Introduction aux GraphCuts en Vision par Ordinateur

Mickaël Péchaud

26 janvier 2007

Table des matières

Introduction

Depuis 1999, les *Graph Cuts* sont de plus en plus utilisés dans des problèmes de Vision par Ordinateur variés. Ils ont été introduits en Vision en 1989 [27], puis oubliés pendant près de 10 ans avant un retour marquant dans le domaine. Cette puissante méthode d'optimisation combinatoire a trouvé des applications pour des problèmes tels que la restauration bayesienne, le défloutage, la segmentation avec ou sans a priori, la stéréo 2 ou multi-caméra. Une des facettes les plus attrayantes des *Graph Cuts* est qu'ils permettent sous certaines conditions de trouver des minima *globaux* de fonctions, évitant ainsi l'écueil classique des minima *locaux*.

Ce modeste polycopié a pour but d'expliquer ce que sont les *Graph Cuts*, ce qu'ils peuvent faire et ne pas faire et quelles ont été leurs applications marquantes en Vision par Ordinateur. Certaines parties sont volontairement peu détaillées, et j'essaye autant que faire se peut de renvoyer vers des références utiles.

J'effectue ce travail alors que je suis en train de tenter d'appliquer ce type de méthodes au problème inverse en magnétoencéphalographie. J'espère qu'il pourra aider les étudiants ou chercheurs désireux de découvrir ce domaine.

N'hésitez pas à me signaler toute coquille ou à me faire part de toute remarque à `pechaud(at)di(point)ens(point)fr`

Le premier chapitre va présenter les *Graph Cuts* d'un point de vue purement combinatoire. Les deux suivants vont se focaliser sur l'utilisation des *Graph Cuts* comme de puissants minimisateurs d'énergie, et le troisième donnera un aperçu des applications possibles en vision.

Chapitre 1

Graph Cuts : comment ça marche ?

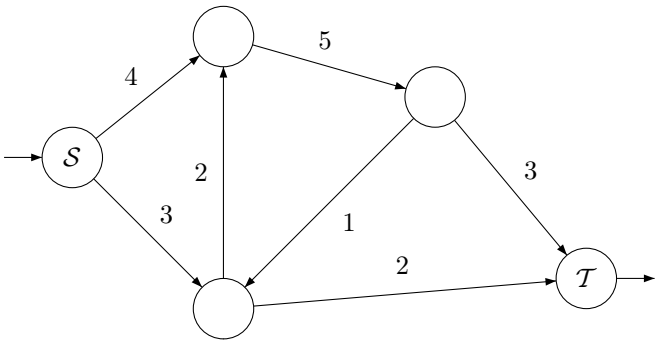
1.1 Notations et définitions

(NB : je vais adopter dans la suite les notations anglaises classiques en théorie des graphes : $\mathbf{V}$ pour *vertices*, $\mathbf{E}$ pour *edges*, w pour *weight*.)

Considérons un graphe orienté $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, où $\mathbf{V}$ est l'ensemble des sommets du graphe, et $\mathbf{E} \subset \mathbf{V}^2$ l'ensemble de ses arêtes (orientées, i.e. $(a, b) \neq (b, a)$).

Prenons de plus une fonction $w : \mathbf{E} \rightarrow \mathbb{R}^+ \cup +\infty$, qui à toute arête associe son poids (sa capacité - nous verrons plus tard pourquoi ce terme). Notons tout de suite que les valeurs prises par w sont nécessairement positives.

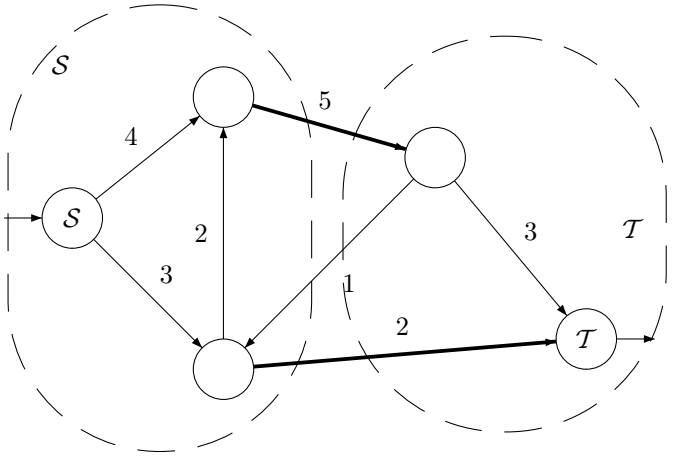
Distinguons de plus 2 sommets $\mathcal{S}$ et $\mathcal{T}$ (respectivement la *source* et le *puits*). L'intérêt de cette terminologie apparaîtra par la suite. On va supposer de plus qu'il n'y a pas d'arête entrante dans $\mathcal{S}$, ni d'arête sortante de $\mathcal{T}$.



Définition 1.1 (Coupe) *: On appelle coupe dans un graphe une partition* $(\mathbf{S}, \mathbf{T})$ *de l'ensemble des sommets (i.e.* $\mathbf{S} \cup \mathbf{T} = \mathbf{V}$ *et* $\mathbf{S} \cap \mathbf{T} = \emptyset$*) telle que* $\mathcal{S} \in \mathbf{S}$ *et*

$\mathcal{T} \in \mathbf{T}$.

De façon équivalente, on peut se donner $F \subset \mathbf{E}$ un sous-ensemble des arêtes tel que dans $\mathbf{G} = (\mathbf{V}, \mathbf{E} - F)$ il n'y ait pas de chemin de $\mathcal{S}$ à $\mathcal{T}$ et tel qu'il n'y ait pas de sous-partie stricte de F vérifiant cette propriété. Les deux définitions coexistent dans la littérature, mais je trouve que la première donne une intuition plus claire de ce qu'on est en train de faire.

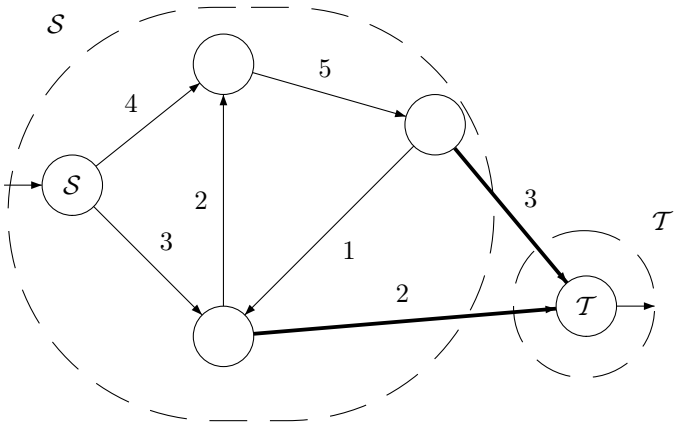


Définition 1.2 (Poids d'une coupe) *: On appelle poids d'une coupe* $(\mathbf{S}, \mathbf{T})$ *le réel positif*

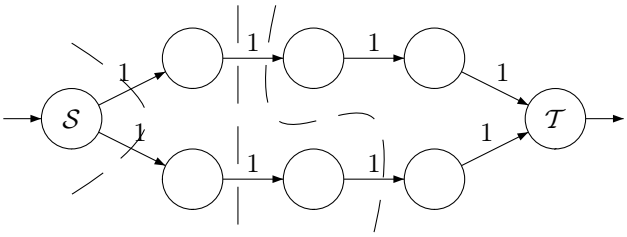
$$\sum_{\substack{(p,q)\in\mathbf{E} \\ p\in\mathbf{S}, q\in\mathbf{T}}} w(p,q) \tag{1.1}$$

Si on a pris la définition équivalente, il s'agit juste de sommer les poids des arêtes dans F.

Définition 1.3 (Coupe minimale - MinCut) *: On appelle coupe minimale pour un graphe* $\mathbf{G}$ *une coupe de poids minimal.*



Il peut évidemment en exister plusieurs :



Comme nous allons le voir par la suite, de nombreux problèmes de vision peuvent se reformuler sous la forme de recherche de coupe minimale dans un graphe.

1.2 Recherche de la coupe minimale

Tout l'intérêt de la reformulation de problèmes divers en recherche d'une coupe minimale pour un graphe bien choisi réside dans le théorème suivant :

Théorème 1.1 *Rechercher une coupe minimale dans un graphe est un problème* **P**.

Notons que de nombreux problèmes similaires, telle la recherche d'une *coupe maximale* dans un graphe sont **NP**-complet.

1.3 Reformulation en terme de flot maximal

Le problème du calcul d'une coupe minimale est d'une certaine façon dual du problème bien connu du calcul du flot maximal dans un graphe.

On peut imaginer un *flot* comme un flot de liquide allant de la source vers le puits, via des arêtes ayant une certaine capacité.

Définition 1.4 (flot) *Soit* $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ *un graphe,* w *la fonction capacité (qui doit être positive),* $\mathcal{S}$ *et* $\mathcal{T}$ *une source et un puits. On appelle* flot *une fonction* $f : \mathbf{E}^* \to \mathbb{R}$ *(*$\mathbf{E}^*$ *est l'ensemble des arêtes et de leur inverse) vérifiant les propriétés suivantes :*

- *pour tout* $e = (p, q) \in \mathbf{E}$

$$f(p, q) = -f(q, p) \tag{1.2}$$

- *pour tout sommet* p *autre que* $\mathcal{S}$ *ou* $\mathcal{T}$*, on a*

$$\sum_{\substack{e=(p,.)\\ e\in\mathbf{E}^*}} f(e) = 0 \tag{1.3}$$

- *pour toute arête* $e \in \mathbf{E}$*, on a*

$$f(e) \leq w(e) \tag{1.4}$$

(1.3) est une contrainte de conservation du flot en chaque sommet, soit une loi de Kirchoff.

(1.4) stipule qu'une arête ne peut contenir un flot dépassant sa capacité.

On voit immédiatement grâce à (1.3) et (1.2) que

$$\sum_{e=(\mathcal{S},.)} f(e) = \sum_{e=(.,\mathcal{T})} f(e) \tag{1.5}$$

On appellera cette quantité *valeur du flot*.

Un *flot maximal* sera tout naturellement un flot de valeur maximale.

Attention, on appelle aussi souvent flot maximal (MaxFlow) la valeur d'un flot maximal.

On a alors le théorème suivant [10] :

Théorème 1.2 *Pour un graphe* **G** *vérifiant nos hypothèses, la valeur d'une coupe minimale est égale à la valeur d'un flot maximal.*

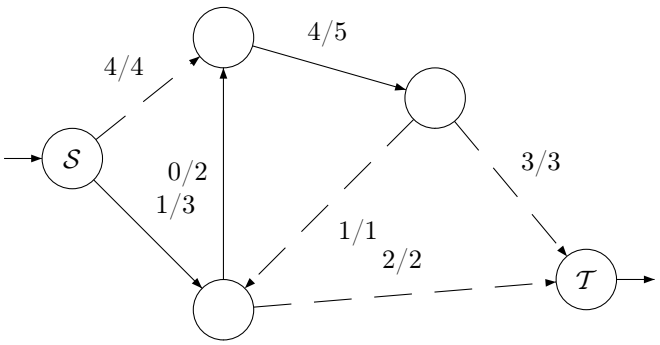
(Le même théorème existe pour le cas non-orienté [43]).

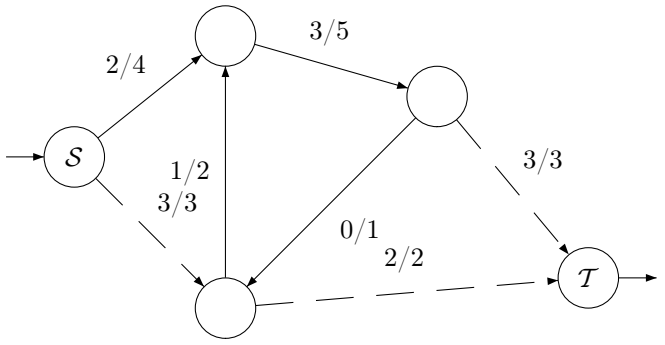
De plus toute arête $e = (p, q)$ contenue dans une coupe minimale (i.e. telle que $p \in \mathbf{S}$ et $q \in \mathbf{T}$ où $\mathbf{S}, \mathbf{T}$ est une coupe minimale) est une arête saturée.

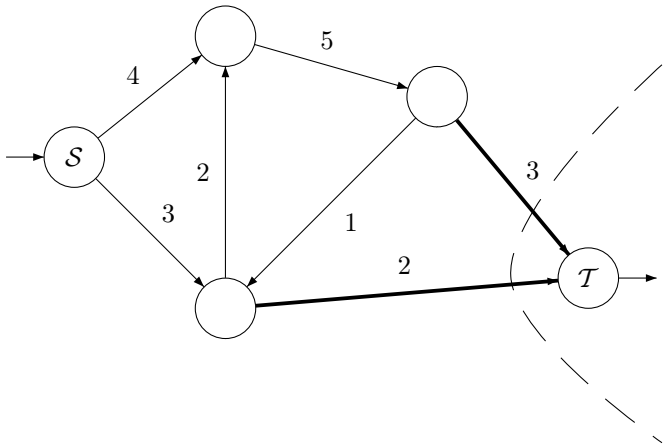
On voit donc que si l'on sait résoudre le problème du flot maximal dans un graphe, on peut très simplement en déduire une coupe minimale : il suffit de prendre l'ensemble des arêtes saturées et d'en retirer itérativement les arêtes inutiles.

Notons également qu'il n'existe pas une telle propriété si le graphe contient des arêtes de poids négatif : en effet, la notion de plus court chemin n'y est même pas forcément définie.

Ci-dessous, 2 flots maximaux, et la coupe minimale correspondante (les arêtes saturées sont en pointillés).







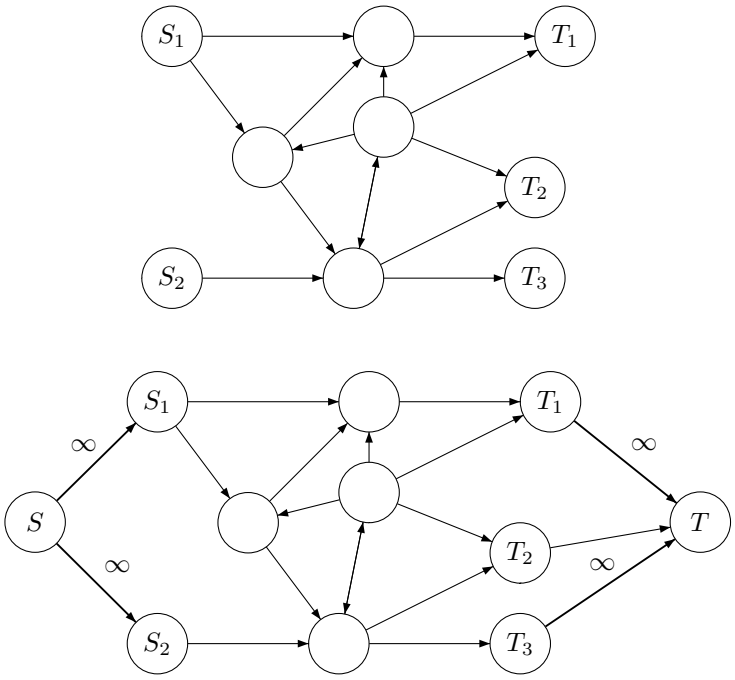
1.4 Quelques problèmes connexes

1.4.1 Plusieurs sources, plusieurs puits

Regroupement de sources et de puits

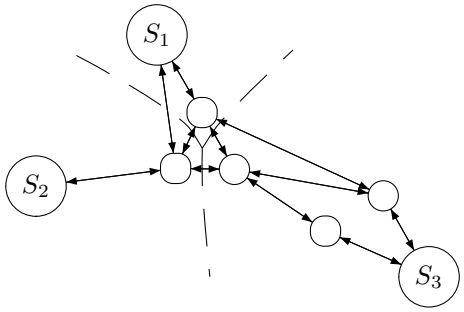
Dans certains problèmes, comme nous le verrons par la suite, il peut être souhaitable d'autoriser à avoir plusieurs sources et/ou plusieurs puits. On se ramène alors au cas à une source ou un puits de façon très simple :

- si on a n sources $\{\mathcal{S}_1, \dots \mathcal{S}_n\}$, on rajoute au graphe un sommet $\mathcal{S}$ et des arêtes de capacité infinie de $\mathcal{S}$ vers $\mathcal{S}_i$.
- si on a n puits $\{\mathcal{T}_1, \dots \mathcal{T}_n\}$, on rajoute au graphe un sommet $\mathcal{T}$ et des arêtes de capacité infinie de $\mathcal{T}_i$ vers $\mathcal{T}$.



Multi-coupes

«Historiquement» il existe une reformulation d'un problème de minimisation d'énergie par des multicoupes dans un graphe [7]. Au lieu d'avoir une source et un puits, on a n *graines* et on souhaite partitionner le graphe en n sous-parties contenant chacune une graine. Ce problème étant **NP**-complet, les auteurs proposent une approximation, qui dans les formalismes ultérieurs apparaitra comme celui des (α, β)-*swap*. À noter que dans cet article, l'algorithme est appliqué au problème de la stéréo deux caméras.



1.5 Trouver un flot maximal

De nombreux algorithmes, souvent assortis de leur cohorte d'heuristiques, ont été développés pour trouver un flot maximal dans un graphe. Ils se divisent en deux grandes classes : les algorithmes par «saturation de chemins» (type Ford-Fulkerson) et les algorithmes par «poussage de flot» (type *Push Relabel*).

Cette section peut-être omise en première lecture.

Je conseille la lecture de [52] pour un tour d'horizon complet des algorithmes de flot maximal.

Les graphes que nous allons devoir couper vont cependant avoir des formes très particulières (en particulier des chemins courts de la source au puit), ce qui peut permettre d'introduire des heuristiques *ad hoc* permettant d'accélérer les algorithmes.

1.5.1 Flot maximal par saturation de chemins

L'idée de base est à partir du flot nul de trouver itérativement un chemin de la source au puits sur lequel il n'y a pas d'arête saturée. On rajoute alors autant de flot que possible à ce chemin (i.e. on sature l'arête qui a une capacité résiduelle minimale).

Voyons ce que celà donne plus précisément :

Définition 1.5 (capacité résiduelle) *On appelle* capacité résiduelle *de* $(p, q) \in \mathbf{E}^*$ *et on note* $r(p, q)$ *la quantité* $w(p, q) - f(p, q)$.

(avec la convention $w(p, q) = 0$ si $(p, q) \notin \mathbf{E}$).

Intuitivement, la capacité résiduelle mesure la quantité de flot que l'on peut faire passer en plus dans une arête (p, q) (ou que l'on peut retirer de l'arête (q, p) dans le cas où $(q, p) \in \mathbf{E}$), tout en respectant les contraintes de capacités.

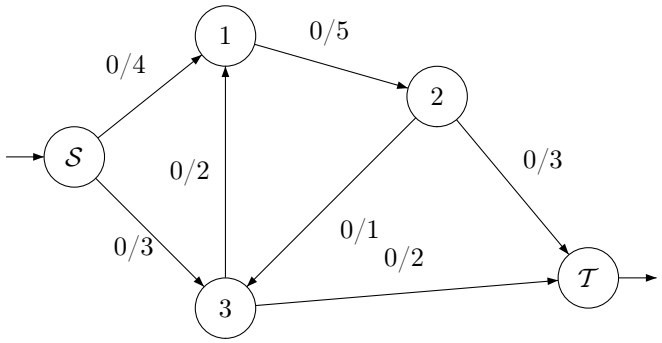
Définition 1.6 (graphe résiduel) *On appelle* graphe résiduel *le graphe* $\mathbf{G}^* = (\mathbf{V}, \mathbf{E}^*, r)$.

Algorithme 1 Algorithme par saturation de chemins

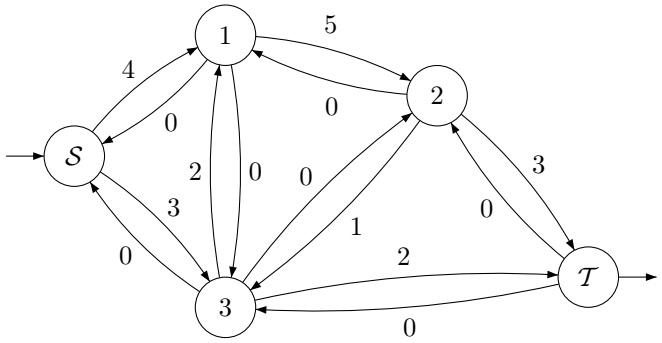
```
Initialisation:
Poser f(p,q) = 0 pour toute arête (p,q) ∈ E*.
tantque il existe un chemin K de S à T dans G* faire
  a ← min(r(p,q)|(p,q) ∈ K))
  pour tout (p,q) ∈ K faire
    f(p,q) ← f(p,q) + a
  fin pour
fin tantque
```

On peut montrer qu'au bout d'un certain nombre d'itérations, il n'y a plus de chemin augmentant si les capacités des arêtes sont rationnelles (un contre-exemple dans le cas général est présenté dans [43][52]). On peut alors démontrer que le flot obtenu est un flot maximal.

Voici un exemple d'application de ce type d'algorithme sur notre graphe. Considérons le graphe de départ suivant :

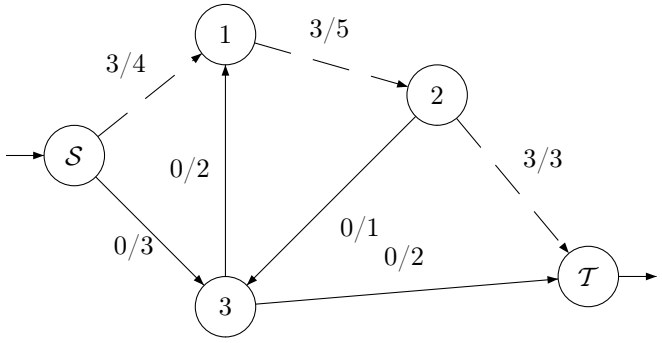


On a le graphe résiduel suivant :

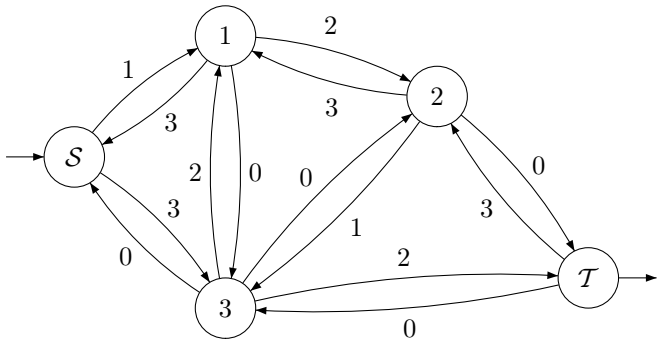


On constate que le chemin de $\mathcal{S}$ vers $\mathcal{T}$ $(\mathcal{S}, 1, 2, \mathcal{T})$ est un chemin dans le graphe résiduel, les 3 arêtes le constituant n'étant pas saturées.

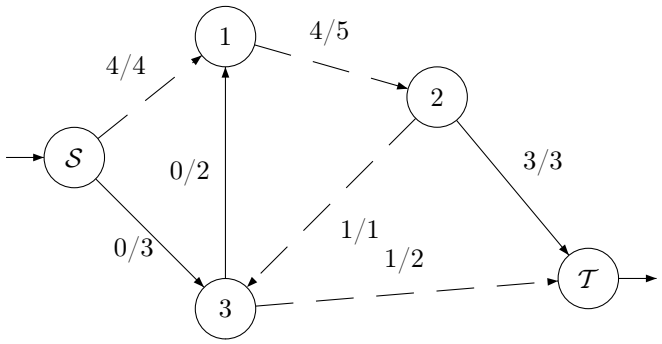
On peut donc rajouter un flot de valeur 3 (i.e. la valeur maximale des capacités résiduelles des arêtes du chemin) sur ce chemin. On obtient alors ceci :



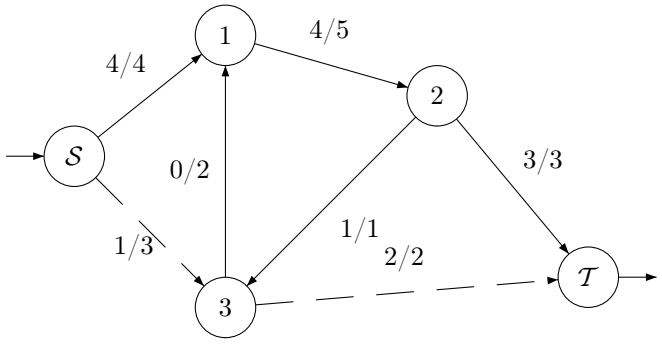
et le graphe résiduel correspondant :



À l'itération suivante, on peut encore rajouter un flot de 1 sur le chemin $(\mathcal{S}, 1, 2, 3, \mathcal{T})$ - les arêtes limitantes étant $(\mathcal{S}, 1)$ et $(2, 3)$.



Ensuite, on rajoute 1 au chemin $(\mathcal{S}, 3, \mathcal{S})$



Il n'y a alors plus de chemin augmentant de $\mathcal{S}$ vers $\mathcal{T}$. L'algorithme s'arrête et renvoie bien un flot maximal de notre graphe.

Notons que le nombre d'itérations peut être exponentiel en la taille des don-

nées avec un choix arbitraire des chemins augmentants.

L'algorithme de Dinic ([13] - redécouvert par Edmonds et Karp 2 ans après [14]) reprend ce principe mais cherche à chaque étape un chemin de longueur minimale dans le graphe résiduel. On obtient alors un algorihme en $O(|\mathbf{V}||\mathbf{E}|^2)$

D'autres algorithmes basés sur les mêmes principes (Dinits [11], Karzanov [34]).

Dans [5], les auteurs proposent un algorithme de type Dinic symétrisé et dynamique dans le sens où les arbres de recherches de chemins dans le graphe résiduel sont conservés et modifiés à chaque itération, plutôt que complètement recalculés. En pratique, il s'agit de l'algorithme le plus rapide actuellement utilisé. Bien que de complexité théorique plus grande que d'autres algorithmes, son temps de calcul observé est quasiment linéaire en la taille des données sur des graphes $2D$.

1.5.2 Flot maximal par poussage de flot

[21] [22] [52]

L'idée de cette algorithme est de considérer non plus des flots mais des pré-flots, qui ne respectent plus la contrainte de conservation du flot : on envoie autant de flot que l'on peut à partir de la source. On se retrouve alors avec des noeuds dits actifs, c'est à dire qui reçoivent un excès de flot. On pousse alors ce flot excessif vers d'autres noeuds disponibles.

Plus précisément :

Définition 1.7 (pré-flot) *On appelle* pré-flot *une fonction $f : \mathbf{E}^* \to \mathbb{R}^+$ vérifiant les mêmes hypothèses qu'un flot, mais où 1.3 est remplacé par :*

$$\sum_{\substack{e=(p,.)\\ e\in\mathbf{E}^*}} f(e) \geq 0 \tag{1.6}$$

pour tout sommet p différent de $\mathcal{S}$ ou $\mathcal{T}$.

Définition 1.8 (sommet actif) *On appelle* sommet actif *du graphe un sommet où 1.3 n'est pas respectée.*

Définition 1.9 (excès de flot) *On appelle* excès de flot *en un sommet p la quantité*

$$e(p) = \sum_{\substack{e=(p,.)\\ e\in\mathbf{E}^*}} f(e) \tag{1.7}$$

Il s'agit donc de la quantité de flot en trop. Un sommet est actif si et seulement si $e(p) > 0$.

Après l'initialisation, f est un pré-flot.

d mesure une sorte de proximité au puits.

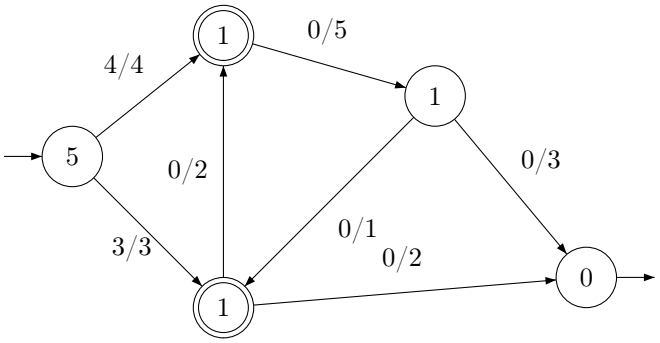
À chaque itération, on va essayer de pousser du flot vers le puits. Si on ne peut pas, on change l'étiquetage du sommet, ce qui l'«éloigne» du puits (on poussera plus difficilement vers ce sommet par la suite).

Voyons les premières itérations de l'algorithme sur un exemple.

Algorithme 2 Algorithme par poussage de flot

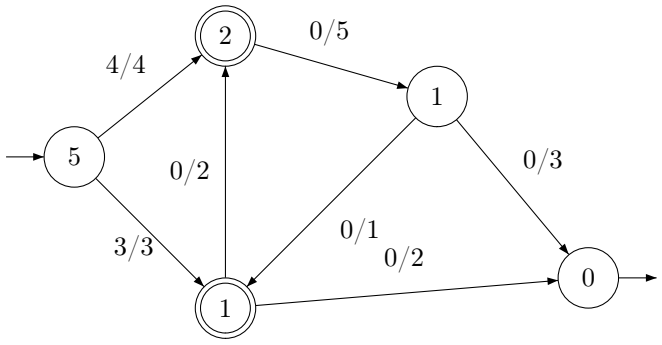
Initialisation:
Étiqueter les noeuds de la façon suivante :
– $\mathcal{S}$ reçoit l'étiquette $d(\mathcal{S}) = |\mathbf{V}|$.
– $\mathcal{T}$ reçoit l'étiquette $d(\mathcal{T}) = 0$.
– tout autre noeud p reçoit l'étiquette $d(p) = 1$.
Poser $f(\mathcal{S}, p) = w(\mathcal{S}, p)$ pour toute arête $(\mathcal{S}, p)$.
tantque il y a des sommets actifs, en choisir un (i) **faire**
  tantque i est actif **faire**
    si il existe $j \in \mathbf{V}$ tel que $(i, j) \in \mathbf{E}^*$, $d(i) = d(j) + 1$ et $r(i, j) > 0$ **alors**
      pousser autant d'excès que l'on peut vers j
      (i.e. mettre à jour f en ajoutant $min(e(i), r(i, j))$ à $f(i, j)$).
      POUSSAGE
    sinon
      ré-étiqueter i de la façon suivante :
      $d(i) \leftarrow min(d(j) + 1 | (i, j) \in \mathbf{E}^* \ \& \ r(i, j) > 0)$
      RÉ-ÉTIQUETAGE
    finsi
  fin tantque
fin tantque

Considérons notre graphe, après initialisation (les sommets actifs sont entourés) :

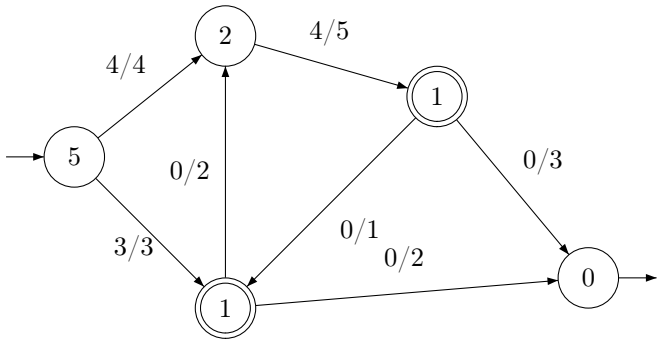


Sélectionnons le sommet actif du haut.

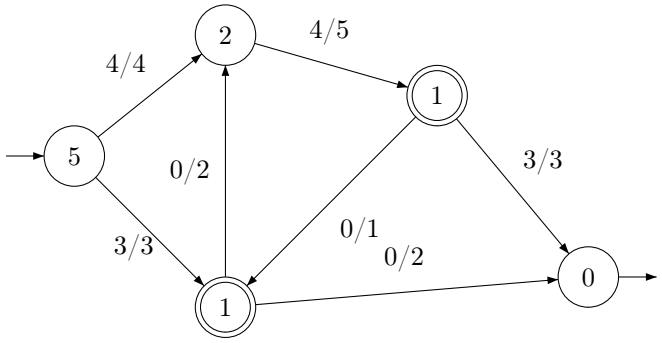
On ne peut clairement pas faire de *POUSSAGE* à partir de ce sommet. On fait donc un *RÉ-ÉTIQUETAGE*, et l'étiquette du sommet passe à $1 + 1 = 2$. Notons que le sommet reste actif.



On peut maintenant pousser du flot à partir de ce sommet vers le sommet étiqueté à 1. La capacité résiduelle de l'arête correspondante étant 5, on peut pousser tout l'excédant du sommet actif, soit 4.

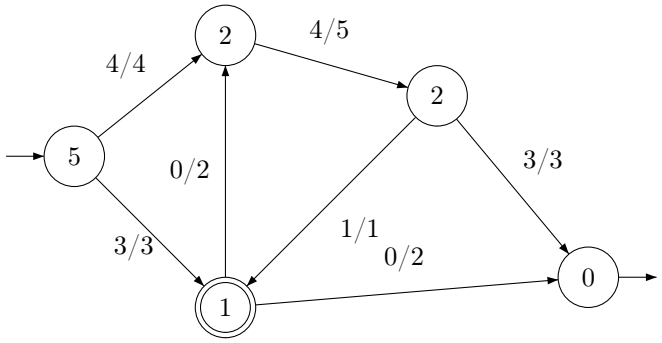


Notre sommet n'est alors plus actif. Choisissons le sommet actif nouvellement créé. On peut pousser 3 vers le puits.



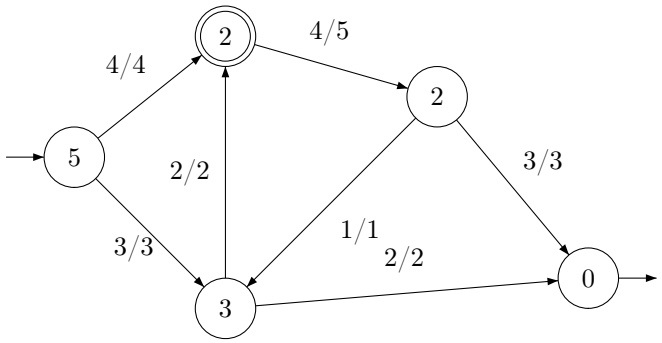
Notre sommet reste actif. Effectuons un *RÉ-ÉTIQUETAGE* dessus. On peut

alors pousser 1 vers l'autre sommet actif.



Sélectionnons le dernier sommet actif.

Celui-ci a un excès de 4. On commence par pousser 2 vers le puits. Nous effectuons ensuite 2 *RÉ-ÉTIQUETAGE* d'affilée. L'excés de flot est alors poussé par exemple vers le sommet situé au-dessus.



etc, etc...

Au final les sommets centraux vont se renvoyer du flot et augmenter leur étiquette. Lorsque l'un des sommets connectés à la source aura une étiquette 6, il poussera tout son excédent de flot vers la source, et l'algorithme sera terminé.

On obtient un algorithme en $O(|\mathbf{V}^3|)$, sur lequel peuvent se greffer de nombreuses heuristiques. Par exemple, plutôt que d'initialiser les étiquettes des sommets centraux à 1, on peut les initialiser à la distance au puits, ce qui évitera des étapes de *RÉ-ÉTIQUETAGE* inutiles.

1.6 Variations sur les algorithmes de coupe minimale/flot maximum

1.6.1 Utilisation de la structure particulière du graphe

Comme nous le verrons par la suite, les graphes pour lesquels on a à calculer des coupes minimales ont souvent des formes particulières très spécifiques :

Par exemple, pour beaucoup de problèmes $2D$, le graphe sera constitué de noeuds aux intersections d'une grille $2D$, avec comme système de voisinage les 4 plus proches voisins, la source et le puits étant situés au-dessus et au-dessous du plan du graphe, et reliés à tous les autres noeuds.

On peut profiter de ces structures particulières en utilisant des algorithmes dédiés.

Par exemple [27] propose une approche hiérarchique en $2D$ pour accélérer le calcul du flot maximal.

1.6.2 Utilisation itérative du flot : dynamic et active *Graph Cuts*

Dynamic cuts

Les *Dynamic Cuts* sont un algorithme développé dans [36] qui permet de calculer un flot maximal dans un graphe en connaissant déjà un flot maximal dans un graphe approchant, où seuls les poids de certaines arêtes changent. Expérimentalement, l'algorithme tourne en un temps linéaire en le nombre d'arêtes modifiées.

Ceci peut permettre de trouver des coupes minimales dynamiquement dans des graphes dont les poids sont mis-à-jour en temps réel. Dans [36], l'algorithme est utilisé pour faire de la segmentation dans des séquences video.

Active *Graph Cuts*

Les *Active Graph Cuts* sont une méthode développée dans [46], et qui permet de calculer une coupe minimale en partant d'une autre coupe minimale. Notons que contrairement au cas précédent, un flot maximal complet n'est pas nécessaire. Les *Active Graph Cuts* vont donc être applicables en changeant la topologie du graphe par exemple.

Outre les applications des *Dynamic Cuts*, les *Active* Graph Cuts vont permettre de faire du multi-échelle. L'utilisation de cet algorithme peut conduire à un gain de temps considérable dans ces cas.

Chapitre 2

Graph Cuts comme minimisateurs d'énergie

2.1 Le problème

Un grand nombre de problèmes en Vision par Ordinateur peuvent se reformuler en terme de minimisation d'énergies. C'est par exemple le cas de nombreux problèmes de segmentation, de stéréo, de restauration d'images...

Les *Graph Cuts* permettent d'obtenir de très bon résultats sur une importante classe de problèmes de ce type.

La forme de base d'une énergie que l'on va tenter de minimiser est la suivante :

On considère un vecteur $(x_p)_{p=1..n}$ de variables à valeur dans un espace fini $\mathcal{L}$ (pour *label*). On cherche alors à minimiser sur l'ensemble des valeurs possibles de x (cet ensemble est fini) une fonction de la forme

$$E(x) = \sum_{i=1}^{p} D_i(x_i) + \sum_{\substack{i=1..n \\ j=1..n}} V_{ij}(x_i, x_j) \tag{2.1}$$

où D et V sont des fonctions arbitraires respectivement de $\mathcal{L}$ dans $\mathbb{R}$ et de $\mathcal{L}^2$ dans $\mathbb{R}$.

Il y aura souvent une variable x_i par pixel. Dans la plupart des cas, D représentera une attache aux données, et V sera un terme de régularité portant sur des pixels voisins. L'énergie deviendra donc :

$$E(x) = \sum_{i=1}^{p} D_i(x_i) + \sum_{\substack{i=1..n \\ j \in N_i}} V_{ij}(x_i, x_j) \tag{2.2}$$

où N_i désigne l'ensemble des voisins du pixel i.

Il y a deux classes de problèmes très importantes avec ce formalisme :

FIG. 2.1 – Patrick et Thomas en niveaux de griscourtesy of Thomas Deneux

FIG. 2.2 – Patrick et Thomas bruités

2.1.1 Problème binaire

Dans le problème binaire, qui est le mieux étudié théoriquement, les variables x_i ne peuvent prendre que 2 valeurs. D'un point de vue théorique, on peut se ramener au cas où ces valeurs sont 0 et 1. Ce problème n'est utile en soit que pour des applications à la restauration binaire et pour à la segmentation.

Voyons par exemple comme il se formule dans le cas de la restauration binaire. Supposons que nous ayons une image 2D I en niveaux de gris (à valeurs dans $[0, 1]$) - par exemple 2.1, et que cette image soit bruitée (2.2). Supposons de plus que l'on veuille obtenir une «approximation» de cette image en noir et blanc strict (2.3) à partir de l'image bruitée. Un simple seuillage donnerait des résultats peu convaincants (2.4).

On peut plutôt penser à minimiser l'énergie suivante :

$$E(x) = \sum_{i=1}^{p} D_i(x_i) + \alpha \sum_{\substack{i=1..n \\ j \in N_i}} V_{ij}(x_i, x_j) \tag{2.3}$$

avec

$$\begin{cases} D_i(0) = I_i \\ D_i(1) = 1 - I_i \end{cases}$$

FIG. 2.3 – Patrick et Thomas en noir et blanc à partir de l'image originale.

FIG. 2.4 – Patrick et Thomas en noir et blanc par seuillage à partir de l'image bruitée.

et

$$V_{ij} = \delta_{ij}$$

Le premier terme de l'énergie est un terme d'*attache aux données*, qui incite le résultat à être proche des données initiales. Le second est un terme de *régularité* qui incite les pixels voisins à prendre les mêmes valeurs. Le paramètre positif α règle le poids relatif de ces 2 termes, et donc la régularité de la solution obtenue.

(Notons qu'un simple seuillage reviendrait à ne conserver que le terme d'attache aux données, sans le terme de régularité.)

Des résultats optimaux sont donnés pour différentes valeurs de α figures 2.5, 2.6, 2.7 et 2.8.

FIG. 2.5 – Restauration binaire par *Graph Cuts* : $\alpha = 0.2$

FIG. 2.6 – Restauration binaire par *Graph Cuts* : $\alpha = 0.3$

FIG. 2.7 – Restauration binaire par *Graph Cuts* : $\alpha = 0.4$

FIG. 2.8 – Restauration binaire par *Graph Cuts* : $\alpha = 0.5$

FIG. 2.9 – Image bruitée

FIG. 2.10 – Restauration 4 niveaux de gris

FIG. 2.11 – Restauration 16 niveaux de gris

FIG. 2.12 – Restauration 64 niveaux de gris

Comme nous allons le voir, les *Graph Cuts* permettent de trouver un optimum global de cette énergie.

Pourquoi cette énergie ? Le choix de cette énergie pour la résolution de ce problème, de même que toutes les énergies que nous considérerons par la suite, peuvent se justifier proprement dans un cadre bayésien. On peut en effet écrire à une constante de normalisation près $P(X|I) = P(I|X)P(X)$, soit en passant au logarithme, $log(P(X|I)) = log(P(I|X)) + log(P(X))$. Le premier terme correspond au premier terme de notre énergie, soit la probabilité d'observer I connaissant X, et le second correspond à un *a priori* sur la forme de X (ici une cohérence spatiale).

2.1.2 Étiquetage général

Dans le problème général, $\mathcal{L}$ peut avoir un nombre arbitraire de valeurs. C'est évidémment le formalisme le plus utile en pratique. Les *Graph Cuts* fournissent plus rarement des minimums globaux, quoiqu'on puisse en obtenir d'excellentes approximations.

Un exemple de restauration 16 niveau de gris d'une image bruitée est présentée figures 2.9, 2.10, 2.11 et 2.12.

Ce type d'énergies à des liens très forts avec les *méthodes graphiques*, sur lesquelles je reviendrai en dernière partie.

2.1.3 Pas de chance...

Théorème 2.1 *Dans le cas général, minimiser*

$$E(x) = \sum_{i=1}^{p} D_i(x_i) + \sum_{\substack{i=1..n \\ j \in N_i}} V_{ij}(x_i, x_j) \tag{2.4}$$

pour une des deux conditions ci-dessus est **NP***-complet.*

Ceci se prouve simplement par réduction au problème qui consiste à trouver un sous-ensemble indépendant de $\mathbf{V}$ de cardinalité maximale[39].

Mais pas de panique, de nombreux cas utiles restent parfaitement accessibles.

2.2 Résolution par *Graph Cuts*

2.2.1 Comment faire

Le cas binaire

Je m'inspire ici de [39], qui est un excellent article synthétique sur ce que l'on peut faire ou pas dans ce cas.

Il existe d'autres constructions de graphes possibles, mais celle qui est présentée à le mérite d'être systématique, générale et simple. En particulier, un des articles fondateurs de l'utilisation des *Graph Cuts* [8] a raté cette construction, et en a proposé une autre avec des nœuds supplémentaires.

Toute l'idée des *Graph Cuts* est de ramener notre problème de minimisation d'énergie à un problème de coupe minimale dans un graphe.

Ce graphe va être constitué d'un sommet par variable de l'énergie, plus une source et un puits, qui vont représenter en quelque sorte les valeurs possibles de chaque variable (conventionnellement, on choisit 0 pour la source et 1 pour le puits).

Nous allons commencer par analyser des cas très simples.

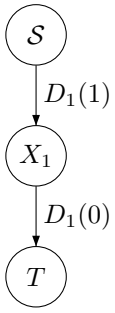
Une variable Soit à minimiser l'énergie

$$D_1(x_1) \tag{2.5}$$

Si $D_1(0) < D_1(1)$, la solution optimale est bien entendu $x_1 = 0$ et réciproquement.

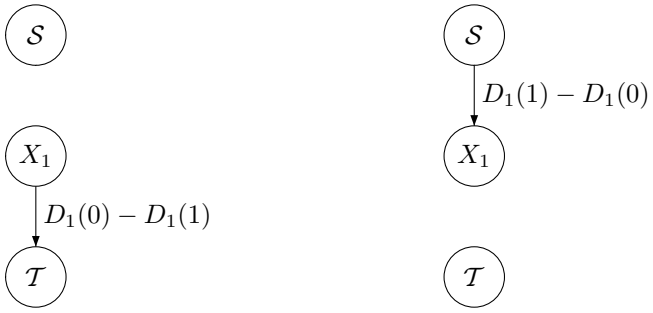
On construit le graphe de la façon suivante :

- $\mathbf{V} = \{\mathcal{S}, \mathcal{T}, X_1\}$
- $\mathbf{E} = \{s_1 = (\mathcal{S}, X_1), t_1 = (X_1, \mathcal{T})\}$
- $w(s_1) = D_1(1), w(t_1) = D_1(0)$



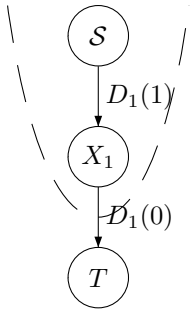
Nous voyons ici apparaître un problème. En effet, les poids des arêtes doivent être *positifs* pour que l'on puisse appliquer les algorithmes de maximisation du flot. Or rien ne garantit la positivité de $D_1(1)$ et $D_1(0)$. On remarque alors que notre problème de minimisation est équivalent au même problème auquel on a ajouté une constante arbitraire. En particulier, si on définit $D'_1(0) = D_1(0) + K$ et $D'_1(1) = D_1(1) + K$, notre problème équivaut à minimiser l'énergie $D'_1(x_1)$. Nous nous sommes donc bien ramenés au cas où les arêtes sont de poids positifs.

On pourrait simplifier encore plus - et c'est ce qu'on fera dans la pratique - en soustrayant la plus petite des 2 valeurs aux 2, ce qui élimine une arête du graphe.

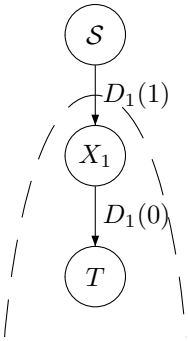


À chaque coupe du graphe correspond de manière univoque une configuration de X :

– la coupe $(\{\mathcal{S}, X_1\}, \{\mathcal{T}\})$ correspond à $x_1 = 0$.



– La coupe $(\{\mathcal{S}\}, \{X_1, \mathcal{T}\})$ correspond à $x_1 = 1$.



et les poids du graphe ont été ajustés de manière à ce que le poids de la coupe soit exactement l'énergie de la configuration associée. Dans cet exemple trivial, on a donc ramené notre problème de minimisation d'énergie à un problème de coupe minimale dans un graphe.

n variables sans interaction Ce cas n'est pas encore particulièrement palpitant, mais il aide à comprendre les choses.

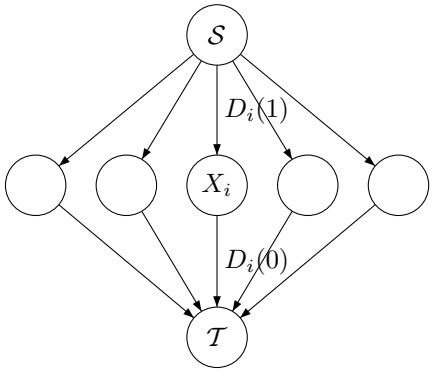
Soit à minimiser l'énergie

$$\sum_{i=1}^{n} D_i(x_i) \tag{2.6}$$

Vu l'indépendance des variables, la solution optimale est bien entendu $x_i = 0$ si $D_i(0) \leq D_i(1)$ et réciproquement.

On construit alors le graphe de la façon suivante :

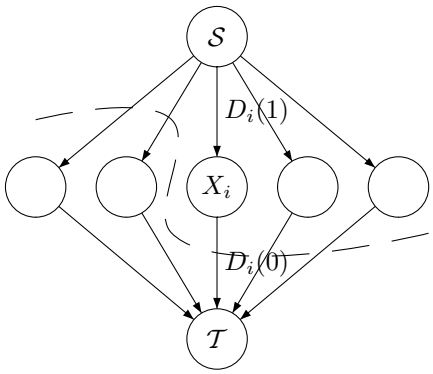
– $\mathbf{V} = \{\mathcal{S}, \mathcal{T}\} \cup \{X_i\}_{i=1..n}$
– $\mathbf{E} = \cup_{i=1..n}\{s_i = (\mathcal{S}, X_i), t_i = (X_i, \mathcal{T})\}$
– $w(s_i) = D_i(1), w(t_i) = D_i(0)$ (après application de la technique mentionnée au paragraphe précédent pour empécher les arêtes négatives).



De même que précédemment à chaque coupe du graphe correspond de manière univoque une configuration de X : $X_i \in \mathbf{S}$ si et seulement si $x_i = 0$.

Par exemple, la coupe suivante correspond à

$$(X_1 = X_2 = 1, X_3 = X_4 = X_5 = 0)$$



On a encore une fois l'égalité de la valeur de la coupe et de l'énergie.

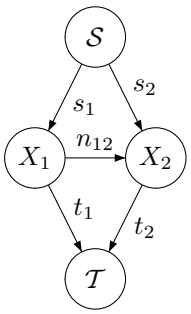
Deux variables avec interaction Nous allons essayer de construire un graphe qui ait la même allure que les graphes donnés ci-dessus.

Considérons donc une énergie à minimiser :

$$D_1(x_1) + D_2(x_2) + V_{12}(x_1, x_2) \tag{2.7}$$

Nous allons choisir la forme suivante pour le graphe :

- $\mathbf{V} = \{\mathcal{S}, \mathcal{T}, X_1, X_2\}$
- $\mathbf{E} = \cup_{i=1..2}\{s_i = (\mathcal{S}, X_i), t_i = (X_i, \mathcal{T})\} \cup \{n_{12} = (X_1, X_2)\}$



Reste à fixer les poids des arêtes tels que la valeur de la coupe soit égale à l'énergie de la configuration associée.

Ceci impose un certain nombre de relations :

$$\begin{cases} x_1 = 0 & x_2 = 0 & \rightarrow & t1 + t2 & = & D_1(0) + D_2(0) + V_{12}(0,0) \\ x_1 = 1 & x_2 = 0 & \rightarrow & s1 + t2 & = & D_1(1) + D_2(1) + V_{12}(1,0) \\ x_1 = 0 & x_2 = 1 & \rightarrow & t1 + s2 + n & = & D_1(0) + D_2(1) + V_{12}(0,1) \\ x_1 = 1 & x_2 = 1 & \rightarrow & s1 + s2 & = & D_1(1) + D_2(1) + V_{12}(1,1) \end{cases}$$

On peut retourner tout ceci dans le sens que l'on veut, ajouter des constantes aux termes qui ne dépendent que d'une seule variable, on aboutit toujours au fait suivant :

$$n = V_{12}(1,0) + V_{12}(0,1) - V_{12}(0,0) - V_{12}(1,1) \tag{2.8}$$

Or, il faut que n soit positif pour que l'on puisse appliquer nos algorithmes. On voit donc apparaitre une condition sur nos termes qui dépendent de deux variables. On pourrait penser que ceci est du à la forme du graphe que nous avons choisie. Or il n'en est rien :

Théorème 2.2 *[39] Toute énergie de la forme* (2.5) *telle qu'il existe* $i \in [1..n], j \in N_i$ *tels que* $V_{12}(1,0) + V_{12}(0,1) < V_{12}(0,0) + V_{12}(1,1)$ *n'est pas minimisable par* Graph Cuts.

Définition 2.1 *On dit que* V_{ij} *est* régulière *(ou* sous-modulaire*) si on a* $V_{12}(0,0) + V_{12}(1,1) < V_{12}(0,1) + V_{12}(1,0)$.

L'application des *Graph Cuts* à notre fonction de deux variables est donc sujette à la sous-modularité de notre énergie.

Cas général On voit très simplement [39] que l'on peut fusionner de tels graphes dans le cas où on a plus de variables, en sommant les capacités des arêtes communes.

On aboutit alors au théorème suivant :

Théorème 2.3 (2.5) *est minimisable par* Graph Cuts *si et seulement si chacun de ses termes dépendant de deux variables est sous-modulaire. Si c'est le cas, trouver une coupe minimale dans le graphe construit ci-dessus permet d'obtenir un minimum global de l'énergie.*

Remarquons que si V_{ij} est un terme de voisinage, on aura en général $V_{ij}(0,0) = V_{ij}(1,1) = 0$. Le terme correspondant sera sous-modulaire.

Interaction à 3 variables .

[39] propose une condition suffisante de minimisabilité par *Graph Cuts* d'une énergie de la forme :

$$E(x) = \sum_{i=1}^{p} D_i(x_i) + \sum_{\substack{i=1..n \\ j=1..n}} V_{ij}(x_i, x_j) + \sum_{\substack{i=1..n \\ j=1..n \\ k=1..n}} V_{ijk}(x_i, x_j, x_j) \qquad (2.9)$$

qui est en fait une condition de sous-modularité de chaque projection par rapport à une des variables des V_{ijk}.

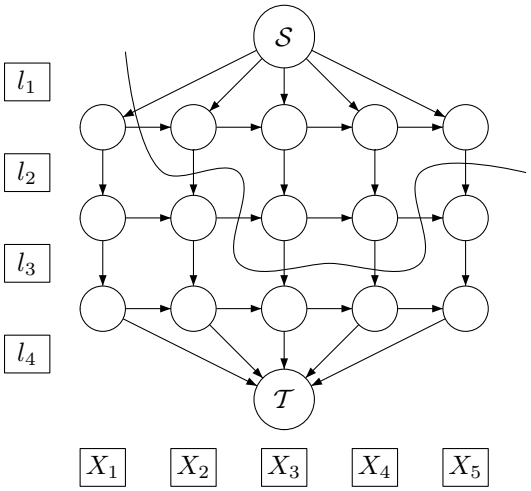
Les applications de ce formalisme sont beaucoup moins fréquentes dans la pratique.

Interaction à plus de variables .

Dans [18], Freedman et Drineas prennent un point de vue complètement algébrique et redémontrent partiellement les résultats vus ci-dessus. Ils trouvent également des conditions (pour l'instant peu exploitables) pour des interactions à k variables.

Le cas multi-étiquette

Le cas où $|\mathcal{L}| > 2$ est évidemment moins favorable. La construction du graphe est la suivante :

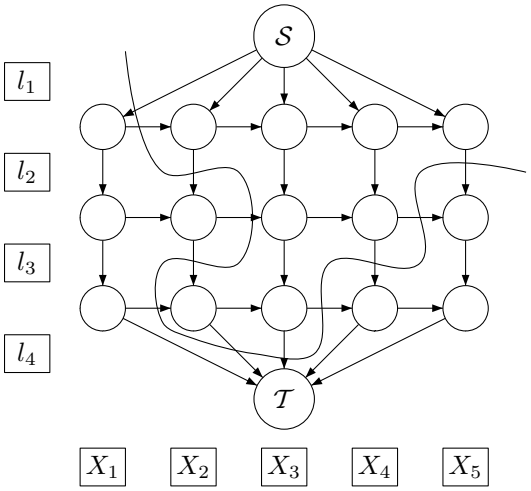


Chaque «colonne» de sommets représente une variable. L'endroit auquel la colonne est coupée indique quelle étiquette va être prise par la variable en question.

Dans cet exemple, on a donc

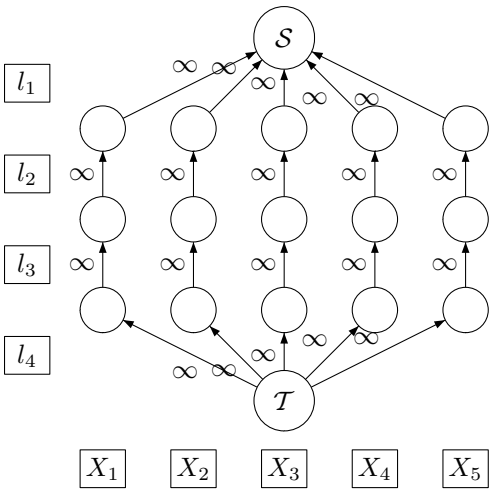
$$(X_1 = l_1, X_2 = l_2, X_3 = l_3, X_4 = l_3, X_5 = l_2)$$

Il y a cependant une subtilité par rapport au cas binaire : il faut garantir que chaque colonne n'est coupée qu'en un seul point. En effet, on pourrait fort bien imaginer la coupe suivante :



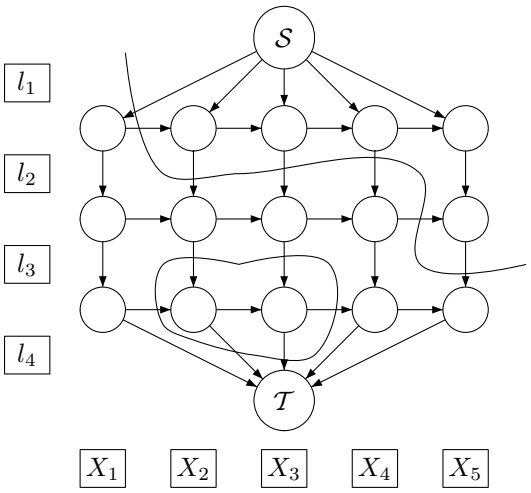
dans laquelle la colonne 2 est coupée deux fois.

Nous allons proscrire ce genre de coupes en rajoutant des liens infinis.



Ces arêtes vont empêcher la coupe de traverser une colonne de sommets de droite à gauche, garantissant ainsi l'unicité de la valeur par colonne.

Un autre type de coupe aurait ici pu intervenir :



En effet, rien ne garantit a priori qu'une coupe ait une seule composante connexe. Les liens infinis empêchent évidemment aussi ce genre de comportement. Mais dans le type de graphe que nous considérons, la minimalité de la coupe minimale n'autorise pas ce genre de comportement : si une coupe contient une composante connexe qui n'est pas reliée à $\mathcal{S}$ ou $\mathcal{T}$, il suffit de la supprimer pour obtenir une coupe plus petite.

Voyons maintenant quelles contraintes impose la positivité des arêtes.

On a le résultat suivant

Théorème 2.4 (Ishikawa[29]) *Supposons*

$$V_{ij}(l_i, l_j) = g(i - j)$$

Alors (2.5) *est minimisable par* Graph Cuts *si et seulement si g est convexe.*

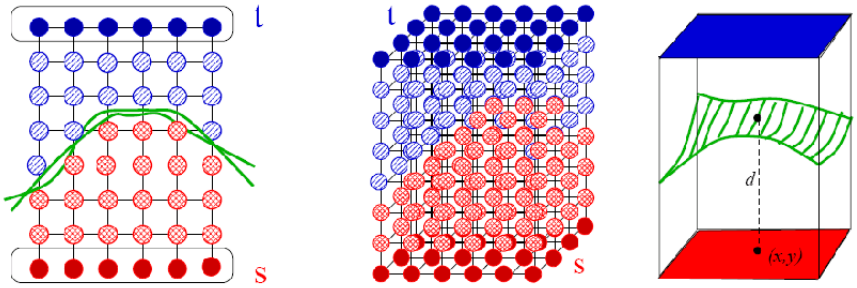
C'est un cas particulier important, qui va trouver de nombreuses applications en pratique (par exemple si $\mathcal{L}$ représente un ensemble de profondeurs pour un problème de stéréo).

En revanche, dès qu'il n'y a plus de notion d'ordonnancement des étiquettes, ou dès que l'on utilise des termes d'interaction qui préservent les discontinuités (c'est à dire qui n'handicapent pas les discontinuités d'intensité grandes par rapport aux petites - on n'a donc plus une fonction convexe), tels

- $V_{ij}(l_i, l_j) = \delta(l_i - l_j)$ (interaction de Potts)
- $V_{ij}(l_i, l_j) = max(|l_i - l_j|, k)$ (linéaire tronqué)

on ne peut plus utiliser cette forme de graphe. Nous allons voir par la suite quoi faire dans ces cas là.

Interprétation géométrique . On peut également donner une interprétation géométrique intéressante aux coupes dans ce type de graphe. Si les variables représentent un pixel chacune, que les termes V_{ij} sont de vrais termes de voisinage entre pixels voisins, et que les étiqettes représentent par exemple une hauteur ou une profondeur, notre graphe - qui est pour l'instant une construction purement théorique - prend une dimension géométrique évidente : on peut voir la coupe comme étant une hypersurface. Dans les graphes $2D$ que nous avons dessinés, la coupe correspond donc à une courbe $1D$. Ci-dessous, le cas $3D$ est illustré (noter le formalisme à plusieurs sources et plusieurs puits, qui est strictement équivalent à celui que nous avons décrit) :



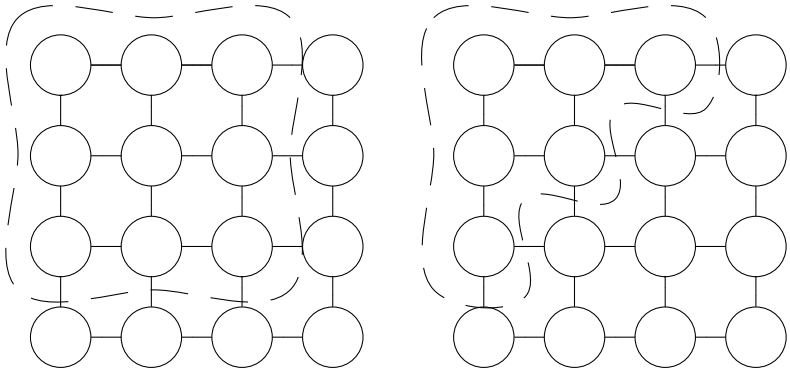
(a) a cut on a 2D grid (b) a cut and a separating hypersurface in 3D

2.2.2 Aspects géométriques

Replaçons nous dans un cas de figure de la la restauration 2D que nous avons déjà évoqué. Supposons que nous utilisions comme voisinage d'un pixel ses 4 plus proches voisins.

Nous sommes dans un cas ou le graphe représente directement le plan de l'image.

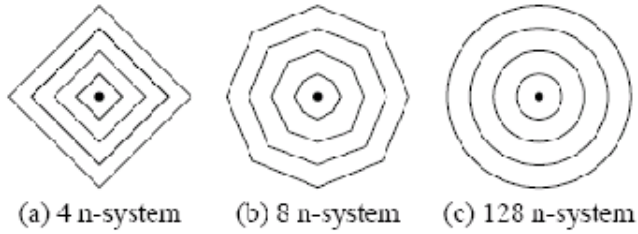
Considérons les deux coupes suivante ($\mathcal{S}$ et $\mathcal{T}$ ne sont pas représentés. Ils sont au-dessus et en-dessous du plan du dessin) :

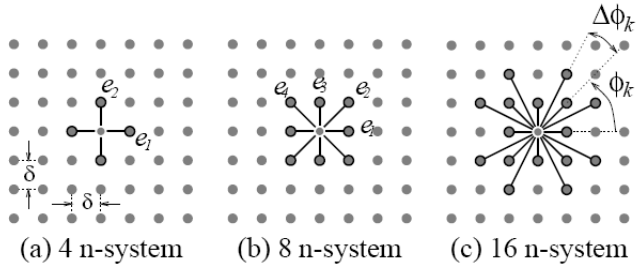


Si l'on fait abstraction des termes d'attache aux données et qu'on suppose que tous les termes de voisinages sont égaux ($V_{ij} = \delta_{i,j}$ par exemple), on voit que ces deux coupes ont la même valeur. Notre choix de voisinage dans la discrétisation peut induire un phénomène de crénelage anisotropique peu souhaitable - en fait, la discrétisation approche l'espace de l'image muni d'une norme $||.||_\infty$.

Est-il possible d'approcher plutôt une norme $||.||_2$?

La réponse est apportée par [6] qui montre comment trouver un système de viosinage et les poids des termes de voisinages pour approcher la norme $||.||_2$, mais aussi des métriques plus générales. Ceci se généralise en n'importe quelle dimension et permet de calculer des géodésiques globalement minimales pour n'importe quelle métrique Riemannienne.





2.2.3 Repousser les limites

Récapitulons les restrictions pour pouvoir trouver un minimum via *Graph Cuts*.

– dans le cas binaire, il faut et il suffit que l'ensemble des termes dépendant de 2 variables soient *sous-modulaires*.

– dans le cas multi-étiquette, il suffit qu'il existe une notion d'ordre sur les étiquettes, et que de plus les terme V_{ij} vérifient une relation de convexité.

Nous allons maintenant voir comment nous faire lorsque ces conditions ne sont pas réalisées.

Cas binaire

L'idée est de construire une énergie sous-modulaire proche de celle que l'on veut minimiser.

Plusieurs possibilités ont été essayées. On peut simplement supprimer les termes non-sous-modulaires [49]. Un autre schéma moins violent a été proposé [48].

Cas multi-étiquette

L'idée est de se ramener itérativement au cas binaire.

Il y a principalement deux types d'algorithmes :

(α, β)-swap. α et β sont 2 étiquettes. À chaque étape, on ne considère que les variables ayant les étiquettes α ou β et on change leur valeur en α ou β uniquement. On applique donc l'algorithme du cas binaire $|\mathcal{L}|^2$ fois pour faire une série d'(α, β)-swap complète (algorithme 3, figure 2.13).

Soit à minimiser une énergie $E(x)$, $(x_i)_i \in \mathcal{L}^n$

Algorithme 3 (α, β)-swap

Initialisation:
initialiser x.
tantque l'algorithme n'est pas stabilisé **faire**
  pour tout couple d'étiquette (α, β) **faire**
    soit $I = \{i \mid x_i = \alpha \text{ ou } \beta\}$
    appliquer une minimisation d'énergie binaire à x_I pour les étiquettes α et β
  fin pour
fin tantque

α-expansion. À chaque étape, on choisit une étiquette α puis on ramène au cas binaire de la façon suivante : on construit $\mathcal{L}' = \{\alpha, I\}$. Tous les pixels interviennent dans l'α-expansion : donner l'étiquette I à un pixel signifie que sa valeur ne change pas, et lui donner l'étiquette α signifie que sa valeur passe à α. À chaque étape, les seuls pixels qui changent prennent la nouvelle étiquette α. Il faut $|\mathcal{L}|$ étapes pour parcourir l'ensemble des étiquettes (algorithme 4).

En pratique, dans les deux cas, on applique la séquence complète jusqu'à ce qu'aucune transformation ne change la configuration actuelle(2.14).

FIG. 2.13 – Deux premières passes d'(α, β)-swap pour le débruitage 4 niveaux de gris. En haut à gauche : image originale, puis l'initialisation et les (α, β) successifs.

Algorithme 4 α-expansion

Initialisation:
initialiser x.
tantque l'algorithme n'est pas stabilisé **faire**
  pour tout couple d'étiquette **faire**
    appliquer une minimisation d'énergie binaire à x pour les étiquettes α et I = rester à la même valeur
  fin pour
fin tantque

On est alors dans un minimum local d'énergie (on ne peut espérer atteindre un minimum global en général). Mais local est ici a comprendre dans un sens très large : deux configurations sont voisines si on peut passer de l'une à l'autre par (α, β)-swap (resp. α-expansion).

En pratique ces algorithmes donnent d'excellents résultats sur de nombreux problèmes, et convergent de plus très rapidement : l'essentiel du gain d'énergie est effectué lors des premières itérations.

Les α-expansions forment un formalisme très puissant, qui est le plus utilisé dans les applications en Vision par Ordinateur.

Pour pouvoir appliquer ces algorithmes sous leur forme première, il faut que le problème binaire auquel on se ramène soit sous-modulaire. On a le théorème suivant :

Théorème 2.5 *[8]*

FIG. 2.14 – Quelques α-expansions pour le débruitage 4 niveaux de gris. En haut à gauche : image originale, puis initialisation et les α-expansions successives. Notez l'absence de changement après la première passe.

- *si V est une métrique, on peut appliquer un algorithme d'α-expansions à E.*
- *si V est une semi-métrique (i.e. non nécessairement définie), on peut appliquer un algorithme d'(α, β)-swap à E.*

Ces algorithmes couvrent donc un grand nombre de cas intéressants, y compris celui des métriques qui préservent les discontinuités.

Dans le cas où ces hypothèses ne sont pas vérifiées, on peut toujours appliquer ces algorithmes en prenant des approximations des problèmes binaires auxquels on se ramène. C'est par exemple le cas dans [61], où l'on obtient de plus une garantie que l'α-expansion approchée décroit l'énergie de départ.

Pour les α-expansions appliquées à une métrique, on peut donner une borne sur la qualité du résultat atteint.

Théorème 2.6 *[8]*

Si x^* est la configuration d'énergie minimale, et x' est la configuration obtenue à la fin des α-expansions. On a alors

$$E(x') \leq 2cE(x^*) \tag{2.10}$$

avec

$$c = max_{\substack{p,q,r,s,l_p,l_q,l_r,l_s \\ r \neq s}} \left| \frac{V_{pq}(l_p, l_q)}{V_{rs}(l_r, l_s)} \right|$$

Par exemple, dans le cas d'une interaction de Potts ($V_{ij}(l_i, l_j) = \delta(l_i, l_j)$) (2.10) nous garantit que l'énergie trouvé par α-expansions est au pire deux fois l'énergie du minimum global.

Notons tout de même que ce type de résultat sue l'énergie ne garantit absolument pas une quelconque proximité «géométrique» de la solution trouvée et de la configuration optimale. En pratique, être proche de l'énergie minimale ne garantit pas la qualité de la solution [5].

Chapitre 3

Applications

3.1 Restauration d'images

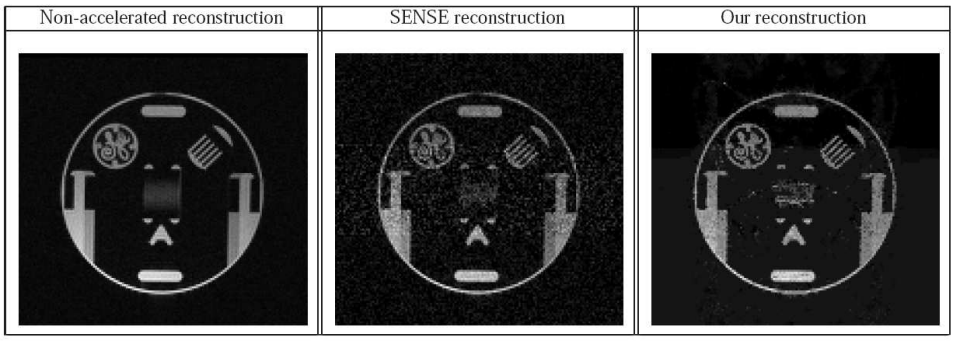
Il s'agit de la première application des *Graph Cuts* en Vision par Ordinateur [27] restée oubliée pendant près de 10 ans. Nous en avons déjà vu le principe dans 2.1.1. Cet article est d'autant plus important historiquement, qu'il a permis pour la première fois de vérifier l'efficacité du très important algorithme de *recuit simulé* [20] qui était la méthode standard pour rédoudre le problème de la *Maximisation A Posteriori* (cet algorithme s'appliquant à n'importe quelle énergie. Il est toutefois peu efficace en pratique cf. 4.3.2).

Une des premières présentations du cas multi-étiquette exact [30] était également appliquée au problème de la restauration d'images.

3.1.1 Déconvolution

Dans [48], les auteurs tentent de déconvoluer une image, qui a par exemple subi un floutage gaussien. Appelons x l'image de départ, H la fonction linéaire de floutage, et y l'image observée. L'idée est alors de retrouver x en minimisant $||y - Hx||_2^2 + V(x)$, où $V(x)$ est un terme de régularité. Cette énergie peut s'exprimer sous la forme de 2.5. Ils appliquent alors un schéma d'α-expansions (2.2.3), avec approximation de l'énergie binaire à chaque étape et obtiennent des conditions pour forcer la décroissance de l'énergie.

Ils appliquent leurs résultats au problème de la recontruction d'images d'IRM.



3.2 Segmentation

3.2.1 Contours Actifs

Les *Contours Actifs* [1] sont la première méthode «énergétique» introduite en vision pour segmenter des objets.

L'idée de base est, considérant une image I et un contour (c'est-à-dire une courbe simple fermée) C, de chercher à minimiser une énergie de la forme

$$E(C) = \int_0^1 \alpha|C_t(v)|^2 + \beta|C_{tt}(v)|^2 + \gamma|\nabla I(C(v))|dv \tag{3.1}$$

Les deux premiers termes imposent une certaine régularité sur l'optimum, alors que le troisième est un terme d'attache aux données, qui pousse la courbe à passer par des points de fort gradient, qui sont présumés être des points candidats appartenant à des contours d'objets. L'importance relative de ces termes est pondérée par les coefficients α, β et γ.

De nombreuses méthodes ont été proposées pour trouver un tel minimum.

Historiquement, la courbe C était paramétrée par des points de contrôle dans la méthode des *snakes* [1], ce qui posait des problèmes liés à la topologie de la courbe, et à la qualité de l'échantillonage.

Une avancée importante a été apportée par les *level sets* [47] [16]. L'idée principale est de représenter C implicitement comme niveau 0 d'une fonction. On s'affranchit ainsi des problèmes de topologie et de paramétrisation, le désavantage étant que l'on doit faire évoluer la fonction partout, et pas seulement sur la courbe (éventuellement sur une bande autour de la courbe, appelée *narrow-band*).

Ces méthodes n'évitent pas l'écueil des minima locaux, et les solutions trouvées vont dépendre de l'initialisation choisie. À noter [32] qui introduit une composante aléatoire pour sortir des minima locaux.

3.2.2 *Graph Cuts* et contours

Les *Graph Cuts* apportent des moyens techniques supplémentaires pour s'attaquer à ces problèmes.

L'idée de départ est la suivante :

- on plaque sur l'image un graphe dont les sommets sont les pixels (ou voxels) de l'image, et dont les arêtes relient les pixels voisins (avec une connectivité à définir, cf. 2.2.2). Les capacités seront d'autant plus faibles que le gradient de l'image sera fort au voisinage de l'arête.
- on place à la main un certain nombre de sources à l'intérieur de l'objet à segmenter, et un certain nombre de puits à l'extérieur (appelées graines).
- on calcule une coupe minimale.

La coupe minimale est alors une hypersurface minimale pour notre problème.

On peut également le reformuler très simplement dans notre formalisme (2.2.1) : on prend un sommet du graphe par pixel (ou voxel), un puits et une

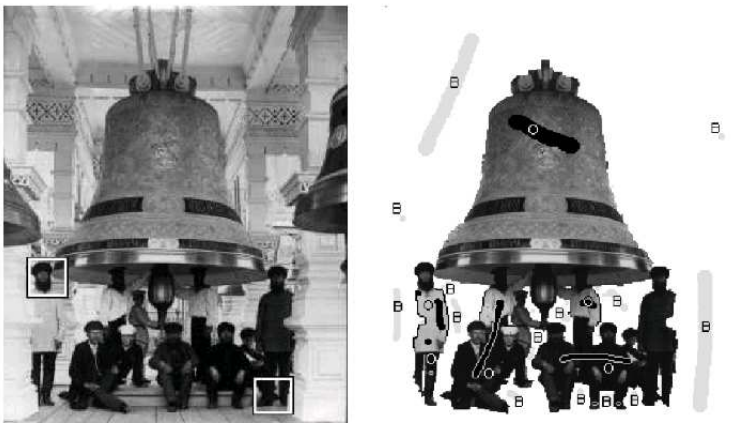
source (les pixels rattachés à la source correspondront à l'objet, et ceux rattachés au puits au fond). Enfin, il suffit de mettre des liens infinis entre la source et les graines de l'objet, et entre le puits et les graines du fond, pour garantir leur placement dans le bon ensemble.

Remarquons que ces réultats sont valables en toute dimension, et qu'il n'y a pas de contrainte de topologie sur les solutions obtenues.

3.2.3 Quelques exemples d'utilisation

Boykov et Jolly [4] utilisent un modèle non-paramétrique : l'objet et le fond sont représentés par des histogrammes de couleurs fixes. Le terme d'attache aux données est déterminée par la probabilité qu'un pixel appartienne à une distribution ou à l'autre.

L'utilisateur dispose également des graines à l'intérieur et à l'extérieur de l'objet. Il s'agit donc de la version de base des algorithmes décrits ci-dessus.



Xu, Bansal et Ahuja [60] : Contours Actifs par *Graph Cuts*.

Même principe que le précédent, sauf que l'utilisateur donne un contour approximatif comme point de départ. Ce contour est étendu en une bande fermée. Son côté intérieur est contraint comme source, et son côté extérieur comme puits. On applique ensuite notre algorithme, ce qui nous garantit un minimum global dans la bande.

Le procédé est itéré jusqu'à convergence.

Lombaert, Sun, Grady et Xu [44] proposent dans le même esprit un algorithme multi-échelle, qui commence par calculer une coupe minimale sur l'image basse résolution. On étend cette solution en une bande à partir de laquelle on applique comme dans l'article précédent les *Graph Cuts*, mais sur l'image a une résolution un peu meilleure. On itère ensuite le procédé jusqu'à remonter à l'image à la résolution originale.

Leur algorithme est significativement plus rapide que celui de [4], pour des résultats similaires.

Rother, Kolmogorov, Blake, Brown et Perez [50][3] : segmentation avec modèles paramétriques du fond et du premier-plan.

L'objet et le fond sont modélisés par des modèles statistiques paramétrés. On commence par minimiser l'énergie par *Graph Cuts*, les paramètres étant fixés (étape M). Puis on réévalue les paramètres des modèles pour la partition obtenue (étape E). On itère le procédé.

Ce type d'algorithmes est très classique en vision. Les *Graph Cuts* sont ici plus rapides que les level-sets, et évitent les minima locaux (à chaque pas M, on a un optimum global).

Juan et Keriven [33] : idem avec du α-matting.

On a ici un modèle paramétrique pour l'intérieur, l'extérieur, et pour la zone ou les 2 objets «cohabitent» par transparence, ce dernier modèle étant un mélange des 2 premiers.

Freedman et Zhang [19] proposent de segmenter des objets en utilisant des *a priori* de forme. Ils semblent obtenir des résultats meilleurs avec *a priori* sur certains problèmes, mais leur heuristique ne permet de traiter que des translations et dilatations du modèle de forme *a priori*.

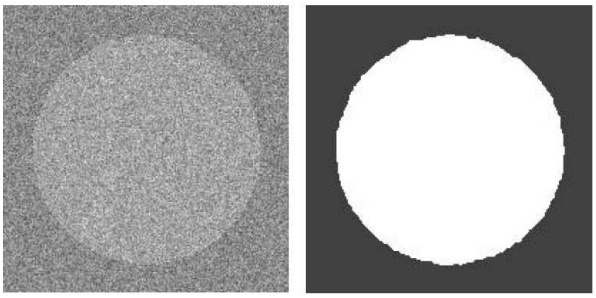
Zabih et Kolomogorov [61]

proposent un algorithme de type *Expectation-Minimization* (EM) pour segmenter un objet : il se basent sur l'approche classique en segmentation qui consiste à se placer dans un espace de «caractéristiques» (*feature space*) - la couleur de chaque pixel par exemple - de «clusteriser» les points obtenus à partir de l'espace de caractéristiques, c'est-à-dire de les regrouper en un certain nombre souvent prédéfini de «paquets», et d'attribuer à chaque pixel de l'image d'origine une étiquette correspondant au paquet dans lequel tombe son point associé dans l'espace des caractéristiques.

Le problème de ces approches est qu'elles ne prennent pas en compte la notion de voisinage dans l'image de départ. [61] propose donc un algorithme itératif avec 2 pas :

E : les paquets étant fixé, on applique un algorithme de *Graph Cuts* pour classer correctement les pixels en fonction de l'étiquette de leur image et des étiquettes des voisins.

M : on fixe l'étiquetage, et on optimise les paramètres de description des paquets.



3.3 Stéréo

3.3.1 Le problème

Le problème de la stéréo est un problème très étudié en vision par ordinateur, qui consiste à retrouver une représentation tridimensionnelle d'un objet ou d'une scène à partir de photos de l'objet prises sous différents angles par des *caméras* calibrées.

Il y a essentiellement 2 approches de ce problème :

Une approche bas-relief (stéréo binoculaire) où l'on fixe une image de référence, et où l'on essaye de calculer la profondeur dans l'espace (ou ce qui revient au même la disparité) de chaque pixel de l'image, en s'aidant des autres images. C'est l'approche utilisée par exemple lorsque l'on a 2 photos d'un même objet prises sous des angles légèrement différents (stéréo binoculaire). On cherche donc à reconstruire une surface (appelée carte de disparité ou de profondeur).

Il y a beaucoup d'approches en programmation dynamique pour résoudre ce problème, ainsi que des *Belief Propagation* (4.3.4).

Une approche haut-relief (reconstruction $3D$) où l'on cherche à reconstruire une surface complète : soit on veut déterminer quels voxels sont occupés

par de la matière, et lesquels ne le sont pas (approche volumique), soit on cherche une surface fermée représentant les bords de l'objets (approche surfacique).

On a typiquement en entrée des photos d'un objet prises sous des angles variés et en nombre plus important.

Une première approximation par excés du volume occupée est parfois donnée par l'enveloppe visuelle, c'est-à-dire l'intersection des cônes englobants l'objet donnés par chaque image.

L'approche classique pour résoudre ce problème est le *space carving* ([41]) qui consiste à partir d'un volume initial (l'enveloppe visuelle) et à le creuser petit à petit. Ce type d'approches souffre de plusieurs inconvénients - notamment du fait qu'un voxel «creusé» ne peut plus être rajouté par la suite.

Meilleures sont les méthodes à base de *level-sets*([15][31]) où l'on fait évoluer une surface fermée (éventuellement initialisée sur l'enveloppe visuelle) de manière à coller aux données.

Dans la plupart des cas, le critère retenu pour placer un voxel ou une surface est la photo-consistance, qui mesure la similarité entre les couleurs ou intensités des projections d'un point de l'espace sur les différentes caméras. Si les informations obtenues ne sont pas consistantes, il est vraisemblable que le point de l'espace soit vide de matière.

Évidemment, ces problèmes sont loin d'être simples à résoudre, et entre-autres problèmes, les *occlusions* (c'est à dire des parties d'objets visibles dans une image mais pas dans une autre) sont souvent gênantes.

3.3.2 Algorithmes utilisant des *Graph Cuts*

Stéréo binoculaire

Roy-Cox [51]. Il s'agit de l'une des premières utilisations des *Graph Cuts* en Vision par Ordinateur. La carte de disparité est retrouvée comme une hypersurface (2.2.1). Les termes d'attache aux données sont dérivés de la photo-consistance, et les termes de voisinage dépendent la régularité de la solution. Les auteurs obtiennent un algorithme efficace, qui tourne empiriquement en un temps $n^{1.2}d^{1.3}$, où n est la taille des images et d est la résolution en disparité.

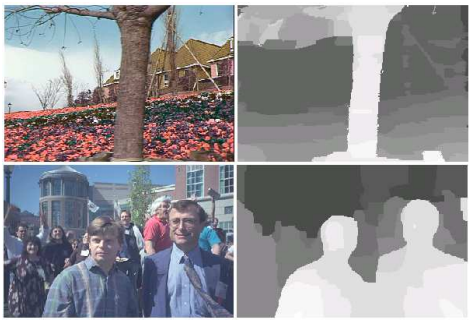
Kim, Kolmogorov et Zabih [35] utilisent la même idée, mais considèrent l'*information mutuelle* plutôt que la photo-consistance pour pouvoir traiter le cas des surfaces *non-lambertienne* (c'est-à-dire dont la radiance «sortante» n'est pas indépendante de l'angle. C'est par exemple le cas s'il y a des reflets). Ils utilisent également un formalisme d'α-expansions (2.2.3), pour avoir un terme de voisinage de type Potts préservant les discontinuités.

Kolmogorov et Zabih [37] proposent un cadre de travail qui va pouvoir s'appliquer a divers domaines. Leur article s'attaque au problème de la mise en correspondance de pixels entre 2 images. Ils mettent en place une énergie générale où les variables vont représenter des couples de pixels, et l'étiquette leur disparité. Leur construction prend en compte les occlusions. Ils utilisent des α-expansions (2.2.3) pour minimiser l'énergie. Ces résultats peuvent être

appliqués à la fois à la stéréo binoculaire et au problème de segmentation du mouvement.

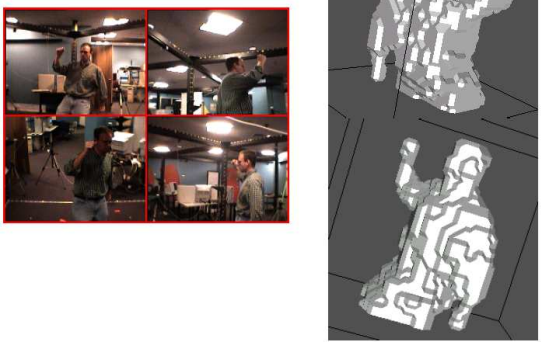
Kolmogorov et Zabih [38] est un formalisme légèrement différent qui permet de mettre en correspondance un nombre arbitraire de caméras. Les occlusions sont toujours prises en compte.

Kolmogorov et Zabih [40] reprend les résultats des 2 articles précédents et les regroupe dans un même formalisme.



Reconstruction 3D

Snow, Viola et Zabih [55] utilisent les *Graph Cuts* pour une approche purement volumique. Chaque voxel de l'espace peut prendre la valeur 1 ou 0 suivant qu'il est occupé ou pas par de la matière. Le terme d'attache aux données est un terme de photo-consistance, et la régularité est liée au nombre de voxels occupés voisins d'un voxel.



Sinha et Pollefeys [54] proposent une formulation beaucoup plus complexe, passant par un graphe $4D$, et qui permet de reconstruire l'objet en prenant l'enveloppe visuelle comme contrainte *dure*. La reconstruction minimise la photo-consistance parmi les reconstructions donnant la bonne enveloppe visuelle.

Vogiatiz, Torr et Cipolla [57]. L'approche est ici surfacique : on cherche à trouver les contours de l'objet 3D. Les auteurs proposent une approche de type *Contours Actifs*. La silhouette donne une approximation de départ, et permet des calculs d'occlusions approchées en chaque point de l'espace. La surface optimiale est trouvée comme coupe minimale dans un voisinage tubulaire intérieur de l'enveloppe visuelle. Une force «ballon» qui tend à gonfler la surface retrouvée évite le biais vers les petites surfaces.

Hornung et Kobbelt [28]. Il s'agit encore d'une approche surfacique, mais où le graphe a un système de voisinage octahédrique.

3.4 Segmentation d'objets en mouvement

Le problème de la segmentation d'objets en mouvement consiste à partir d'une séquence vidéo de retrouver les «couches de même mouvement» (plus simplement les objets en mouvement) dans la scène, c'est à dire de retrouver leur support dans chaque image et d'évaluer les paramètres de leur mouvement.

Une approche consiste à utiliser des algorithmes à deux pas de type EM. L'un des pas va calculer les paramètres du mouvement rendant le mieux compte de la séquence étant donné les supports des couches. L'autre va, à paramètres fixés, trouver la partition optimale des images en supports. Des opérations supplémentaires sont également nécessaires pour fusionner des couches, ou pour découper une couche, leur nombre n'étant pas connu à priori

Dans ce cadre [12] propose un modèle intéressant dans lequel l'étape de partitionnement optimal est effectuée par *Graph Cuts*, dans un formalisme d'α-expansions (2.2.3).

[59] utilise des algorithmes à base de *Graph Cuts* pour trouver des régions «graines» donnant une correspondance fiable entre 2 images (leur formalisme est curieux dans la mesure où ils utilisent une approche de type level sets explicitement destinée ne pas trouver une partition optimale au sens où ils l'entendent). Ils extraient ensuite les couches de mouvement par *Graph Cuts*, en utilisant un formalisme qui leur permet de trouver explicitement les occlusions.

3.5 Synthèse de textures

La synthèse de texture consiste, à partir d'une petite image, à recréer une texture complète. Les applications sont essentiellement en synthèse d'image.

[42] propose un algorithme intéressant. Il s'agit d'un algorithme itératif, qui choisit une partie de l'image de départ, et qui essaye de la recoller dans l'image d'arrivée, en respectant au mieux les morceaux déjà collés. Les *Graph Cuts* interviennent pour trouver ce «recollement» optimal.

Le formalisme utilisé est un peu particulier, mais doit pouvoir se reformuler comme des α-expansions (2.2.3).


3.6 Mosaïque digitale

[49]

Le problème de la mosaïque digitale est de créer automatiquement une image «résumé» à partir d'une collection donnée d'image, dans le but par exemple pour un utilisateur particulier de se retrouver dans sa base de photos de vacances.

Le problème est reformulé de la façon suivante : l'image d'arrivée est découpée en blocs, qui vont être nos variables. L'étiquette de chaque blos indique le numéro de l'image de départ dans lequel on va chercher le bloc de pixels à recopier, ainsi que la position de ce bloc de pixel. Les termes de voisinages vont servir à encoder le fait que l'on souhaite recopier non pas des blocs de façon incohérente, mais que l'on voudrait recopier plutôt des blocs déjà contigüs dans les images de départ. Ils vont également empêcher que l'on aille chercher le même bloc 2 fois. Les termes dépendant d'une seule variable vont permettre de choisir préférentiellement les blocs «saillants» dans les images de départ.

L'algorithme de résolution proposé est un algorithme d'α-expansions. Ils donnent cependant 2 extensions. La première permet d'étendre l'algorithme dans le cas où il y a des contraintes dures (certaines valeur de V à ∞). La seconde propose, dans un cadre d'α-expansions, une façon d'approcher l'énergie non-sous-modulaire par une énergie sous-modulaire à laquelle on applique l'α-expansion avec garantie de décroissance de l'énergie de départ.

Chapitre 4

Alternatives discrètes aux *Graph Cuts* en Vision par Ordinateur

Dans cette section, je vais évoquer quelques alternatives possibles aux graphcuts pour minimiser les énergies. Le propos de cette section n'est pas du tout de détailler ces méthodes, mais plutôt de donner les idées et des pointeurs vers la littérature.

4.1 Normalized Cuts

Les *Normalised Cuts* ([53]) sont une méthode basée sur la théorie des graphes pour faire du clustering et de la segmentation.

L'idée générale est d'associer un sommet du graphe à chaque pixel, et de mettre des arêtes entre tous les pixels, les poids étant d'autant plus grands que les pixels sont semblables.

On cherche alors à trouver une partition (A, B) de ce graphe optimisant une quantité qui assure à la fois la cohérence des pixels de A, celle des pixels de B et la dissimilarité des pixels de A par rapport à ceux de B, tout en évitant le biais vers les petites coupes.

Il s'agit d'un problème de coupe minimale dans un graphe qui n'a pas de source ou de puits.

La recherche de la solution optimale se reformule comme la recherche de vecteurs propres d'une matrice associée au graphe.

Voir également [26].

4.2 Marches Aléatoires

[25] [24]

Les *Marches Aléatoires* sont une famille d'algorithmes pour segmenter une image ou un volume en n parties. L'idée est d'associer un graphe à l'image, où l'on place un sommet en chaque pixel et où les sommets associés à des pixels

voisins sont reliés par des arêtes, de poids d'autant plus grand que les pixels sont similaires.

L'utilisateur place alors des graines à l'intérieurs du ou des objets à segmenter, et à l'extérieur.

Pour déterminer à quel objet un pixel non marqué appartient, on effectue une marche aléatoire sur le graphe depuis le sommet associé au pixel : à chaque sommet, il est d'autant plus probable d'aller vers un autre sommet que l'arête les reliant a une valeur importante. La marche s'arrête lorsqu'on a atteint l'une des graines.

On définit ainsi en chaque point une probabilité d'atteindre chaque graine à l'issue de la marche. Le point sera alors rattaché à la graine ayant la plus grande probabilité d'être atteinte.

Ce problème ce reformule ici aussi en terme d'algèbre linéaire (en l'occurence des résolutions de systèmes linéaires).

À noter [23] qui introduit dans ce formalisme des modèles non-paramétriques.

4.3 Minimisation d'énergies et Champs de Markov

Nous allons ici intéresser aux méthodes basées sur les Champs de Markov. [2]

4.3.1 Reformulation

Considérons notre énergie :

$$E(x) = \sum_{i=1}^{p} D_i(x_i) + \sum_{\substack{i=1..n \\ j \in N_i}} V_{ij}(x_i, x_j) \tag{4.1}$$

sans pour l'instant faire d'hypothèse sur l'ensemble des valeurs prises par x_i.
Passons à l'exponentielle de l'opposé :

$$exp(-E(x)) = \prod_{i=1}^{p} exp(-D_i(x_i)) \prod_{\substack{i=1..n \\ j \in N_i}} exp(-V_{ij}(x_i, x_j)) \tag{4.2}$$

Posons de plus

$$K = \sum_{x} exp(-E(x)) \tag{4.3}$$

et

$$p(x) = \frac{exp(-E(x))}{K} \tag{4.4}$$

de façon à ce que $p(x)$ soit une distribution de probabilité sur l'ensemble des configurations possibles. Cette distribution s'appelle *distribution de Gibbs* de *potentiel* $P = \{exp(-D_i), exp(-V_{ij})\}$. On appelle *système de voisinage* de cette distribution la relation de voisinage $\mathcal{N}$ sur $(X_i)_i$ où $X_p \, \mathcal{N} \, X_q$ si et seulement si X_p et X_q interviennent dans un même élément du potentiel.

Il est alors immédiat que minimiser notre énergie revient à retrouver la configuration la plus probable sous la distribution p.

Or p est loin d'être quelconque, et peut se représenter dans le cadre des *Champs de Markov*.

Définition 4.1 *Soit* $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ *un graphe* non-*orienté. On appelle* Champ de Markov *sur* $\mathbf{G}$ *une famille de variables aléatoires* $(X_i)_{i \in \mathbf{V}}$ *(i.e. il y a une variable aléatoire par sommet du graphe) de loi p vérifiant :*

$$p(X_i|(X_j)_{j \in \mathbf{V}}) = p(X_i|(X_j)_{j \in N_i}) \tag{4.5}$$

où N_i est l'ensemble des nœuds voisins de i dans le graphe.

(4.5) signifie que la valeur du champ en un sommet donné ne dépend *conditionnellement* que de celle des sommets voisins.

Théorème 4.1 (équivalence Markov Gibbs) *[20] Soit X un champ de Gibbs de potentiel P. Si V est un potentiel de voisinage $\mathcal{N}$, alors c'est un champ de Markov pour ce même système de voisinage.*

Ce théorème permet de replacer notre problème de minimisation d'énergie dans le cadre d'un problème probabiliste appelé *Maximisation a Posteriori* (*MAP*) sur un champ de Markov.

Il existe une littérature prolifique sur le sujet. Je vais par la suite citer rapidement les principaux résultats.

Notons que dans ce formalisme, il n'y a pas de différence profonde entre le cas binaire et le cas multi-étiquette.

NB : dans la suite, je confondrai parfois un nœud dans le graphe et la variable aléatoire associée à ce nœud lorsqu'il n'y aura pas de confusion possible.

4.3.2 Recuit simulé

Il s'agit de la première méthode proposé pour la *MAP* en Vision par Ordinateur.

Le *recuit simulé* consiste essentiellement en une marche aléatoire sur l'ensemble des états du champ de Markov. À chaque pas, on choisit une variable aléatoirement, et on modifie son état si celà conduit a une augmentation de sa probablité (ce qui est testable localement) avec une certaine probabilité liée à l'augmentation de la probabilité du champ sous p.

Une telle approche tombant très vite dans des minima locaux très éloignés du minimum global, on introduit un paramètre T de température, qui tend à autoriser des sauts vers des configurations de plus grande énergie. [20] montre que si on choisit une horloge de décroissance de T appropriée, l'algorithme tend en probabilité vers la loi uniforme sur les configurations d'énergie minimale.

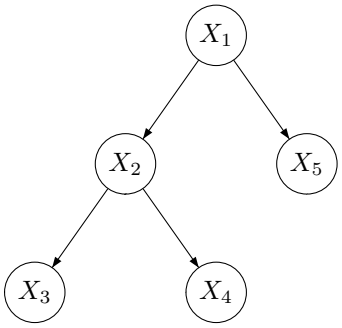
Cependant, l'horloge théorique est très lente et inapplicable en pratique, et on choisit donc des schémas de décroissance de T plus rapides.

Cet algorithme a été supplanté par des méthodes bien plus puissantes que nous allons maintenant décrire.

4.3.3 Belief Propagation

Nous allons commencer par le cas où le graphe associé au Champ de Markov est un graphe sans cycle - c'est à dire un arbre. L'algorithme s'appelle dans ce cas *Belief Propagation*, ou *max-product*.

Plaçons nous par exemple sur l'arbre suivant, enraciné en X_1 :



et écrivons

$$p(x) = \frac{1}{K}\Psi_1(x_1)\Psi_2(x_2)\Psi_3(x_3)\Psi_4(x_4)\Psi_5(x_5)\Psi_{12}(x_1,x_2)\Psi_{15}(x_1,x_5)\Psi_{23}(x_2,x_3)\Psi_{24}(x_2,x_4) \tag{4.6}$$

une distribution de Gibbs générale sur cet arbre.

Probabilité maximale sur un arbre

Supposons pour commencer que nous souhaitions résoudre le problème consistant à calculer $M = max_X p(X)$.

$$M = max_{x_1} max_{x_2} max_{x_3} max_{x_4} max_{x_5} p(x) \tag{4.7}$$

Nous allons réordonner les *max* de manière à éliminer d'abord ceux qui portent sur les feuilles, avant de remonter vers la racine, et faire remonter les termes à maximiser le plus possible dans l'équation :

$$\begin{aligned} M &= \frac{1}{K} max_{x_1} max_{x_2} max_{x_3} max_{x_4} max_{x_5} \Psi_1(x_1)\Psi_2(x_2)\Psi_3(x_3)\Psi_4(x_4)\Psi_5(x_5) \\ &\quad \Psi_{12}(x_1,x_2)\Psi_{15}(x_1,x_5)\Psi_{23}(x_2,x_3)\Psi_{24}(x_2,x_4) \\ &= \frac{1}{K} max_{x_1}\Psi_1(x_1) max_{x_2}\Psi_2(x_2)\Psi_{12}(x_1,x_2) max_{x_3}\Psi_3(x_3)\Psi_{23}(x_2,x_3) \\ &\quad max_{x_4}\Psi_4(x_4)\Psi_{24}(x_2,x_4)(max_{x_5}\Psi_5(x_5)\Psi_{15}(x_1,x_5)) \end{aligned} \tag{4.8}$$

on calcule le dernier maximum en parcourant exhaustivement les valeurs possibles de x_5. Appelons $m_{5\to1}(x_1)$ le résultat (soit $m_{5\to1}(x_1) = max_{x_5}\Psi_5(x_5)\Psi_{15}(x_1,x_5)$). $5 \to 1$ indique que nous sommes en train de maximiser sur x_5 et que le nœud père de x_5 est x_1. Nous verrons dans la section suivante pourquoi une telle notation.

$$M = \frac{1}{K}\, max_{x_1}\Psi_1(x_1)m_{5\to1}(x_1)\; max_{x_2}\Psi_2(x_2)\Psi_{12}(x_1,x_2)max_{x_3}\Psi_3(x_3) \\ \Psi_{23}(x_2,x_3)(max_{x_4}\Psi_4(x_4)\Psi_{24}(x_2,x_4)) \tag{4.9}$$

en maximisant ensuite sur x_4, pour chaque valeur de x_2, on obtient

$$M = \frac{1}{K}\, max_{x_1}\Psi_1(x_1)m_{5\to1}(x_1)\; max_{x_2}\; m_{4\to2}(x_2)\; \Psi_2(x_2)\Psi_{12}(x_1,x_2) \\ (max_{x_3}\Psi_3(x_3)\Psi_{23}(x_2,x_3)) \tag{4.10}$$

puis de même avec x_3

$$M = \frac{1}{K}\, max_{x_1}\Psi_1(x_1)m_{5\to1}(x_1)\; (max_{x_2}\Psi_2(x_2)\Psi_{12}(x_1,x_2)\; m_{4\to2}(x_2)\; m_{3\to2}(x_2)) \tag{4.11}$$

et x_2

$$M = \frac{1}{K}\, max_{x_1}\Psi_1(x_1)m_{5\to1}(x_1)\; m_{2\to1}(x_1) \tag{4.12}$$

Il reste alors à maximiser sur l'ensemble des valeurs de x_1 pour obtenir le résultat.

Au lieu des $O|\mathcal{L}|^{(n)}$ (n étant le nombre de nœuds du graphe) opérations nécessaires pour résoudre ce problème de façon naïve en maximisant sans réordonner les termes, et donc en parcourant l'ensemble des configurations possibles, nous nous sommes ramenés à $O|\mathcal{L}|^2$.

Notons que ce type de résultats s'étend à des graphes plus généraux que des arbres, avec tout de même une plus grande complexité.

MAP sur un arbre

On veut maintenant trouver une configuration atteignant la valeur M, c'est à dire de probabilité maximale.

Reprenons la dernière étape du calcul de M (4.12). Pour calculer la valeur de M à partir de cette équation, on doit parcourir exhaustivement l'ensemble des valeurs possibles de x_1. En particulier, on obtient donc comme effet de bord de l'algorithme la valeur de x_1 dans une configuration de probabilité maximale.

On pourrait alors songer à appliquer l'algorithme à l'arbre n fois en choisissant à chaque fois un nouveau nœud comme racine. Cette idée va en fait échouer s'il n'y a pas unicité de la configuration maximale.

Mais on peut en fait s'en sortir très simplement à partir de notre algorithme de départ : il suffit à chaque étape maximisation max_{x_i}, de garder une trace de

la valeur de la variable x_i qui maximise notre expression (cette valeur dépend bien entendu de x_j). On notera cette valeur $\delta_i(x_j)$.

Par exemple, à partir de (4.11), on maximise sue x_2. Pour chaque valeur de x_1, on stocke une valeur de x_2 pour laquelle le maximum est atteint dans $\delta_2(x_1)$.

Une fois $\bar{x}_1$, la valeur optimale de x_1 determinée par la dernière étape, on obtient immédiatement $\bar{x}_2 = \delta_2(\bar{x}_1)$, et ainsi de suite.

En stockant les δ_i à chaque étape de l'algorithme précédent, on résout donc le problème de MAP grâce à un algorithme de type programmation dynamique dans un arbre, où l'on redescend de la racine vers les feuilles (4.3.3).

Notons que des algorithmes basés sur le même principe existent pour des problèmes d'*inférence* sur des Champs de Markov, c'est à dire le problème consistant à calculer des probabilités maximales ou conditionnelle.

Algorithme 5 Algorithme Max-Product sur un arbre

Entrées: (A, X, p) un Champ de Markov sur un arbre de potentiel Ψ, π la fonction qui à un nœud associe son père, et f la fonction qui a un nœud envoie l'ensemble de ses fils. Nous noterons r la racine.

Sorties: Une configuration de MAP sur (A, X, p)

tantque il reste des nœuds non traités, en choisir un (i) dont tous les fils ont été traités **faire**

Calculer $m_{i \to p(i)}(x_{p(i)}) = max_{x_i} \Psi_i(x_i) \Psi_{p(i)i}(x_{p(i)} x_i) \prod_{j \in f(i)} mj \to i$

Calculer $\delta_i(x_{p(i)}) = argmax_{x_i} \Psi_i(x_i) \Psi_{p(i)i}(x_{p(i)} x_i) \prod_{j \in f(i)} mj \to i$

fin tantque

Poser $\bar{x}_r = \delta_r$

tantque il reste des nœuds non traités, en choisir un (i) dont le père a été traité **faire**

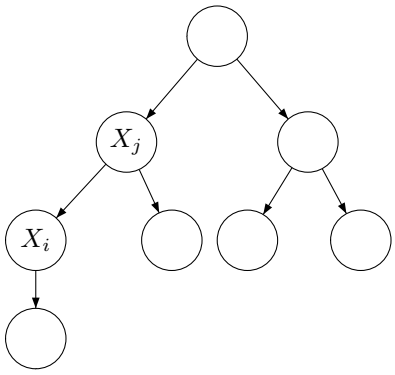
Poser $\bar{x}_i = \delta_i(\bar{x}_{p(i)})$

fin tantque

Renvoyer $\bar{x}$

Reformulation en terme de passage de messages

Considérons 2 nœuds X_i et X_j, où X_i est un fils de X_j.



Au moment où l'on maximise sur j, interviennent uniquement les potentiels dépendant des variables associées aux nœuds descendants de X_j. En bref, $m_{j\to i}$ ne dépend que des variables descendantes de j.

On peut voir $m_{j\to i}$ comme un «message» passant de j vers i (d'où la notation $j \to i$). Dans ce cas, l'algorithme est interprété comme le fait de faire remonter des messages des feuilles vers la racine.

$m_{j\to i}$ peut par ailleurs s'exprimer simplement en fonction des messages passés par les descendants du nœuds j :

$$m_{j\to i}(x_i) = max_{x_j}(\Psi_j(x_j)\Psi_{ij}(x_i,x_j)\prod_{k\in\mathcal{F}_j} m_{k\to j}(x_j)) \tag{4.13}$$

où $\mathcal{F}_j$ est l'ensemble des fils du nœud j.

4.3.4 Loopy Belief Propagation

[17]

L'algorithme de *Loppy Belief Propagation* (LBP), se base sur un l'algorithme de MAP exact décrit dans la section précédente.

L'idée du LBP est, dans le cas où le graphe contient des cycles, de passer ces mêmes messages dans un ordre aléatoire sur les sommets (ou en parallèle).

Algorithme 6 Loopy Belief Propagation

Initialisation:
Initialiser des messages $m^0_{p\to q}(l_q) = 1$ pour tous p et q voisins dans le graphe.

Initialiser t à 0.
tantque $t < T$ **faire**
  $m^t_{j\to i}(x_i) = max_{x_j}(\Psi_j(x_j)\Psi_{ij}(x_i,x_j)\prod_{k\in\mathcal{F}_j} m_{k\to j}(x_j))$
fin tantque
sortir pour tout i $\bar{x}_i = argmax(\Psi_i(x_i)\prod_{j\in\mathcal{N}_i} m_{j\to i}(x_i))$

Avec une condition d'arrêt à déterminer.

On espère alors que les messages vont converger, et que les valeurs des étiquettes données à la fin vont donner une configuration proche du MAP.

4.3.5 Tree-ReWeighted

[58] [45] [56]

L'idée des *Tree-ReWeighted Message Passing* (TRW) est de trouver un ensemble d'arbres couvrants du graphe, et d'appliquer successivement à chacun BP. (On appelle famille d'arbres couvrante $\mathcal{A} = (A_i)_i$ une famille d'arbre telle que toute arête du graphe soit contenues dans au moins un des A_i).

Algorithme 7 TRW

Initialisation:
Touver une famille $\mathcal{A}$ d'arbres couvrante du graphe G.
Initialiser des messages $m^0_{p\to q}(l_q) = 1$ pour tous p et q voisins dans le graphe.

Initialiser t à 0.
tantque $t < T$ **faire**
  tantque on a pas parcouru tous les arbres **faire**
    on choisit un arbre $A_i \in \mathcal{A}$
    on lui applique BP, en ne touchant pas aux messages correspondants à des arêtes non-contenues dans A_i.
  fin tantque
fin tantque
sortir pour tout i $\bar{x}_i = argmax(\Psi_i(x_i) \prod_{j\in\mathcal{N}_i} m_{j\to i}(x_i))$

On espère ici aussi qu'il y aura convergence vers une solution correcte.

Théorie et pratique

En fait, les 2 algorithmes précédents peuvent se replacer dans un cadre plus théorique assez complexe, et peuvent se voir comme des relaxations de reformulation en programmation linéaire de notre problème.

Il n'y a cependant pas de «bonne» propriété de convergence, bien qu'on puisse caractériser les états vers lesquels on converge par une notion de consistance locale ([45] [56]). L'exposé de cette théorie dépasse largement la portée de ce travail.

Les derniers résultats en découlant ont conduit à un algorithme de type TRW où l'on maintient un minorant de l'énergie en permanence, et où la croissance de ce minorant est garantie. On obtient donc une évaluation de la distance à l'énergie minimale.

Les performances de ces algorihmes sont étudiées dans [9]. En pratique, les résultats sont excellents sur des graphes peu-connectés, et les algorithmes semblent mieux se comporter s'il y a des termes de voisinages assez forts. Dans certains cas, TRW se comporte mieux que les *Graph Cuts*. Sur des problèmes complexes, il arrive cependant que ces algorithmes ne convergent pas du tout, renvoyant des résultats totalement inexpoitables.

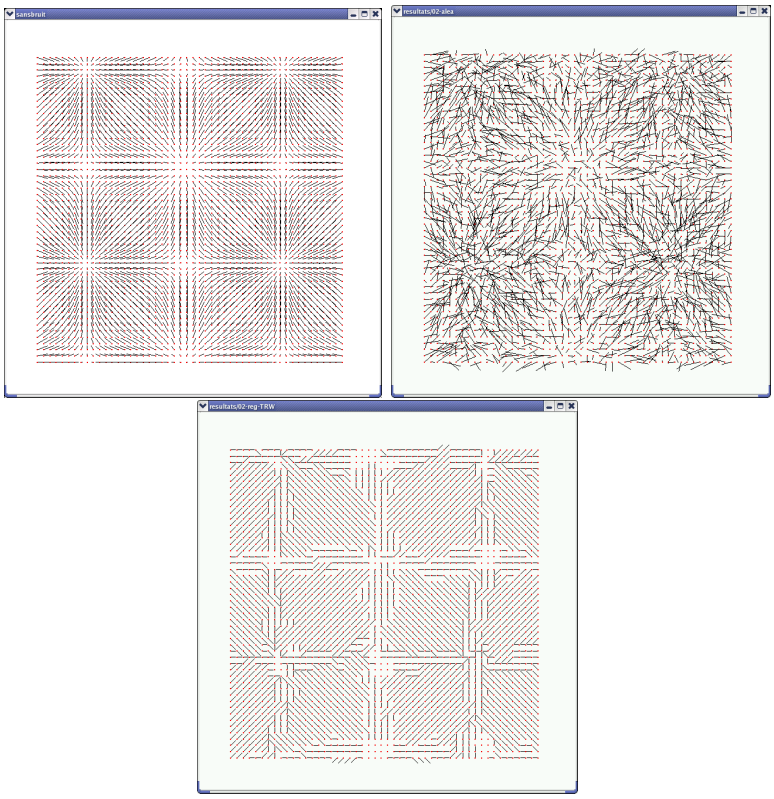


FIG. 4.1 – Régularisation de champs de vecteur par TRW.
1-champ non bruité 2-champ bruité 3-champ débruité par TRW avec 8 orientations possibles

Conclusion

Nous avons vu que les *Graph Cuts* sont un outil très important d'optimisation discrète. De nombreux problèmes de minimisation d'énergie peuvent être résolus globalement grâce à eux. Dans les cas où trouver un minimum global n'est pas possible, nous avons vu qu'existent de nombreux algorithmes efficaces basés sur les *Graph Cuts*. De plus, des pistes très intéressantes sont ouvertes par leur interprétation géométrique.

J'espère avoir convaincu le lecteur que les *Graph Cuts* sont un puissant outil d'optimisation combinatoire, qui trouve de très nombreuses applications en vision par ordinateur. J'espère également avoir donné suffisamment de pistes bibliographiques pour satisfaire la curiosité de toute personne intéressée par le domaine.

Remerciement

Merci à Patrick Labatut ses relectures et remarques constructives sur ce travail !

Bibliographie

[1] Kass M Witkin A and Terzopoulos D. Snakes : Active contour models. First international conference on computer vision, 1987.

[2] B.Chalmond. *Éléments de modélisation pour l'analyse d'images*. Mathématiques & Applications 33. Springer, 2000.

[3] A. Blake, C. Rother, M. Brown, P. Perez, , and P. Toor. Interactive image segmentation using an adaptive gmmrf model. In *Lecture Notes in Computer Science*, volume 3021, pages 428–441, Jan 2004.

[4] Boykov and Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. 2001.

[5] Member-Yuri Boykov and Member-Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(9) :1124–1137, 2004.

[6] Y. Boykov and V. Kolmogorov. Computing geodesics and minimal surfaces via graph cuts, 2003.

[7] Y. Boykov, O. Veksler, and R. Zabih. Markov random fields with efficient approximations. In *CVPR '98 : Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, page 648, Washington, DC, USA, 1998. IEEE Computer Society.

[8] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. In *ICCV (1)*, pages 377–384, 1999.

[9] V.Kolmogorov C.Rother. Comparison of energy minimization algorithms for highly connected graphs. Technical Report MSR-TR-2006-19, 2006.

[10] G. B. Dantzig and D. R. Fulkerson. On the max-flow min-cut theorem of networks. *Ann. Math. Studies*, 38, 1956.

[11] E.A. Dinits. Algorithm for solution of a problem of maximum flow in a network with power estimation. *Doklady Akademii Nauk SSSR*, 194, 1970.

[12] Romain Dupont, Nikos Paragios, Renaud Keriven, and Philippe Fuchs. Spatio-temporal motion estimation & segmentation in layers : Robust estimators, markov random fields, visual grouping and graph-based optimization.

[13] E.A.Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. (11) :1277–1280, 1970.

[14] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2) :248–264, April 1972.

[15] O. Faugeras and R. Keriven. Variational principles, surface evolution, pde's, level set methods and the stereo problem. In *IEEE Trans. Image Processing*, volume 7, pages 336–344, 1998.

[16] Olivier D. Faugeras and Renaud Keriven. Level set methods and the stereo problem. In *Scale-Space Theories in Computer Vision*, pages 272–283, 1997.

[17] P. Felzenszwalb and D. Huttenlocher. Efficient belief propagation for early vision, 2004.

[18] Daniel Freedman and Petros Drineas. Energy minimization via graph cuts : Settling what is possible. In *CVPR*, pages 939–946. IEEE Computer Society, 2005.

[19] Daniel Freedman and Tao Zhang. Interactive graph cut based segmentation with shape priors. In *CVPR (1)*, pages 755–762, 2005.

[20] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Trans. Pattern Anal. Machine Intell.*, 6(6) :721–741, Nov. 1984.

[21] A. V. Goldberg, É. Tardos, , and R. E. Tarjan. Network flow algorithms. Technical report, Mars 1989.

[22] Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4) :921–940, 1988.

[23] Leo Grady. Multilabel random walker image segmentation using prior models. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1 of *CVPR*, pages 763–770, San Diego, June 2005. IEEE, IEEE.

[24] Leo Grady. Random walks for image segmentation. *IEEE trans. on Pattern Analysis and Machine Intelligence*, 2006.

[25] Leo Grady and Gareth Funka-Lea. Multi-label image segmentation for medical applications based on graph-theoretic electrical potentials. In Milan Šonka, Ioannis A. Kakadiaris, and Jan Kybic, editors, *Computer Vision and Mathematical Methods in Medical and Biomedical Image Analysis, ECCV 2004 Workshops CVAMIA and MMBIA*, number LNCS3117 in Lecture Notes in Computer Science, pages 230–245, Prague, Czech Republic, May 2004. Springer.

[26] Leo Grady and Eric L. Schwartz. Isoperimetric graph partitioning for data clustering and image segmentation. *IEEE Pattern Analysis and Machine Intelligence*, 28(3) :469–475, 2006.

[27] D.M. Greig, B.T. Porteous, and A.H. Seheult. Exact maximum a posteriori estimation for binary images. *J. R. Statist. Soc. B*, 51 :271–279, 1989.

[28] Alexander Hornung and Leif Kobbelt. Hierarchical volumetric multi-view stereo reconstruction of manifold surfaces based on dual graph embedding. In *Computer Vision and Pattern Recognition*, 2006.

[29] H. Ishikawa. Exact optimization for markov random fields with convex priors, 2003.

[30] H. Ishikawa and D. Geiger. Mapping image restoration to a graph problem, 1999.

[31] O.Faugeras J.-P.Pons, R.Keriven. Multi-view stereo reconstruction and scene flow estimation with a global image-based matching score. Technical Report RR5321, 2005.

[32] Olivier Juan, Renaud Keriven, and Gheorghe Postelnicu. Stochastic motion and the level set method in computer vision : Stochastic active contours. *Int. J. Comput. Vision*, 69(1) :7–25, 2006.

[33] Juan.O. and Keriven.R. Trimap segmentation for fast and user-friendly alpha matting.

[34] A.V. Karzanov. Determining the maximal flow in network by the method of preflows. *Soviet Math. Dokl.*, 15 :434–437, 1974.

[35] Junhwan Kim, Vladimir Kolmogorov, and Ramin Zabih. Visual correspondence using energy minimization and mutual information. In *ICCV*, pages 1033–1040, 2003.

[36] P. Kohli and P.H.S. Torr. Efficiently solving dynamic markov random fields using graph cuts. 2005.

[37] Vladimir Kolmogorov and Ramin Zabih. Computing visual correspondence with occlusions via graph cuts. In *International Conference on Computer Vision*, volume 2, pages 508– ?, 2001.

[38] Vladimir Kolmogorov and Ramin Zabih. Multi-camera scene reconstruction via graph cuts. In *ECCV (3)*, pages 82–96, 2002.

[39] Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts ? In *ECCV (3)*, pages 65–81, 2002.

[40] Vladimir Kolmogorov, Ramin Zabih, and Steven Gortler. Generalized multi-camera scene reconstruction using graph cuts. In *International Workshop on Energy Minimization Methods*, 2003.

[41] Kiriakos N. Kutulakos and Steven M. Seitz. A theory of shape by space carving. Technical Report TR692, 1998.

[42] V. Kwatra, A. Schodl, I. Essa, and A. Bobick. Graphcut textures : image and video synthesis using graph cuts, 2003.

[43] L.Ford and D.Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[44] Herve Lombaert, Yiyong Sun, Leo Grady, and Chenyang Xu. A multilevel banded graph cuts method for fast image segmentation. In *ICCV*, pages 259–265, 2005.

[45] Wainwright MJ, Jaakkola TS, and AS Willsky. Tree-based reparametrization framework for approximate estimation on graphs with cycles. *IEEE Trans. on Information Theory*, 49(5) :1120–1146, may 2003.

[46] O.Juan and Y.Boykov. *Active Cuts* for real-time graph partitionning in vision. 2006.

[47] Stanley Osher and James A Sethian. Fronts propagating with curvature-dependent speed : Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79 :12–49, 1988.

[48] Ashish Raj and Ramin Zabih. A graph cut algorithm for generalized image deconvolution. In *ICCV '05 : Proceedings of the Tenth IEEE International Conference on Computer Vision*, pages 1048–1054, Washington, DC, USA, 2005. IEEE Computer Society.

[49] C. Rother, S. Kumar, V. Kolmogorov, and A. Blake. Digital tapestry. In *CVPR '98 : Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Washington, DC, USA, 2005. IEEE Computer Society.

[50] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "grabcut" : interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3) :309–314, 2004.

[51] S. Roy and I. Cox. A maximum-flow formulation of the n-camera stereo correspondence problem, 1998.

[52] Alexander Schrijver. *Combinatorial optimization : polyhedra and efficiency.* volume A, paths, flows, matchings, chapter 1-38. Springer, 2003. Schrijver.

[53] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2000.

[54] Sudipta Sinha and Marc Pollefeys. Multi-view reconstruction using photo-consistency and exact silhouette constraints : A maximum-flow formulation. In *International Conference on Computer Vision*, oct 2005.

[55] Dan Snow, Paul Viola, and Ramin Zabih. Exact voxel occupancy with graph cuts. In *Computer Vision and Patern Recognition*, volume 1, 2000.

[56] V.Kolmogorov. Convergent tree-reweighted message passing for energy minimization. Technical Report MSR-TR-2005-38, 2005.

[57] George Vogiatzis, Philip H. S. Torr, and Roberto Cipolla. Multiview stereo via volumetric graph-cuts. In *Computer Vision and Pattern Recognition*, pages 391–398, 2005.

[58] M. Wainwright, T. Jaakkola, and A. Willsky. Map estimation via agreement on (hyper)trees : messagepassing and linear programming approaches, 2002.

[59] J. Xiao and M. Shah. Motion layer extraction in the presence of occlusion using graph cut, 2004.

[60] Ning Xu, Ravi Bansal, and Narendra Ahuja. Object segmentation using graph cuts based active contours. *cvpr*, 02 :46, 2003.

[61] Ramin Zabih and Vladimir Kolmogorov. Spatially coherent clustering using graph cuts. In *CVPR (2)*, pages 437–444, 2004.