitement d'images. Une des applications est le problème de la segmentation d'image. Dans ce problème, on considère une image dont on aimerait séparer le plan de fond (background) et le plan principal (foreground). Pour ce faire, l'utilisateur donne à l'algorithme l'image en question, et deux ensembles de pixels b et f. L'algorithme doit alors partitionner les pixels en deux régions B (et $\mathcal{P} \setminus B$, avec P dénotant l'ensemble des pixels de l'image) telles que

- $b \subseteq B$, et $f \subseteq \mathcal{P} \setminus B$ (autrement dit, b et f constituent des sous régions imposées dans le background et foreground respectivement)
- le coût $c(B)$ est minimal, où c est une fonction prédéfinie qui sera typiquement petite lorsque B correspond "bien" à un contour du background (voir ci-dessous pour la définition de c)

Dans l'article "Interactive Graph Cuts for Optimal Boundary and Region Segmentation of Objects in N-D Images" (Yuri Y. Boykov and Marie-Pierre Jolly), les auteurs traitent ce problème de segmentation en se réduisant au problème du flot max. [1]
Formalisons à présent le problème que nous allons résoudre dans ce TP.

SegmentationImage
Entrée: un triplet (im, f, b) avec

- im un objet de la classe Img représentant une image en niveaux de gris. Cette image est stockée sous la forme d'un tableau de pixels, avec l'hypothèse que pour tout pixel de coordonnée (i,j), im.get(i,j) est un entier dans $[0, 255]$ (avec 0 correspondant à un pixel noir, et 255 un pixel blanc. On fixe l'orientation suivante : i représentera le numéro de colonne, j celui de ligne, la colonne 0 étant à gauche, et la ligne 0 en haut.
- f et b sont des listes de couples d'entiers correspondant à la liste des pixels imposés dans le foreground et background respectivement, avec $f \cap b = \emptyset$

Sortie: une (f, b) coupe, c'est à dire sous ensemble de pixels B tel que $b \subseteq B$ et $f \subseteq \mathcal{P} \setminus B$
Objectif: minimiser $c(B)$ définie ci-dessous.

A noter que, étant donnée une solution B au problème de SegmentationImage, on pourra alors facilement créer une autre image $im2$ dans laquelle on isole le foreground du background. Pour ce faire, il suffira de (cf la question en fin de sujet sur la méthode "appliquerFiltre" de Img)

- mettre en blanc les pixels de b et en gris clair les pixels de $B \setminus b$ (le background devient donc gris clair, sauf les pixels de b qui sont blancs)
- mettre en noir les pixels de f, et laisser l'image intacte pour les pixels de $(\mathcal{P} \setminus B) \setminus f$ (le foreground reste intact, sauf les pixels de f qui sont noirs)

Afin de définir la fonction à minimiser c nous avons besoin des notations suivantes, qui sont illustrées Figure 1.

Arêtes de l'image. On dénote $\mathcal{P}$ l'ensemble des pixels, et $\mathcal{E}(\mathcal{P})$ l'ensemble des arêtes de l'image, une arête étant n'importe quelle paire de pixels voisins. On prend la convention qu'un pixel a pour voisins les pixels à distance 1 dans l'image. Un pixel au milieu de l'image aura donc 4 voisins (haut, bas, gauche, droit), un au bord aura 3 voisins, et un dans un coin 2 voisins. Par exemple sur une image 2x2, on a $\{(0,0),(0,1)\} \in \mathcal{E}(\mathcal{P})$ car les pixels $(0,0)$ et $(0,1)$ sont voisins, mais par exemple on a $\{(1,0),(0,1)\} \notin \mathcal{E}(\mathcal{P})$.

Coût d'une coupe. Etant donné deux niveaux de gris $nivx$ et $nivy$ (qui sont donc des entiers dans $[0, 255]$), on défini la fonction penalite(nivx, nivy) =

- soit v = |nivx-nivy|;
- si (v <= 10) return 1000;

[1] Voir également la version résumée ici https://julie-jiang.github.io/image-segmentation/

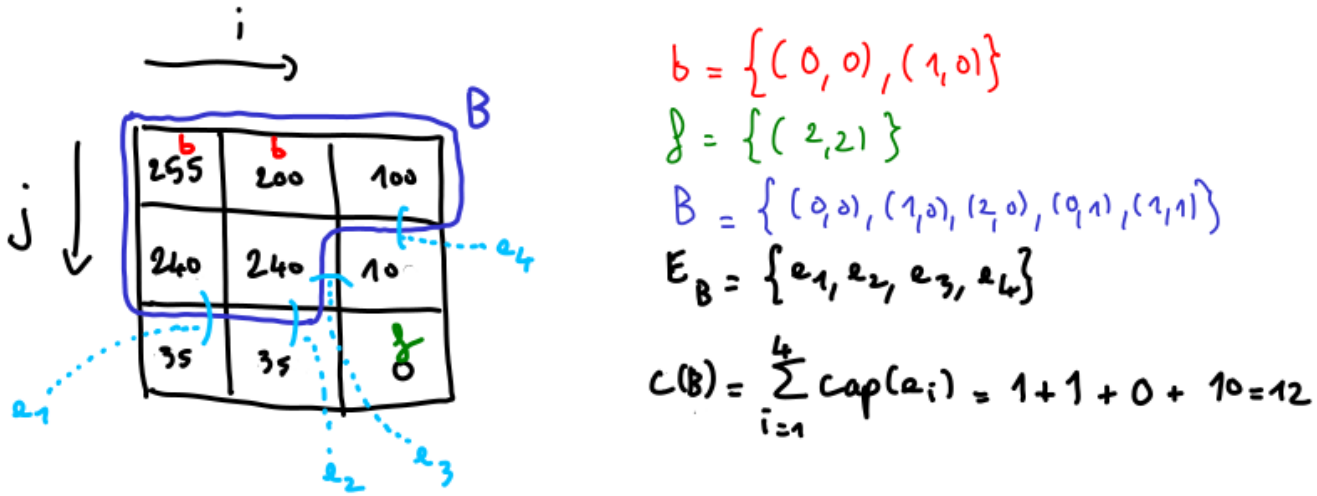

Figure 1: Une entrée du problème de SegmentationImage qui est ici une image en 3x3, ainsi qu'une coupe B.

- si (v <= 35) return 100;
- si (v <= 90) return 10;
- si (v <= 210) return 1;
- sinon return 0;

L'idée est que penalite(nivx,nivy) représente la pénalité payée lorsque deux pixels (de niveaux de gris nixv et nivy) se retrouvent d'une part et d'autre de la coupe B. Ainsi, plus nivx et nivy sont proches, plus la pénalité est grande, car cela signifie que l'on est en train de couper l'image dans une zone monochrome. Cette fonction de pénalité simpliste n'est pas celle proposée dans l'article ci-dessus, libre à vous à la fin du TP de la modifier pour observer les résultats (mais attention, pour le rendu il faudra bien remettre celle ci-dessus). Etant donné une arête $\{(i,j),(i',j')\} \in \mathcal{E}(\mathcal{P})$, le coût de l'arête est $cap(\{(i,j),(i',j')\}) =$
$penalite(im.get(i,j), im.get(i',j'))$. Etant donné une (f,b) coupe B, on dénote E_B les arêtes "à la frontière de B", c'est à dire $E_B = \{\{p_1,p_2\} \in \mathcal{E}(\mathcal{P})$ tel que $p_1 \in B$ et $p_2 \notin B\}$. Et enfin, la fonction $c(B)$ à minimiser est définie par

$$c(B) = \sum_{(i,j),(i',j') \in E_B} cap(\{(i,j),(i',j')\})$$

.

2 Transformation des instances : InstanceSegmentation → InstanceSegmentationGraphe → Reseau

2.1 Prise en main des classes Img et InstanceSegmentation

1. Dans la classe "Main", compléter le code de la méthode `public static Img creerImgLigne()` qui créer une Img d'une ligne et trois colonnes, avec, de gauche à droite, les teintes de gris suivantes : $255, 200, 100$. Pour ce faire, regardez quels sont les constructeurs/méthodes fournies dans la classe Img. Pensez à lancer les tests automatiques publics (en exécutant le main de la classe TestAutomatiques) pour valider cette question (et les suivantes .. on ne le répétera donc plus!).

Vous pouvez également (hors question, pas noté) dans le `main` de la classe `Main` récupérer l'Img produite par `creerImgLigne()`, et utiliser la méthode `creerImage(String fileName)` de la classe `Img` pour créer un fichier pgm et pouvoir ainsi observer cette image ligne. Attention, l'image est si petite que certains visionneurs peuvent avoir des comportements étranges!

2. Dans la classe `Main`, compléter le code de la méthode `public static InstanceSegmentation creerInstanceLigne()` qui crée une InstanceSegmentation contenant l'Img précédente, la liste de pixels $(2,0)$ pour f, et $(0,0)$ pour b. Pour ce faire, regardez quels sont les constructeurs/méthodes fournies dans la classe `InstanceSegmentation`.

2.2 De InstanceSegmentation à InstanceSegmentationGraphe.

Pour résoudre le problème SegmentationImage, nous allons tout d'abord le réduire vers un problème très similaire dans lequel, informellement, on remplace la notion d'image et de pixels voisins par un graphe sur lequel on place des capacités sur les arêtes. Cette formulation en terme de graphe nous facilite la notion de voisinage, et un autre avantage est que ce problème a un sens pour tout type de graphe, même si nous utiliserons uniquement ici des graphes de type grille. Voici tout d'abord ce nouveau problème, qui ressemble donc beaucoup à SegmentationImage.

SegmentationImageGraphe
Entrée: un triplet (g,f,b) avec

- g de type Graphe, représentant un graphe orienté avec des capacités sur les arcs ($cap(u,v) \geq 0$ est la capacité de l'arc $u->v$)
- f et b deux sous ensembles de sommets de g, avec $f \cap b = \emptyset$

Sortie: une (f,b) coupe, c'est à dire sous ensemble de sommets B tel que $b \subseteq B$ et $f \subseteq V(g) \setminus B$
Objectif: minimiser $c_2(B)$ définie ci-dessous.

Les notations précédentes sont adaptées aux graphes. On note $V(g)$ l'ensemble des sommets de g, et $E(g)$ l'ensemble de ses arcs. Pour toute (f,b) coupe B, on note E_B l'ensemble des arcs sortants de B, c'est à dire $E_B = \{(u,v) \in E(g) \mid u \in B \text{ et } v \notin B\}$, et on note

$$c_2(B) = \sum_{(u,v)\in E_B} cap(u,v)$$

.
Maintenant que le problème SegmentationImageGraphe est défini, nous allons effectuer une partie de la réduction SegmentationImage $\leq_S$ SegmentationImageGraphe : nous allons écrire la fonction qui associe une instance de SegmentationImageGraphe à une instance de SegmentationImage. En terme de Java, cela se traduit par le constructeur `public InstanceSegmenationGraphe(InstanceSegmentation isegm)` qui etant donné son paramètre *isegm* (composé d'un triplet (img, f, b)), va définir les attributs $(this.g, this.f, this.b)$ de l'objet InstanceSegmentationGraphe de la façon suivante

- $this.g$ doit être un Graphe à $n = |\mathcal{P}|$ sommets (n est donc égal au nombre de pixels de img). Les pixels de la première ligne seront associés aux sommets 0 (pour pixel en haut à gauche), 1, ..., $L-1$,puis pour la deuxième ligne L, $L+1$, etc. Le pixel de coordonnées (i,j) sera associé au sommet d'indice $calculIndice(i,j)$ (calculIndice est une méthode de Img)
- pour chaque sommets u et v du graphe (où u correspond à un pixel (i,j), et v correspond à un pixel voisin (i',j') de x,) on ait l'arc $u->v$ de capacité égale à la pénalité entre les niveaux de gris de (i,j) et (i',j')
- $this.f = \{calculIndice(i,j) | (i,j) \in f\}$ ($this.f$ contient tous les numéros de sommets associés aux pixels de f), et $this.b = \{calculIndice(i,j) | (i,j) \in b\}$

Voir la Figure 2 pour un exemple.

3. Dans la classe `InstanceSegmentationGraphe`, compléter le code du constructeur `public InstanceSegmentationGraphe(InstanceSegmentation isegm)`. Pensez à regarder les méthodes de la classe Graphe (pour savoir comment en construire un, et définir les capacités de ses arcs), et les méthodes de la classe Img pour les conversions entre coordonnées de pixels et numéros de sommets.

2.3 De InstanceSegmentationGraphe à Reseau

Pour résoudre le problème SegmentationImage, nous allons le réduire vers le problème de la coupe minimum dans les réseaux. On rappelle ce problème.

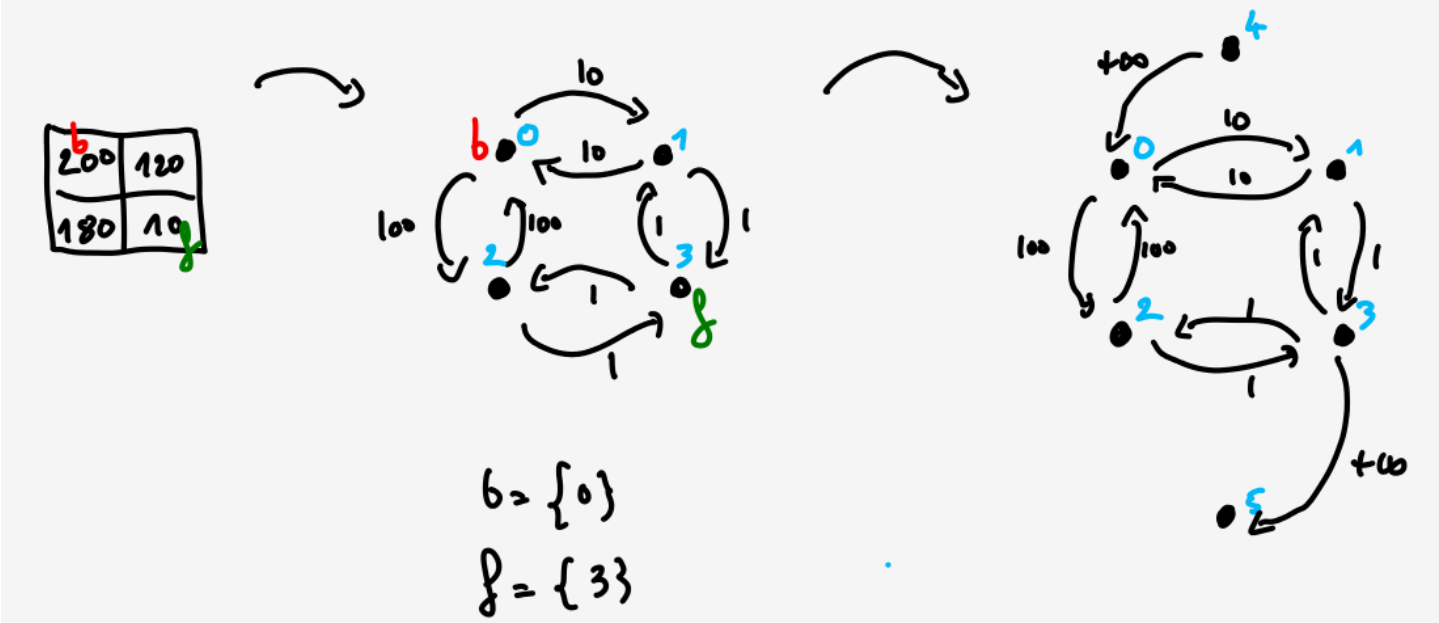

Figure 2: A gauche : une InstanceSegmentation. Au milieu : l'InstanceSegmentationGraphe correspondante. A droite : le Reseau correspondant.

MinCut
Entrée: un triplet (g,t,s) avec

- g de type Graphe, représentant un graphe orienté avec des capacités sur les arcs ($cap(u,v) \geq 0$ est la capacité de l'arc $u->v$)
- s et t deux sommets de sommets de g, avec $s \neq t$

Sortie: une (s,t) coupe, c'est à dire sous ensemble de sommets B tel que $s \in B$ et $t \notin B$
Objectif: minimiser $c_2(B)$ définie ci-dessus.

Le problème MinCut ressemble donc beaucoup à InstanceSegmentationGraphe, à la différence qu'on peut juste spécifier deux sommets s et t à "séparer", plutôt que deux ensembles f et b.
Maintenant que le problème MinCut est défini, nous allons effectuer une partie de la réduction SegmentationImageGraphe $\leq_S$ MinCut : nous allons écrire la fonction qui associe une instance de MinCut à une instance de SegmentationImageGraphe. En terme de Java, cela se traduit par le constructeur `public Reseau(InstanceSegmenationGraphe ins)` qui etant donné son paramètre *ins* (composé d'un triplet (g,f,b)), va définir les attributs $(this.g, this.s, this.t)$ de l'objet Reseau de la façon suivante

- $this.g$ doit être un Graphe à $n+2$ sommets (où n est le nombre de sommets de g), avec $this.s = n$, et $this.t = n+1$
- pour tout u,v sommets dans $[0, n-1]$, les arcs dans $this.g$ doivent être les mêmes que les arcs dans g
- les arcs $s->v$ sont à $+\infty$ (Integer.MAX_VALUE) pour $v \in b$, et 0 sinon
- les arcs $v->t$ sont à $+\infty$ (Integer.MAX_VALUE) pour $v \in f$, et 0 sinon

Voir la Figure 2 pour un exemple.

4. Dans la classe `Reseau`, compléter le code du constructeur `public Reseau(InstanceSegmenationGraphe inst)`.

3 Résolution, et traduction des solutions : InstanceSegmentation ← InstanceSegmentationGraphe ← Reseau

Nous avons maintenant retraduit le problème en un problème de MinCut sur un réseau. L'objectif de cette section est donc de résoudre optimalement MinCut pour trouver un sous ensemble B du Reseau, puis de traduire cette solution B en une solution pour l'InstanceSegmentationGraphe, puis pour l'InstanceSegmentation.

3.1 Résolution du problème max flot/min cut.

La méthode la plus importante de Reseau est la méthode `public ArrayList<Integer> coupeMin()` qui calcule une coupe minimale dans le réseau. Cette méthode se contente d'appeler `flotMaxCoupeMin()`, qui calcule à la fois un flot maximum et une coupe minimum.

5. Dans la classe `Reseau`, compléter le code de la méthode `public Couple<Flot, ArrayList<Integer>» flotMaxCoupeMin()`. Pensez à utiliser les méthodes `trouverCheminDansResiduel(..)` et `modifieSelonChemin(..)` (dans la classe `FLot`) qui vous sont fournies.

6. Dans la classe `Main`, compléter le code de la méthode `public static ArrayList<Integer> testMinCut()` qui crée le réseau de la Figure 3.1, et retourne une coupe minimum de ce réseau, et lancez ce test.

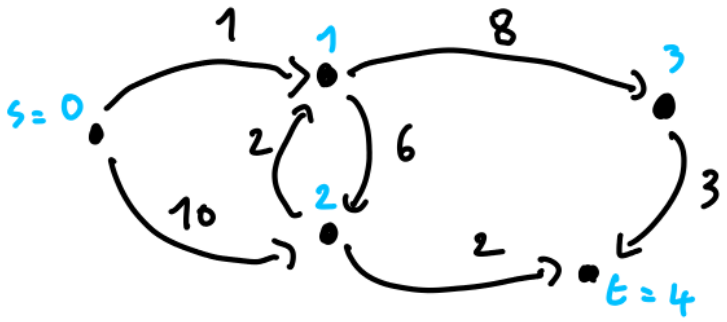

Figure 3: Un exemple de réseau.

3.2 Traduction des solutions : de Reseau à InstanceSegmentationGraphe.

Nous allons maintenant écrire la réduction SegmentationImageGraphe $\leq_S$ MinCut : pour résoudre une InstanceSegmentationGraphe, on la traduit en un réseau, on calcule une coupe minimale dans ce réseau, et on "traduit" cette coupe minimale en une solution pour SegmentationImageGrpahe. Cette réduction est à écrire dans la méthode suivante.

7. Dans la classe `InstanceSegmentationGraphe`, compléter le code de la méthode `public ArrayList<Integer> calculOpt()` qui calcule une solution optimale de l'`InstanceSegmentationGraphe` en se réduisant à MinCut.

3.3 Traduction des solutions : de InstanceSegmentationGraphe à InstanceSegmentation

Nous allons maintenant écrire la réduction SegmentationImage $\leq_S$ SegmentationImageGraphe : pour résoudre une InstanceSegmentation, on la traduit en une InstanceSegmentationGraphe, on calcule une solution optimale avec la question précédente, et on "traduit" cette solution en une solution pour SegmentationImageGrpahe. Cette réduction est à écrire dans la méthode suivante.

8. Dans la classe `InstanceSegmentation`, compléter le code de la méthode `ArrayList<Couple<Integer,Integer> > resoudre` qui calcule une solution optimale de l'InstanceSegmentation en se réduisant à InstanceSegmentationGraphe.

3.4 Création de l'image résultat

Observez le code de la méthode `creerImageSegmentee()` de `InstanceSegmentation`, qui se sert donc de la solution optimale calculée pour créer l'image résultat.

9. Dans la classe `Img`, compléter le code de la méthode `public Img appliquerFiltre (ArrayList<Couple<Integer,Integer> > B, ArrayList<Couple<Integer,Integer> >f, ArrayList<Couple<Integer,Integer> >b)` qui crée une nouvelle image (et ne modifie pas this), dans laquelle on :

- met en blanc (255) les pixels de b et en gris clair (200) les pixels de $B \setminus b$ (le background devient donc gris clair, sauf les pixels de b qui sont blancs)
- met en noir (0) les pixels de f, et laisse l'image intacte pour les pixels de $(\mathcal{P} \setminus B) \setminus f$ (le foreground reste intact, sauf les pixels de f qui sont noirs)

3.5 Test réel

Le TP est fini, essayez maintenant sur un exemple réel : l'image "baby_2k.pgm" présente dans votre dossier images. Pour cela, vous pouvez dé-commenter le test (non noté) présent dans votre main. Ce test créer une Img à partir de l'image du bébé, ajoute quelques points de foreground et background, et calcule une image segmentée qui est écrite dans le fichier "images/output_baby_2k.pgm". Vous pouvez également changer la fonction de pénalité, mais attention : pensez bien à remettre celle fournie de base pour le rendu.