- Identification de sous-structures optimales pour les graphes de flots -

JOACHIM Julien - Avril 2022, MP Baimbridge

Repo GitHub du projet, avec l'integralite du code source (2KLOC) :
https://github.com/Soonies/TIPE_graphes_flot_efficaces/tree/master/banana/lib

1. Resume

Ce rapport presente des pistes de recherche pour l'identification de sous-structures optimales de reseaux a flots solutions d'un probleme min-cost. On presente d'abord une structure de donnee efficace pour representer des graphes de flots dynamiques avec des operations pour la plus part en temps $\Theta(1)$. On decrit ensuite certaines sous-structures optimales pour des cas particuliers.

Les sections du rapport ou apparraissent le symboles "☯" ne sont pas des inititatives totalement personelles mais relevent plutot d'applications ou de rappels de resultats et de notions deja classiques dans la litterature

2. Definitions preliminaires et positionnement du probleme

Dans cette partie, on donne les definitions necessaires a la description des objets manipules dans ce projet, ainsi que l'enonce du probleme sur lequel nous nous sommes

concentres.

2.1. Definitions

Definition 1 *Graphe de flot* ☯

Un graphe de flot (aussi appelle « reseau a flot » ou simplement « reseau ») est une variante sur la structure classique de graphe dirige ou l'on assigne aux sommets et aux arcs des attributs particuliers :

- Pour tout sommet v, on se donne un attribut $b(v)$ appelle « production du sommet v » qui represente la quantite de flot que le sommet v fournit au reseau
- A tout arc $e = (i, j)$ on attribue :
  - Un cout $c(i, j)$ qui est une fonction de cout classique (distance, prix, perte, etc...)
  - Un flot $f(i, j)$ qui peut representer une certaine quantite d'information ou de biens passant par l'arc
  - et une capacite $u(i, j) \geq 0$ qui est la quantite maximale de flot qui peut passer par cet arc

Les sommets de production positive sont appelles des "sources". Inversement, ceux de production negative (ils ont une "demande" positive) sont appelles "puits".

Definition 2 *Flot admissible* ☯

Pour qu'un flot soit dit valide, ou « admissible », il doit respecter certaines contraintes :

1. Contrainte de capacite:

   Le flot sur un arc ne depasse pas la capacite maximale de l'arc

   $\forall e \in E\ ,\ 0 \leqslant f(e) \leqslant u(i, j)$
2. Contrainte d'equilibre

   Pour tout sommet le flot entrant doit etre egal au flot sortant plus la production du sommet

   $$\forall i \in V\ ,\ \sum_{j\ tq\ (j,i)\in E} f(j, i) = \sum_{j\ tq\ (i,j)\in E} f(i, j) + b(i)$$
3. Contrainte d'integralite

   Les flots sur les arcs prennent des valeurs entieres

Remarque : la contrainte 3 n'est pas necessaire theoriquement car le concept de flot prenant des valeurs reelles est tout a fait concevable. Cependant, les processeurs actuels ne pouvant manipuler des reels arbitraires, on sera limite dans la pratique a travailler avec des flots a valeurs flottantes. De la sorte, on pourra toujours se ramener aux entiers, quitte a multiplier par un facteur 10^n.

Definition 3 *Cout total d'un flot* ☯

Si un flot est admissible, on peut lui associer un cout total, defini par la fonction suivante :

$$\varphi_E(f) \;=\; \sum_{(i,j)\in E} c(i,j)\cdot f(i,j)$$

Ainsi, les $c(i,j)$ correspondent a un « cout par unite de flot » dont on devra s'aquitter proportionnellement a la quantite de flot qu'on l'on fait passer par l'arc (i,j)

Avec ces definitions emergent de nouvelles questions: Existe-t-il des flots admissibles pour tous les graphes? Peut-on systematiquement maximiser la valeur d'un flot pour un graphe donne, et si oui en minimiser le cout total?
Dans ce projet, on se concentrera sur les graphes de flot dont les arcs ont des capacites infinies, et tels que la somme des productions et des demandes soit nulles (i.e. le reseau produit autant qu'il en consomme). Grace a cette hypothese simplificatrice, on peut montrer que de tels graphes possedent toujours un flot admissible de cout minimal:

Proposition 1 *Existence d'un flot de cout minimal (min-cost flow)*

Soit G un graphe de flot de capacite infinie et pour lequel $\sum_{i\in V} b(i) \;=\; 0$.
Alors il existe un flot admissible sur G dont le cout est minimal parmi tous les flots admissibles.
Dans la suite, on appellera les flots de couts minimaux des "flots optimaux".

Les problemes max-flow (trouver un flot de valeur maximale sur un graphe avec une unique source et un unique puit) et min-cost flow (trouver un flot admissible de cout minimal) sont tres classiques, et la litterature sur le sujet fournit des algorithmes en temps fortement polynomiaux et pseudo-polynomiaux respectivement, que nous utilisons d'ailleurs dans ce projet. (Thomas H. Cormen 1989 ; Ravindra K. Ahuja 1993) (voir *infra*)

Dans la suite, on appellera -1- **Probleme min-cost** le probleme suivant:

Etant donne un ensemble V de sommets du plan, determiner $E^\star$ un ensemble d'arcs et $f^\star$ un flot admissible sur le graphe (E,V) tel que le cout total $\varphi(f^\star)$ soit minimal parmis tous les autres flots possibles, c'est-a-dire:

$$E^\star \text{ et } f^\star \text{ realisent} : \; \varphi_{E^\star}(f^\star) = \min_{\substack{E\in \mathcal{P}(V\times V) \text{ ensemble d'arcs} \\ \text{et} \\ f \text{ un flot admissible sur } (V,E)}} \varphi_E(f)$$

2.2. Specification du probleme et motivations

-1-La problematique de ce projet est la suivante :

Problematique:

> Nous souhaitons savoir si les graphes de flot de couts minimaux possedent des sous-structures optimales. C'est-a-dire, formule autrement, :
> Peut-on systematiquement deduire un flot optimal sur un graphe G d'une combinaison de flots optimaux sur des sous-graphes de G? Et reciproquement?

Cette question est interessante d'un point de vu algorithmique puisqu'a l'affirmative, elle nous permet de concevoir des strategies de resolution du probleme min-cost d'une maniere qui differe des solutions classiques sur le sujet (via une approche par programmation dynamique). L'interet reside dans les applications possibles: en effet, de tels reseaux optimaux possedent la propriete "cut-and-paste" decrite dans le CLRS(Thomas H. Cormen 1989), ce qui permet d'ameliorer le cout total d'un reseau preexistant en "collant" un sous-graphe optimal dans le graphe inital.

Pour repondre a ce questionnement, nous nous proposons, dans un premier temps, de développer une intuition sur la forme generale que peuvent prendre ces sous-structures optimisantes par une approche experimentale (Section 3.). Nous nous attarderons ensuite sur la resolution theorique a proprement parler dans une seconde partie (Section 4.).

3. Approche experimentale et empirique

On notera dorenavant, pour un graphe $G = (V, E)$:

$$n := |V| \quad m := |E| \quad U := \max_{v \in V \text{ et } e \in E} \{u(e), b(i)\} \quad \text{et} \quad C := \max_{e \in E} c(e)$$

On essaie dans cette partie de se familiariser avec les differentes formes que peuvent prendre le squelette d'un graphe de flot minimal (solution du Probleme min-cost -1-) pour des instances de V de tailles variables.

Pour ce faire, etant donne un ensemble de sommets V quelconques (munis de leurs demandes respectives), nous allons resoudre le probleme min-flow sur le graphe complet construit a partir de V grace a des algorithmes classiques.
Plus precisement, l'approche consiste d'abord a generer un ensemble aleatoire V de n sommets du plan euclidien, chacuns munis d'un attribut « demande » $b(i) \in \mathbb{Z}$, tels que la somme de toutes les demandes de V soit nulle (pour respecter la contrainte d'equilibre).

Le plan est aussi muni d'une fonction de "distance"[1]. On calcule ensuite un graphe de flot dont le cout global est minimal parmi tous les flots possibles sur le graphe complete de V.

Dans ce projet, on applique en particulier les methodes ☯ :

- de « cycle-canceling » : un algorithme classique en temps non polynomial ($O(m^2 n\, UC)$) qui procede par eliminations successives des cycles a couts negatifs dans le graphe residuel (voir Section 3.4. ou l'on a joint la page **GitHub** du projet ou l'on a implemente cet algorithme)
- et de convergence par « cost-scaling » ou "echelonnement des couts": en temps pseudo-polynomial $(O(mn \log n \log C)$ (aussi implementee dans la Section 3.4.)

Elles sont toutes les deux decrites dans la reference (Ravindra K. Ahuja 1993).

Puisque nous sommes amenes a manipuler des graphes complets, et que les algorithmes que nous utilisons ont une complexite asymptotique importante (au moins d'ordre n^5 pour cycle canceling, n^3 par double scaling), il devient necessaire d'utiliser des structures de donnees efficaces afin de pouvoir utiliser nos algorithmes en temps raisonnable. C'est l'objet de la partie suivante.

3.1. Struture de graphe de flot efficace

On souhaite pouvoir manipuler des graphes de flots de grande taille, tout en ayant la capacite d'ajouter, de modifier et de supprimer des aretes afin pouvoir liberer de la place en memoire lorsqu'un arc n'est plus utilise (emondage). De plus, on veut pouvoir avoir acces aux voisins et aux predecesseurs d'un sommet en temps raisonnable.

[1] Il ne s'agit pas ici d'une distance au sens topologique du terme. On fait ici reference a une fonction de cout quelconque entre deux points A = (x,y) et B = (u,v) du plan. On verra plus tard que les proprietes de la fonction de cout choisie a un impact direct sur la structure des solutions.

	Matrice d'adjacence	Liste d'Adjacence	Foward and reverse star	(Dans ce rapport)
En memoire	$\Theta(n^2)$	$\Theta(n+m)$	$\Theta(n+m)$	$\Theta(n+m)$
Liste des predecesseurs et des successeurs	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$ (sous forme d'iterateur)
Existence d'un arc	$\Theta(1)$	$O(a_{max})$ ou a_{max} est l'arité maximale des sommets $O(n)$ pour une borne polynomiale	$\Theta(1)$	$\Theta(1)$
Ajout d'un arc	$\Theta(1)$	$O(a_{max})$ ou $O(n)$	–	$\Theta(1)$

Figure 1: Comparaison des complexites temporelles des differentes implementations classiques de graphe vis-a-vis de celle proposee dans ce rapport

Il se trouve que les implementations classiques des graphes, a savoir les implementations par listes et matrices d'adjacence, et le modèle « Forward and Reverse Star »(ou *FRS*) tel que decrit dans (Ravindra K. Ahuja 1993), offrent la possibilite pour chacunes, de construire certaines des primitives enoncees plus tot. Mais aucune ne peut toutes les supporter, ou du moins dans une complexite temporelle interessante.
Pour cette raison, on developpe une structure de donnee inspiree du modèle *FRS* qui pourra implementer toutes ces primites en temps constant pour la plus part, avec une contrepartie en memoire en $\Theta(m+n)$ (voir tableau comparatif Figure 1)

3.2. Structures auxilliaires

On aura besoin des structures auxilliaires : `dynamic_array` et `linked_array`.
L'implementation de ces structures en Ocaml est disponible en annexe

3.2.1. Type `dynamic_array` ☯

C'est un tableau dynamique classique, redimensionnable, muni des primitives de la Figure 8 Section 6. Annexe et Bibliographie.
Les indices sont fixes: une cellule gardera le meme indice de son ajout au tableau jusqu'a sa suppression.
En pratique, l'implementation de cette structure de donnees inclut d'autres fonctions utiles, telles que des iterateurs de type folds, des "mutators" type map, etc...

3.2.2. Type `linked_array`

Cette structure ressemble a celle d'une liste doublement chainee, mais dont on peut acceder aux elements des cellules en $O(1)$, comme dans un tableau de type `Array`.

En quelque sorte, tout se passe comme si on munissait un tableau d'un ordre de parcours. (voir le diagramme explicatif Figure 2).
Les indices sont fixes: une cellule gardera le meme indice de son ajout a la chaine jusqu'a sa suppression.

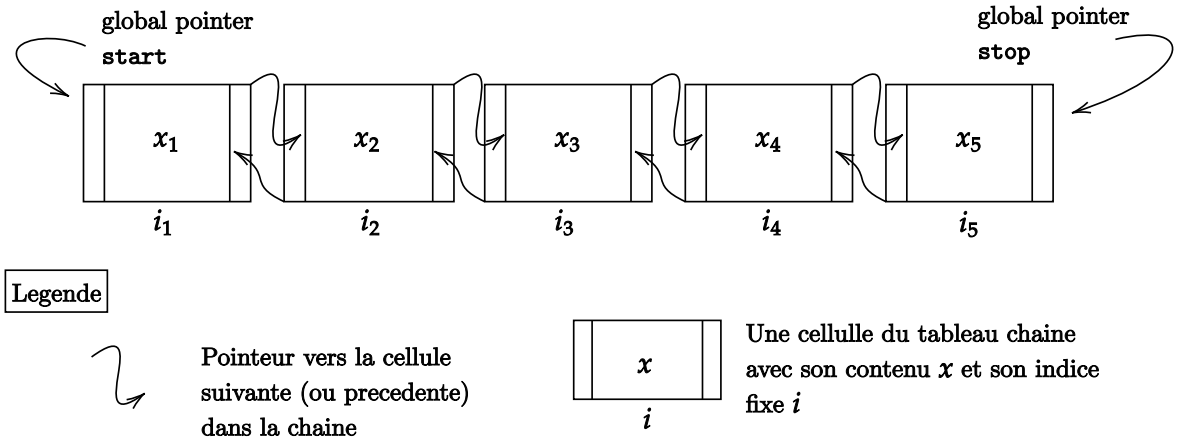


Figure 2: Diagramme illustrant la structure linked_array

On pourra consulter l'interface de cette structure dans Figure 9, dans la Section 6. Annexe et Bibliographie.
Il est possible d'inserer un element en debut ou en fin de chaine, ou entre deux cellules consecutives.
On dote aussi cette structure d'une fonctionnelle `iter_range : 'a linked_array -> int -> int -> iterateur` qui prend en entree `tbl : linked_array` et deux indices i et j, et qui renvoit un iterateur qui va parcourir la chaine des elements de tbl entre l'indice i et l'indice j .

De maniere analogue a la structure `dynamic_array`, on a aussi pris la liberte de fournir d'autres primitives utiles (folds, maps, etc...).

3.3. Type Flow_graph

Grace aux types auxiliaires decrits precedement, on peut creer une structure de graphe qui remplit le cahier des charges decrits au debut de la Section 3.1. (voir le diagrame explicatif Figure 3):

- A la creation du graphe, on assigne a chaque sommet un unique indice id que l'on enregistre dans une table d'association (dictionnaire).

- A chaque fois que l'on ajoute un arc, on lui attribue un unique indice `id`
  On stocke les informations sur cet arcs (capacite, cout, flot) dans une `dynamic_array`, a l'indice `id` fixe auparavant. Ainsi, toutes les informations sur cet arc sont disponibles en $O(1)$ par une simple consultation du tableau des arcs

- Pour exploiter l'information concernant les successeurs, on maintient un doublet (`forward_range , forward_chain`) de type i`nt array *  int linked array` a chaque ajout ou suppression d'un arc :
  - Dans la chaine `forward_chain,` on insere consecutivement les successeurs d'un meme sommet a chaque fois que l'on ajoute un arc, de sorte que pour obtenir la liste des voisins d'un sommet v particulier, il suffit de se rappeller des indices du debut et de la fin de la plage reservee a l'adjacence de v dans `forward_chain`.
  - C'est exactement ce que l'on fait dans le tableau `forward_range`, aux indices $2i$ et $2i+1$, ou l'on stocke le debut et la fin de la plage des sommets du graphe.

Ainsi, on conserve un acces a « l'ensemble » des voisins d'un sommet en $O(1)$ simplement en consultant sa plage dediee dans « `forward_range` ».

- Le processus de suppression en decoule naturellement.

- On aura aussi pris soin de se rappeler des indices des arcs, grace a une table d'association par exemple.

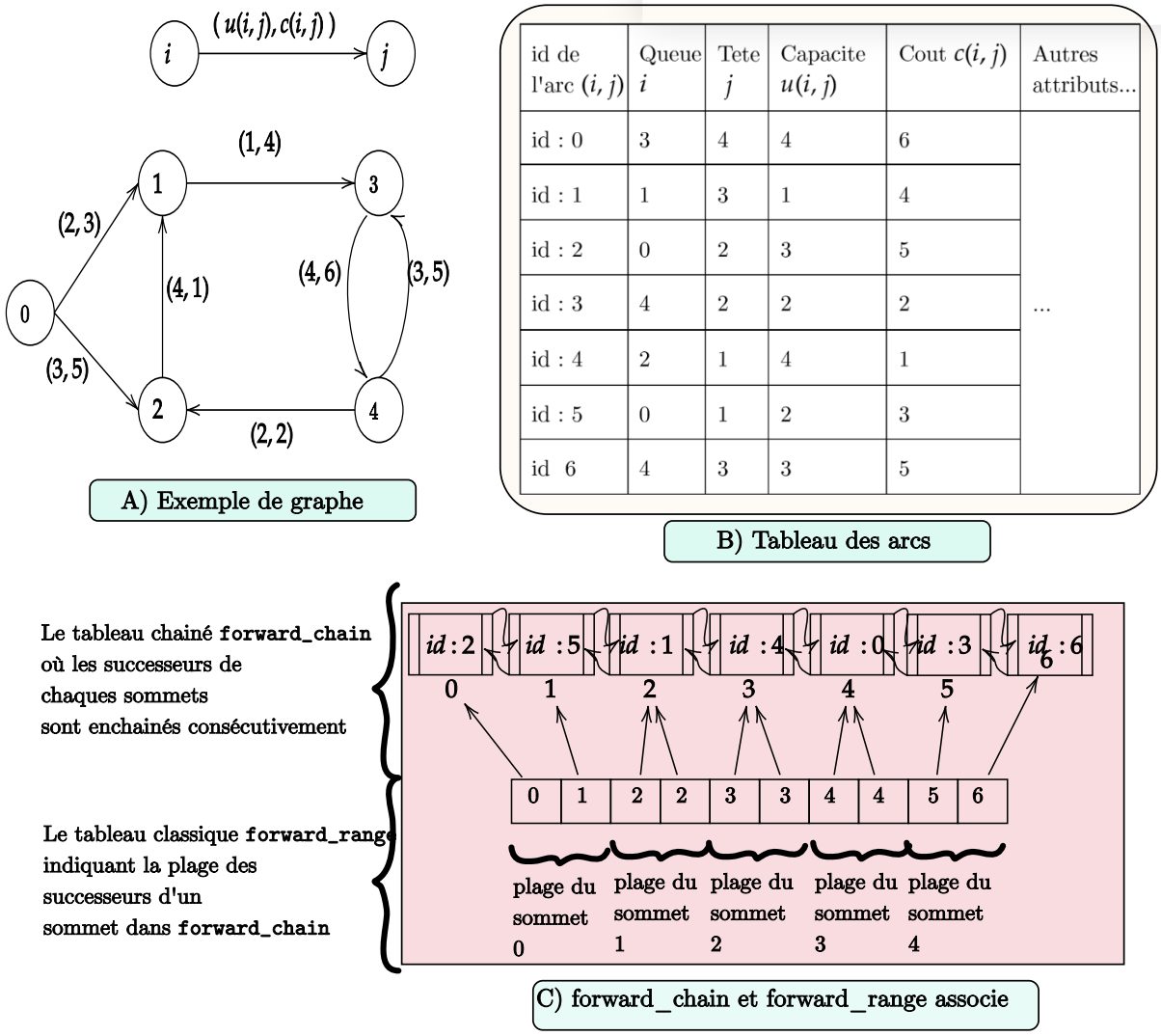


Figure 3: Diagramme illustrant la structure `flow_graph`
A) : un exemple de graphe de flot avec 5 sommets
B) le tableau dynamique des arcs associes au graphe A
C) les tableaux `foward_range` et `foward_chain` stockant l'information sur les successeurs de chaque sommets

Grace a cette construction, on peut modifier le graphe en place, et ajouter ou retirer des arcs en temps constant, ce qui est tres utile pour les operations d'emondage.

3.4. Implementation en OCaml du projet

Nous avons implementes les structures de donnees decrites dans les sections precedentes ainsi que les algorithmes classiques de resolution du probleme min-cost en OCaml. Voici la page GitHub sur laquelle vous pourrez retrouver l'entierete du code

source de ce projet, ainsi que les fichiers compiles en remontant dans l'arborescence:

https://github.com/Soonies/TIPE_graphes_flot_efficaces/tree/master/banana/lib

(2 Kilo Lines of Code)

Compilation : Dune
Bibliotheque d'affichage des graphes : plplot → Necessite un server virtuel type XServ + (putty+ WSL) si sous windows

Il y a parfois des details de l'implementation que nous avons choisis de ne pas expliciter ici (parce que pas utile), mais certaines transformations de graphes sont necessaires afin de faire fonction les algorithmes correctement. Cependant le code est en general bien commente.

Contenu du projet (appelle "Banana" par erreur) :

```
# banana/bin/main.ml ->
 fichier executable

# banana/lib ->
*.mli -> fichiers d'interface
*.ml -> fichiers de definition de modules

``` display.ml  ->  module d'affichage, met a disposition des
foncteurs qui permettent l'affichage de graphes polymorphiques

```dynamic_array.ml ; linked_array.ml ;  flow_graph.ml ->
definition des structures de donnees principales ainsi que de leurs
primitives

```flow_graph.md -> markdown document qui documente
l'implementation de la structure flow_graph

```resolution_strategies.ml -> algorithmes et procedures permettant
in fine la generation de graphes de flots optimaux pour des
intances de sommets V variees
```

3.5. Des exemples de graphes de flots minimaux

Voici des exemples de graphes de flots optimaux obtenus pour differentes fonctions de couts, et pour plus ou moins de variabilite au niveau des productions des sommets

(plus ou moins d'ecart entre la plus grande production et la plus grande demande). La quantite du flot passant par un arc est representee par une plus forte epaisseur, et une couleur plus claire. De meme, les productions des sommets sont d'autant plus grandes que le sommet est jaune, et la demande d'autant plus forte que le sommet est bleu.

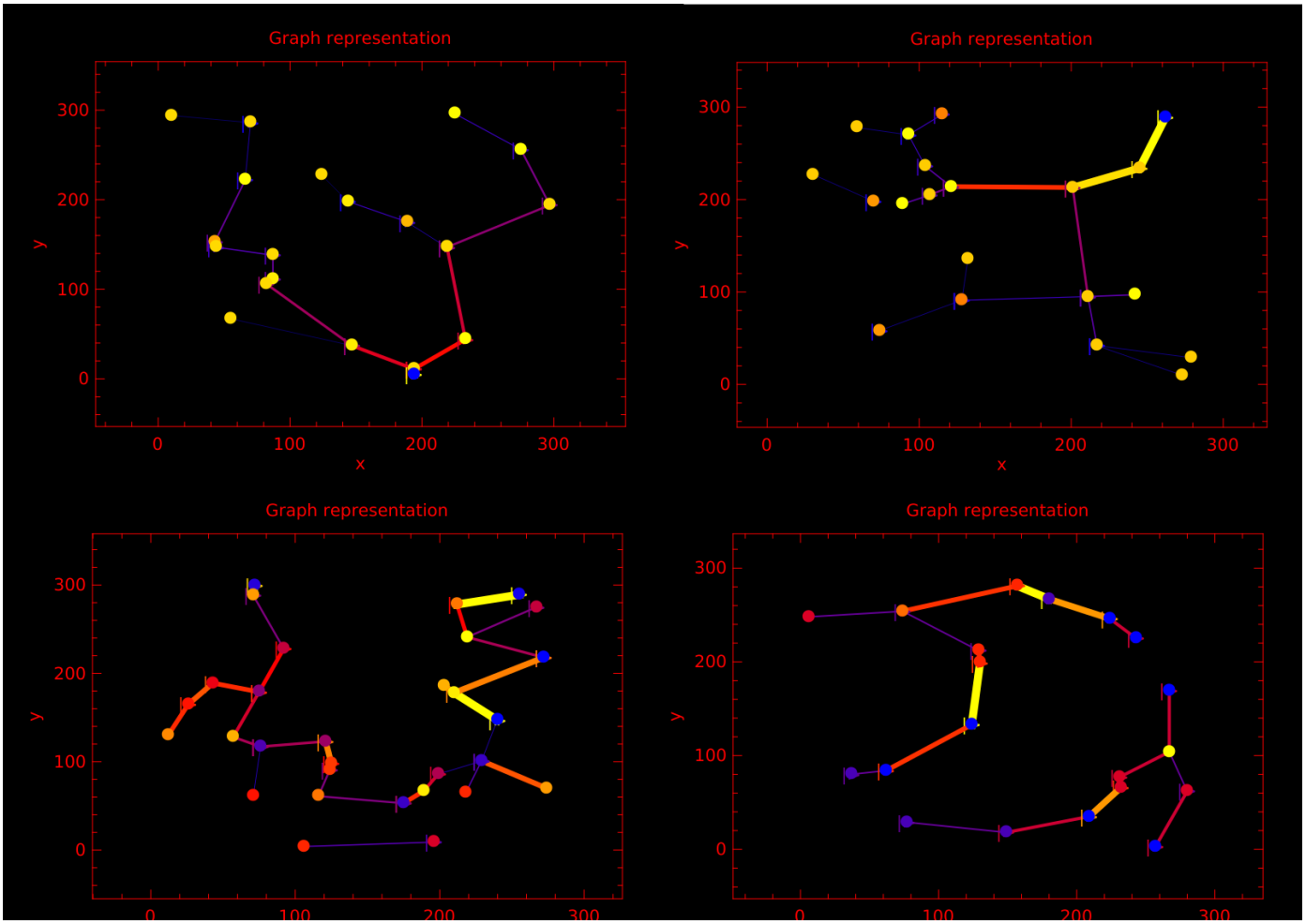


Figure 4: Squelettes de graphes de flots optimaux pour differentes instanes de V et differentes fonctions de cout
Top left et Top Right: Peu de variabilite / Bottom left & Bottom Right: Beaucoup de variabilite

4. Approche theorique

Dans cette partie, nous nous attachons a apporter des elements de reponse formels au probleme -1- que nous nous sommes poses.
Pour un ensemble de sommets V donne, nous souhaitons donc decrire le passage d'une solution optimale d'une sous-structure de V a une autre (ou les sous-structures de V sont les parties V. Ainsi, si on a numerote les sommets de $V = \{v_1, v_2, \dots, v_n\}$, une sous-structures de V est par exemple l'ensemble $\{v_2, v_4, \dots, v_{16}\}$) .

On commence par enoncer quelques proprietes generales sur la structure des graphes de flot solutions du probleme min-cost, ainsi qu'une reduction du probleme qui nous permettra d'en simplifier l'etude, puis on decrit des strategies qui exploitent la sous-optimalite dans des cas pariculiers du probleme.

4.1. Proprietes generale du squelette des graphes optimaux

Proposition 2 *Propriete des cycles a couts negatifs* ☯

Un graphe est solution du probleme min-cost ssi son graphe residuel ne contient pas de cycles de couts negatifs

Corollary 1 *Squelette de graphes optimaux*

Le squelette d'un graphe de flot optimal est une foret

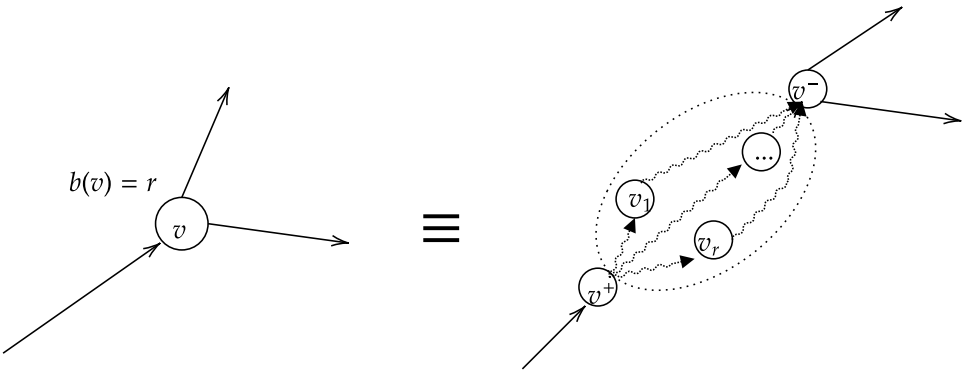
4.1.1. Reduction du probleme

On propose une reduction du probleme aux graphes de productions unitaires ($b(i) = \pm 1$) qui nous permettra de simplifier notre etude grace au ?:

Proposition 3 *Reduction aux graphes de production unitaire*

Tout graphe de flot est "equivalent" a un graphe de flot dont tous les sommets ont une production $b(i) \in \{1; 0; -1\}$

Preuve:



Graphe original

Graphe reduit

$b(v^+) = b(v^-) = 0$

et

$\forall i,\ c(v^+, v_i) = c(v_i, v^-) = 0$

Figure 5: Passage au graphe de flot reduit

de v, ainsi que les $b(v)$ nouveaux sommet relies a v^+ et v^- par des arcs de couts nuls et de capacite 1.

Reciproquement, pour tout graphe reduit $\widetilde{G}$, il existe un unique graphe de flot classique G dont la reduction est $\widetilde{G}$. □

Lemma 1 *Equivalence graphes de flot unitaire et graphes de flot classiques*

Les deux assertions suivantes sont equivalentes:

(1) Une propriete P est vraie pour les graphes dont les productions sont dans $\{\pm 1, 0\}$

(2) P est vraie pour tous les graphes de flots

4.2. Passage de n a $n-1$

Supposons que l'on dispose d'un graphe minimal $G = (V, E)$ muni d'un flot $f^\star$ solution du probleme min-cost pour un ensemble de sommet V a n elements. Comment en deduire un graphe minimal sur un sous ensemble de V a $n-1$ elements?

Notons v le sommet supprime de V, et $V' := V \setminus \{v\}$,
$Adj^+[v]$:= l'ensemble des predecesseurs de v dans G,
$Adj^-[v]$:= l'ensemble des successeurs de v dans G,

4.2.1. Un algorithme simple

Ci-apres nous donnons un algorithme en $O(m+n)$ qui opere a la descente d'ordre.

Initialisation
Soit $G' = (V', E') := (V', E \setminus \{\textit{ensemble des arcs impliquants } v\})$
Pour tout $e \in E'$,
    $f(e) \leftarrow f^\star(e)$
Execution
Pour tout $u \in Adj^+[v]$:
    $b(u) \leftarrow b(u) + f^\star(u, v)$
Pour tout $w \in Adj^-[v]$:
    $b(w) \leftarrow b(w) + f^\star(v, w)$

Proposition 4

Le flot f de G' est un flot admissible

Proposition 5

G' est un graphe de flot minimal pour l'ensemble V'

Preuve:
On utilise un argument "cut-and-paste"
Par l'absurde, supposons que f *n'est pas un flot optimal sur* G'*. Alors il existe un flot* f' *sur* G' *de cout strictement inferieur a celui de* f*. Soit alors* f_0 *un flot sur* G *tel que:*
$$\begin{cases} \forall e \in E',\ f_0(e) = f'(e) \\ Si\ non,\ \ f_0(e) = f^\star(e) \end{cases}.$$
Alors

$$\varphi_E(f_0) = \sum_{e\in E'} c(e)f'(e) + \sum_{(u,v)\in E} c(u,v)f^\star(u,v) + \sum_{(v,w)\in E} c(v,w)f^\star(v,w) \overset{par\ hypothese}{<} \varphi_{E^\star}(f^\star),$$

ce qui est absurde car on a suppose G *optimal. Conclusion:* f est bien un flot optimal sur G'. □

4.2.2. Interet de la methode

L'interet du passage d'un graphe d'ordre n a un graphe d'ordre $n-1$ est limite en general, car on pourrait se contenter d'appliquer directement les algorithmes de resolution a l'ordre $n-1$. Cependant, pour des configurations de sommets particulieres (en cercle, en ligne droite, graphes creux...), on dispose d'algorithmes rapides qui peuvent trouver une solution au probleme min-cost en $O(n \log n)$, comme ceux decrits dans l'article (Vaidyanathan and Ahuja 2010).
Des lors, l'enjeux devient de reconnaitre une structure speciale partielle dans certains ensembles de sommets, afin de les completer judicieusement pour retrouver une structure speciale complete, et d'appliquer ensuite ces algortihmes efficaces. On opere enfin l'algorithme de redescente autant de fois que l'on a ajoute un sommet pour revenir au probleme initial. Cette methode permet d'obtenir une solution en $O(n \log n + k(m+n))$, ou k est le nombre de sommets completes.

4.3. Passage de n a $n+1$

Supposons que l'on dispose d'un graphe minimal $G = (V, E)$ muni d'un flot $f^\star$ solution du probleme min-cost pour un ensemble de sommet V a n elements. Comment en deduire un graphe minimal sur un ensemble contenant V, a $n+1$ elements?

Quand le passage d'une instance de taille n a une autre de taille inferieure etait assez direct,
l'inverse est plus complexe. En effet, il y a plusieurs possibilites pour faire en sorte de conserver la propriete d'equilibre. La propriete suivante nous permet de travailler sur n'importe quelle de ces possibilites, puisque la propriete d'optimalite du flot trouve nous permet de nous ramener a toutes les autres.

Proposition 6 *Lien entre flots optimaux sur un ensemble de sommets du plan*

Si V et V' sont deux ensembles de sommets du plan identiques, mais dont les productions respectives de chaques sommets sont differentes, et qu'on connait un flot optimal sur V, alors on peut directement en deduire un flot optimal sur V'.

Preuve : Theoreme de decomposition

5. Pistes de recherche pour la suite

Du fait de la contraite de temps (concours), nous n'avons pas encore pu aboutir a la resolution de notre probleme initial meme si nous avons des pistes, que nous poursuivrons ulterieurement:

- On remarque d'ores et deja que le squellette du graphe optimal sera fortement influe par les fonctions de couts en fonction des proprietes qu'elles possedent (des inegalites triangulaires nottament). En particulier, pour des fonctions de types distaces $||\ .\ ||_p$, on obtient des graphes en forme d'etoile (car on peut prouver que pour ces normes, quelque soit le flot passant par un arc, le chemin le moins cher est la ligne droite) (voir Figure 6). Tandis que pour des normes du type $x^n + y^n$, aura plutot une structure semblable a celle de la vascularisation d'une feuille (voir Figure 7)
- Des lors, on peut donner du sens a la notion "d'independance d'un ensemble de sommet" vis a vis du reste du graphe, qui nous permettra de reduire le probleme a des instances plus petites: c'est notre idee de sous-structures optimales

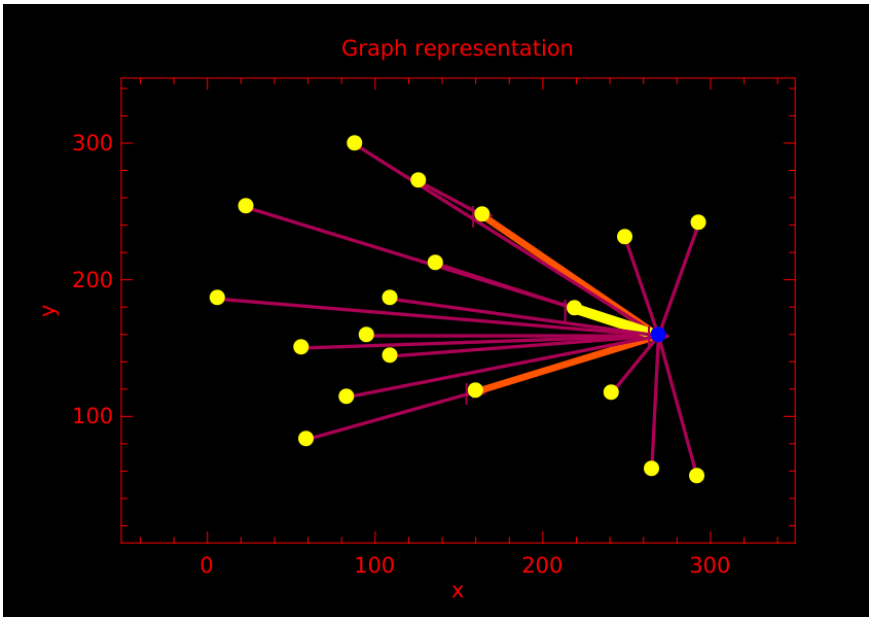


Figure 6: Graphe optimal obtenu pour la norme N_4

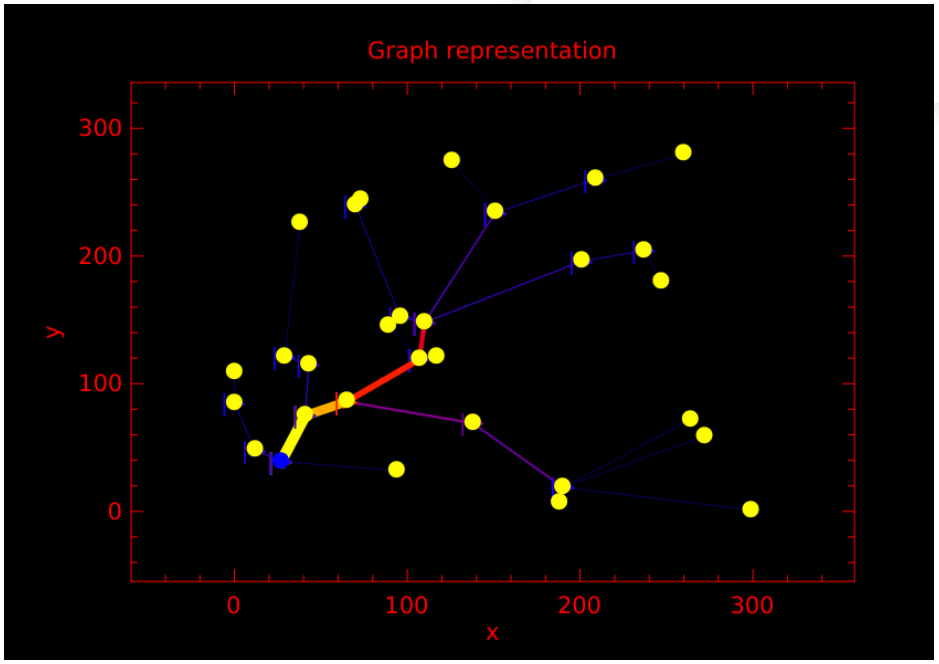


Figure 7: Graphe optimal obtenu pour une fonction de cout quadratique

6. Annexe et Bibliographie

Figures annexees:

	Specification	En memoire	En temps

`create n`
 	
`int-> 'a t`
	Input: n la taille anticipee du tableau a creer
	Output : un tableau vide
		$\Theta(n)$	$\Theta(1)$

`add tbl x`
 	
`'a t -> 'a -> int`
	Input : Un tableau tbl / un element x a ajouter a tbl
	Output : l'indice de x dans tbl
	Action : l'element x a ete ajoute a tbl
		$\Theta(1)$ en complexite amortie
		$O(n)$ si non
			$\Theta(1)$ amortie
			$O(n)$ si non

`remove tbl i`
 	
`'a t -> int -> unit`
	Input: Un tableau tbl / l'indice de l'element a enlever i
	Output: -
	Action: L'element d'indice i a ete retire du tableau
		$\Theta(1)$ amortie
		$O(n)$ si non
			$\Theta(1)$ amortie
			$O(n)$ si non

`set tbl i x`
 	
`'a t -> int -> 'a -> unit`
	Input: Un tableau tbl / un indice i / un element x
	Output: -
	Action: L'element d'indice i de tbl a ete remplace par x
		$\Theta(1)$	$\Theta(1)$

`see tbl i`
 	
`'a t -> int -> 'a`
	Input: Un tableau tbl / un indice i
	Output: L'element d'indice i de tbl
		$\Theta(1)$	$\Theta(1)$

Figure 8: Extrait de l'interface de la structure dynamic_array

Primitive	Specification	En memoire	En temps

`create n`
 	
`int-> 'a t`
	Input: n la taille anticipee du tableau a creer
	Output : un tableau chaine vide
		$\Theta(n)$	$\Theta(1)$

`insert_add tbl i x`
 	
`'a t -> int -> 'a -> int`
	Input : Un tableau tbl / indice i / element x a ajouter a tbl
	Output : l'indice de x dans tbl
	Action : l'element x a ete iserer dans tbl juste apres l'element d'indice i
		$\Theta(1)$ en complexite amortie
		$O(n)$ si non
			$\Theta(1)$ amortie
			$O(n)$ si non

`remove tbl i`
 	
`'a t -> int -> unit`
	Input: Un tableau tbl / l'indice de l'element a enlever i
	Output: -
	Action: L'element d'indice i a ete retire du tableau
		$\Theta(1)$ amortie
		$O(n)$ si non
			$\Theta(1)$ amortie
			$O(n)$ si non

`set tbl i x`
 	
`'a t -> int -> 'a -> unit`
	Input: Un tableau tbl / un indice i / un element x
	Output: -
	Action: L'element d'indice i de tbl a ete remplace par x
		$\Theta(1)$	$\Theta(1)$

`see tbl i` `'a t -> int -> 'a`	Input: Un tableau tbl ~ un indice i Output: L'element d'indice i de tbl	$\Theta(1)$	$\Theta(1)$
`previous tbl i (resp. next)` `'a t -> int -> pointer`	Input: tableau tbl ~ indice i Output: le pointeur renvoyant au predecesseur (resp. successeur) de l'element d'indice i	$\Theta(1)$	$\Theta(1)$
`iterc_range f tbl` `( int -> 'a -> unit) -> 'a t -> int -> int -> unit`	Input : fonction f ~ tableau tbl ~ indice de debut a ~ indice de fin b Output: - Action: applique f aux elements de tbl et a leurs indices le long de a chaine, entre l'indice a et l'indice b	$\Theta(1)$	$\Theta(b-a)$

Figure 9: Extrait de l'interface de la structure linked_array

Ravindra K. Ahuja T.L.M. James B. Orlin1993 – *Network Flows: Theory, Algorithms, and Applications.*

Thomas H. Cormen1989 – *Introduction to Algorithms.*

Vaidyanathan B., **Ahuja R.2010** – Fast Algorithms for Specially Structured Minimum Cost Flow Problems with Applications, *Operations Research*, 58, pp. 1681–1696.