Sur le problème des préflots

DUVILLIE Guillerme OULD MOHAMED ABDELLAHI CHEIKH Mehdi

15 mai 2012

Table des matières

Première partie

Introduction

Le sujet de TER porte sur la recherche d'un flot maximum dans un graphe quelconque par l'utilisation d'algorithmes reposant sur la notion de *préflot*, qui est une fonction obtenue par relaxation des contraintes de la structure de flot.

1 Un peu d'histoire

Le problème de flot maximum, consistant à déterminer la quantité maximum de *ressources* que l'on peut transporter d'un point A à un point B dans un graphe, trouve ses racines, d'après A. Schrijver [6] dans un rapport de Harris et Ross rédigé pour l'US Air Force, en 1955, intitulé *Fundamentals of a Method for Evaluating Rail Net Capacities*. Il s'agit là de la résolution d'un problème de flot maximum dans un graphe basé sur le réseau de chemin de fer couvrant l'ouest de l'Union Soviétique et les pays satellites de l'Europe de l'est, afin d'obtenir une coupe minimale de celui-ci, ce pour des raisons stratégiques.

Le problème fut ensuite étudié par Ford et Fulkerson qui rédigèrent un algorithme permettant de résoudre le problème par l'utilisation de chaînes améliorantes dans le graphe, en 1955. De nombreuses améliorations de l'algorithme ont vues le jour telles que l'algorithme d'Edmonds-Karp en 1969 [5] ou l'algorithme de Dinic en 1970 [5], permettant, à l'aide d'une sélection judicieuse des chaînes améliorantes de réduire la complexité de l'algorithme initial.

Note : Il sera supposé, tout au long de ce document que le lecteur connaît ces derniers algorithmes.

Cependant, il existe de nombreux graphes pour lesquels l'exécution des algorithmes cités ci-dessus est très longue. Le graphe représenté à la figure 1 (p. 4) est l'un d'eux. C'est pour cette raison que la notion de préflot a été développée, et aujourd'hui, les algorithmes basés sur les préflots font partie des plus rapides développés à ce jour.

L'algorithme d'Edmonds-Karp, recherchant une chaîne améliorante de plus court chemin dans le graphe, effectue $(n-6)$ recherche de chaînes améliorantes et atteint son temps maximal d'exécution. Ce problème se résout très vite à l'aide de préflots, ce que nous allons voir.

Au préalable, la notion de préflot sera définie [1], puis le principe de chaque algorithme sera présenté, ainsi que la démonstration de leur validité et leur borne de temps d'exécution. La dernière partie portera sur le détail des tests effectués et sur le traitement des résultats [2].

1. Ainsi qu'un bref rappel de la notion de flot
2. Les structures de données utilisées sont présentées dans la documentation fournie

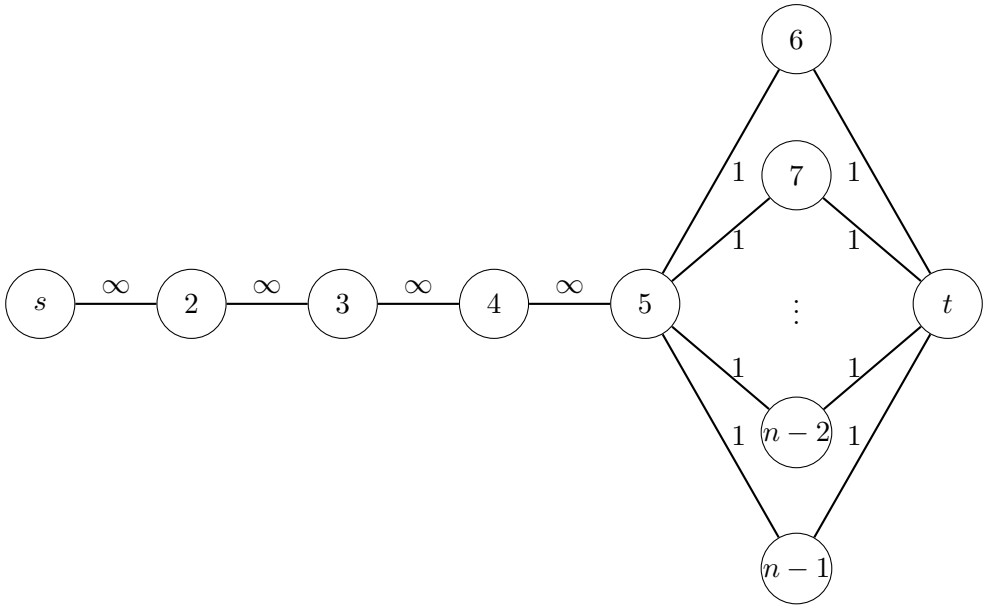


FIGURE 1 – Graphe mettant en défaut les algorithmes de recherche de chaînes améliorantes

2 Définitions

2.1 Quelques rappels sur les graphes

Un graphe G est défini par un ensemble de sommets noté S et un ensemble d'arêtes notées A [3], ainsi qu'un sommet source noté s et un sommet puits t.

Les notations utilisées dans les graphes :

Soit un sommet $i \in S$, le voisinage sortant (respectivement entrant) de i sera noté $A^+(i)$ (resp. $A^-(i)$).
Soient $i, j \in S$, s'il existe une arête reliant i à j appartenant à A, cette dernière sera notée (i, j) et sa capacité $c(i, j)$.

2.1.1 La notion de flot

Dans un graphe $G(S, A)$, un flot f de G est une application $x : A \to \mathbb{N}$ vérifiant les conditions suivantes :

$$\forall (i,j) \in A \qquad 0 \leq x(i,j) \leq c(i,j) \quad \text{avec } x(i,j) \text{ la valeur du flot sur l'arête } (i,j) \tag{1}$$

$$\forall i \in S - \{s,t\}, \quad \sum_{j \in A^+(i)} x(i,j) = \sum_{k \in A^-(i)} x(k,i) \tag{2}$$

$$f = \sum_{j \in A^+(s)} x(s,j) = \sum_{k \in A^-(t)} x(k,t) \tag{3}$$

Toute l'essence du problème étudié consiste en la maximisation de cette valeur.

3. Il paraît important de faire remarquer que la définition d'un graphe non simple est parfaitement rigoureuse, si l'on rajoute à cette définition une fonction dite d'incidence notée $\gamma : A \to S \times S$ qui associe à chaque arête de A un couple de sommets de S.

2.1.2 La notion de préflot

Le préflot étant une fonction obtenue par relaxation de contraintes sur les flots, son domaine de définition est donc le même : il s'agit d'une application $x : A \to \mathbb{N}$ respectant les contraintes suivantes :

$$\forall (i,j) \in A \qquad 0 \leq x(i,j) \leq c(i,j) \tag{4}$$

$$\forall i \in S - \{s,t\}, \quad \sum_{j \in A^+(i)} x(i,j) \leq \sum_{k \in A^-(i)} x(k,i) \tag{5}$$

$$\sum_{j \in A^+(s)} x(s,j) \leq \sum_{k \in A^-(t)} x(k,t) \tag{6}$$

On introduit alors la valeur $e(i)$ représentant l'excédent de flot d'un noeud $i \in S$, elle est définie comme la différence entre le préflot entrant dans un noeud et le préflot sortant de celui-ci :

$$e(i) = \sum_{k \in A^-(i)} x(k,i) - \sum_{j \in A^+(i)} x(i,j) \tag{7}$$

On peut alors écrire l'inéquation (5) de cette manière :

$$\forall i \in S - \{s,t\}, \quad \sum_{j \in A^-(i)} x(i,j) = \sum_{k \in A^+(i)} x(k,i) + e(i) \tag{8}$$

De la même manière, l'inéquation 6 sera réécrite de la sorte :

$$\sum_{j \in A^+(s)} x(s,j) = \sum_{k \in A^-(t)} x(k,t) + \sum_{i \in S - \{s,t\}} e(i) \tag{9}$$

Un sommet $i \in S$ ayant un excédent e non nul : $e(i) > 0$ sera qualifié appelé sommet actif, on dira aussi qu'il est débordant.

2.1.3 La notion de réseau résiduel

Soit un graphe $G(S,A)$, un flot f et i,j deux sommets e G. On dira que l'arête (i,j) appartiendra au réseau résiduel A_f si et seulement si :

$$r(i,j) = c(i,j) - x(i,j) > 0 \tag{10}$$

On appelle alors $r(i,j)$ la capacité résiduelle de l'arête (i,j). De plus la présence d'un flot f circulant sur l'arc (i,j) de valeur $x(i,j)$ entraîne la présence d'une arête (j,i) dans le réseau résiduel de capacité résiduelle $r(j,i) = x(i,j)$.

2.2 Quelques fonctions importantes

2.2.1 Les fonctions de potentiel

Il est important de réaliser un point précis sur les fonctions de potentiel. Les démonstrations réalisées au cours de ce rapport se basent sur l'utilisation de ces fonctions.

Considérons une structure de données appelée D_0 sur laquelle sont effectuée des opérations numérotées. Appelons D_i la structure de données résultant de la i^e opération. On appelle fonction de potentiel, toute fonction Φ qui, à une structure de données D_i associe un réel appelé potentiel, noté $\Phi(D_i)$. Ce formalisme est décrit à la figure 2 (p. 6).

L'incidence de chacune des opérations sur la valeur de la fonction potentielle et la recherche d'une borne supérieure à cette dernière permettra de déterminer des bornes sur le nombre d'opérations effectuées et donc sur la complexité de l'algorithme.

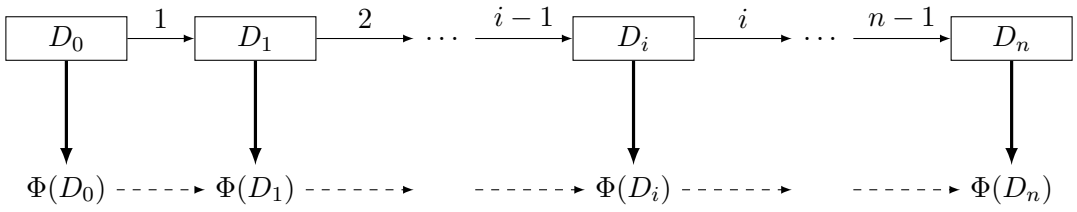


FIGURE 2 – Représentation du principe d'une fonction de potentiel

2.2.2 Fonction de distance

Soit $G(S, A)$ un graphe ayant s pour sommet source et t pour sommet puits. Une fonction $d : S \to \mathbb{N}$ est une fonction de distance [4] si elle vérifie les propriétés suivantes :

$$d(s) = |S| \tag{11a}$$

$$d(t) = 0 \tag{11b}$$

$$\forall (i,j) \in A_f \quad d(i) \leq d(j) + 1 \tag{11c}$$

Une notion inhérente à la fonction de distance, est celle d'*arête acceptable* : une arête (i, j) est dite acceptable si et seulement si $d(i) = d(j) + 1$. Par définition, toute arête acceptable appartient au réseau résiduel.

Deuxième partie
Les algorithmes

Au cours de cette partie l'algorithme de préflot générique sera étudié et sa validité sera démontrée, deux autres algorithmes (FIFO et High Label) très similaires au générique seront présentés et puisque ces derniers ne sont qu'une amélioration de l'algorithme générique, leur validité découlera de la validité de ce dernier.

Les calculs de complexité seront développés dans la partie suivante.

3 L'algorithme Générique

3.1 Principe

L'algorithme générique repose sur les principes suivants :

1. une action locale
2. l'utilisation de la relaxation de contrainte des préflots
3. une fonction de distance

D'une façon imagée, assimilons le graphe à un réseau de tuyaux de capacité donnée, les sommets à des réservoirs pouvant accumuler une quantité infinie de préflot et étant caractérisés par leur distance au puits. Le but de la manoeuvre étant d'acheminer une quantité maximale de flot de la source vers le puits à l'aide du réseau fourni. Pour des raisons de commodité, la distance du nœud source est fixée au nombre de sommets dans le réseau.

4. Aussi appelée *fonction de hauteur*

Intuitivement, il est facile de comprendre que, pour que le préflot aille de la source vers le puits, celui ci doit transiter d'un réservoir a vers un réservoir b satisfaisant la condition suivante : *La distance de a au puits doit être supérieure à la distance de b.* Ceci indique que le nœud cherchant à pousser est plus loin de t que le nœud cible. En imaginant que l'on fasse le contraire, le préflot resterait cantonné à sa source sans même entrer dans le réseau défini par le graphe. L'action de faire transiter le préflot d'un nœud a à un nœud b est appelée opération de *poussage*.

Au moment ou un nœud a reçoit une quantité de préflot de l'un de ses voisins b ayant réalisé une opération de poussage, la quantité de préflot envoyée de b vers a est alors directement stockée dans a, s'ajoutant ainsi à la quantité de flot déjà stockée. Cette dernière sera appelée *excédent du sommet a* et notée $e(a)$. À un moment donné, si, pour un nœud quelconque a, $e(a) > 0$, alors a sera dit excédentaire, débordant ou actif.

Seulement, l'opération de poussage à elle seule ne permet pas d'obtenir un flot. Ceci est illustré par l'exemple de la figure 3 : après la phase 2, le nœud 1 est encore débordant mais il lui est impossible d'effectuer une opération de poussage puisque l'arête le reliant à t est saturée et s et 2 ont une distance au nœud puits supérieure ou égale à celle de 1. Il faut donc introduire une opération supplémentaire, appelée réétiquetage.

Le réétiquetage consiste à réévaluer la distance du nœud débordant au nœud puits de façon à ce qu'il soit possible, après cette opération, d'effectuer une opération de poussage depuis ce nœud. Soit un nœud excédentaire i de distance d_i et son v_i son voisinage de sorte que $\forall v \in v_i$, on a : $d_v \geq d_i$. La distance de i doit être réévaluée afin de pouvoir évacuer l'excédent de flot, il faut donc augmenter d_i jusqu'à ce qu'un poussage puisse être réalisé ce qui revient à fixer $d_i = 1 + \min(d_v), \quad v \in v_i$.

L'algorithme générique se résume en deux étapes qui sont la procédure d'initialisation qui permet de fournir une fonction de distance valide et de créer les premiers nœuds actifs en saturant les arcs sortant de s et une boucle agissant sur les nœuds débordants par poussage ou réétiquetage.

Algorithm 1 Algorithme Générique

```
1: Initialisation()
2: while Il existe un noeud actif i do
3:     Pousser-Réétiqueter(i)
4: end while
```

Algorithm 2 Procédure d'initialisation

```
1: for i ∈ S do
2:     Calculer la distance d(i) en nombre d'arêtes de i à t
3: end for
4: for a ∈ A(s) do
5:     x(a) ← c(a)
6: end for
7: d(s) ← |S|
```

La procédure Pousser-Réétiqueter appelle à quelques explications. La variable p est un booléen permettant de déterminer si l'algorithme a effectué une opération de poussage, si ce n'est pas la cas, la procédure effectue un réétiquetage. La variable f quant à elle permet de déterminer s'il s'agit là de la première itération, afin de pouvoir initialiser correctement m (qui est utilisé pour rechercher le plus petit des voisins du nœud effectuant l'algorithme).

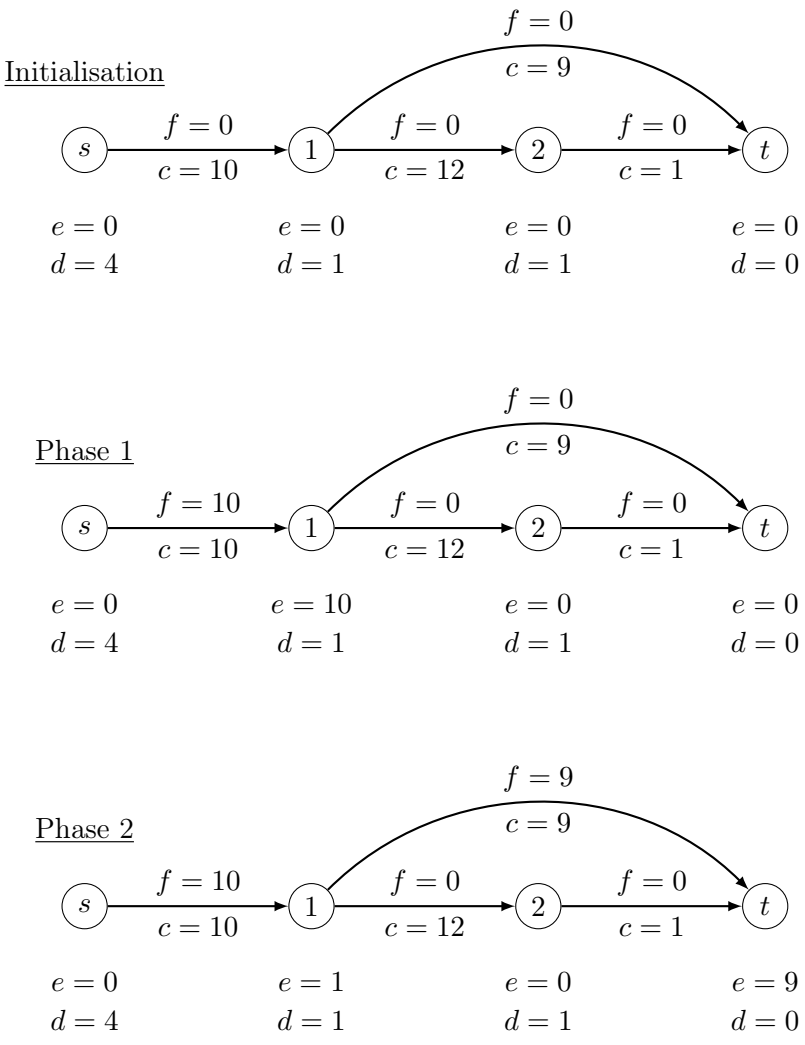


FIGURE 3 – Exécution d'un algorithme n'utilisant que les méthodes de poussage

Algorithm 3 Procédure Pousser-Réétiqueter

```
1: p ← false
2: f ← true
3: for j ∈ A(i) do
4:    if (i, j) est une arête acceptable then
5:       x(i, j) ← min(e(i), r(i, j))
6:       p ← true
7:       BREAK
8:    else if p = false and d(i) < d(j) then
9:       if r(i, j) > 0 and f = true then
10:         m ← d(j) + 1
11:         f ← false
12:      else if r(i, j) > 0 then
13:         m ← min(m, d(j) + 1)
14:      end if
15:   end if
16: end for
17: if p = false then
18:   d(i) ← m
19: end if
```

Il sera démontré plus loin que cet algorithme s'exécute en $O(n^2m)$ [5].

3.2 Validité

Théorème 3.1 *L'algorithme de préflot générique se termine [3].*

Pour montrer que l'algorithme de préflot générique résout le problème du flot maximum, Nous allons démonter que s'il se termine, Alors le préflot f calculé est un flot maximum !.

Passage par les opérations de poussage et réétiquetage :
Considérons un graphe $G = (S, A)$ de source s et de puits t, un préflot f et une fonction de distance valide d. Soit u un sommet de G, on a alors :

Lemme 3.2.1 *Si u est débordant, alors il peut être soumis à une opération soit de poussage, soit de réétiquetage.*

Preuve
Soit (u,v) un arc résiduel quelconque, donc on a $d(u) \leq d(v) + 1$ car d est une fonction de distance . Si une opération de poussage ne peut s'appliquer à u, alors pour tous les arcs résiduels (u, v), on doit avoir $d(u) < d(v) + 1$, ce qui implique que $d(u) \leq d(v)$. Donc une opération de réétiquetage peut être appliqué à u.

Retour à d la fonction de distance :

Lemme 3.2.2 *A n'importe quelle itération de l'algorithme, la fonction d est une fonction de distance valide.*

Preuve
On raisonne par récurrence sur le nombre d'opérations élémentaires effectuées. Après l'initialisation au départ, d est bien une fonction de distance valide.

Obeservons l'influence des opérations sur cette dernière :

- Le réétiquetage : Comme nous l'avons déjà vu. On assure que si d est une fonction de distance valide, alors elle le restera après une opération de réétiquetage de u. Considérons un arc $(u, v) \in A_f, \quad u, v \neq s, t$, par définition, on a $d(u) \leq d(v) + 1$, envisageons deux cas :
  1. $d(u) \geq d(v)$: l'opération de réétiquetage appliquée à u augmentera la valeur de $d(u)$ d'au moins 1 et donc après l'opération, on aura toujours $d(u) \geq d(v)$
  2. $d(v) = d(u) + 1$: l'opération de réétiquetage appliquée à u augmentera la valeur de $d(u)$ d'au moins 1 et donc après l'opération, on aura toujours $d(u) \geq d(v)$

  L'opération de réétiquetage conserve la validité de la fonction de distance.
- Le poussage : Considérons les sommets u et v tels que u pousse vers v. Nous envisagerons deux cas :
  1. le poussage crée l'arête (v, u) dans A_f : par définition du poussage, $d(u) = d(v) + 1$ donc d est toujours valide
  2. le poussage supprime l'arc (u, v) dans A_f : la contrainte correspondante est supprimée et donc d est toujours un fonction de distance valide

Lemme 3.2.3 *Il n'existe aucun chemin allant de la source s au puits t dans le réseau résiduel G_f.*

Preuve
Supposons le contraire, c'est-à-dire qu'il existe un chemin élémentaire $\mu = \{v_0, ..., v_k\}$ avec $v_0 = s$ et $v_k = t$ de s vers t dans A_f. μ élémentaire donc $k < | s |$ Soit $i = \{0,1,...,k-1\}$, l'arc $(v_i, v_{i+1}) \in A_f$. Comme d est une fonction de distance $d(v_i) \leq d(v_{i+1}) + 1$ encore pour $i = \{0,1,...,k-1\}$, en appliquant ces inégalités sur tout le chemin μ, on aura $d(s) \leq d(t) + k$. Mais comme $d(v_k) = d(t) = 0$, on obtient $d(s) = k < | s |$, ce qui contredit la contrainte $d(s) = | s |$ pour une fonction de distance

Théorème 3.2 *Si l'algorithme de préflot se termine, Alors le préflot f calculé est un flot maximum.*

Preuve
Regardons la boucle 'Tant que' dans l'algorithme préflot générique A chaque exécution de test de cette boucle, f est un préflot.

Initialisation : Initialiser-préflot fait de f est un préflot.

Conservation : Les seules opérations faites dans la boucle 'tant que' sont soit l'opération de poussage, soit celle de réétiquetage. L'opération de réétiquetage modifie uniquement la distance du sommet sur lequel elle est appliquée, donc elle ne modifie pas la valeur du préflot. De plus, si f est un préflot avant poussage il le reste après et donc, les opérations conserve le préflot f.

Terminaison :A la fin de l'exécution de l'algorithme, chaque sommet de S-$\{s,t\}$ doit avoir un excédent nul, en effet d'après les lemmes 5.0.1 et 5.0.2 et l'invariant selon lequel f est toujours un préflot, il n'y a pas de sommets débordants, or comme d est une fonction de distance valide, le lemme 5.0.3 nous dit qu'il n'y a pas de chemins entre la source s et le puits t dans le réseau résiduel A_f. Alors d'après le théorème du flot maximum et la coupe minimum le préflot f calculé est un flot maximum.

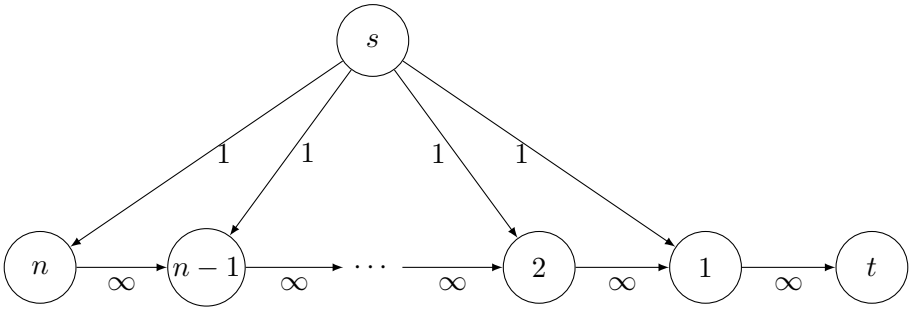


FIGURE 4 – Exemple de graphe mettant en défaut l'algorithme FIFO

4 Les algorithmes dérivés

Nous présenterons dans cette section, les différents algorithmes reposant sur l'algorithme générique présenté précédemment. Pour la totalité d'entre eux, il s'agit de chercher à réaliser un choix judicieux pour l'examen des nœuds débordants.

4.1 L'algorithme FIFO

L'algorithme FIFO, utilise une file afin de gérer l'ordre des nœuds actifs à traiter : au moment ou un sommet du graphe devient débordant, celui-ci est placé dans une file, dont la politique de traitement est du type *First In First Out*.

A chaque examen, l'algorithme réalise des opérations de poussage tant que celles-ci sont possibles et tant que le nœud i examiné est débordant. Si, lorsqu'il n'existe plus d'arêtes acceptables, $e(i)$ est toujours supérieur à 0, l'algorithme réalise donc un réétiquetage et place à nouveau i dans la file des sommets à examiner. L'initialisation de l'algorithme est la procédure principale restent les mêmes que l'algorithme générique.

Ce choix permet une amélioration de la complexité de l'algorithme générique en $O(n^3)$ [1].

4.2 L'algorithme High label

Il existe des graphes pour lesquels, l'algorithme FIFO est très lent. On peut par exemple citer celui décrit figure 4.

En effet, lors de l'initialisation de ce dernier, les nœuds vont être placés en file d'attente selon l'ordre de traitement des arêtes sortant de s. Si celui ci se fait par ordre lexicographique, le sommet 1 sera placé dans la file en premier puis le 2 et ainsi de suite jusqu'au sommet n. L'examen se fera donc dans cet ordre, on s'aperçoit alors que 1 pousse son excédent vers t, puis 2 pousse son excédent vers 1 et ainsi de suite jusqu'au nœud n poussant vers $n-1$. Puis le traitement recommence à partir de 1 jusque $n-1$, puis de 1 jusque $n-2$, etc... On dénombre alors $n!$ opérations.

Le principe de l'algorithme High Label est de choisir pour l'examen, le nœud ayant la distance la plus grande, de lui appliquer le plus grand nombre de poussage et de le réétiqueter si besoin est. L'algorithme effectue, sur le graphe représenté figure 4, seulement n opérations. Nous démontrerons que la complexité de cet algorithme s'en voit améliorée pour atteindre du $O(n^2\sqrt{m})$ [1].

Troisième partie
Complexité

5 Algorithme Générique

Nous nous intéressons à présent à la complexité de l'algorithme générique (algorithme 1 p. 7). Afin de pouvoir calculer cette dernière, nous allons avoir besoin de quelques bornes utiles.

Lemme 5.0.1
Chaque opération de réétiquetage d'un nœud augmente strictement sa distance d au nœud puits.

Preuve
Considérons un nœud actif i soumis à une opération de poussage-réétiquetage. Supposons, à présent, qu'il existe un nœud j tel que après réétiquetage on ait : $d_{new}(i) \leq d_{old}(i)$. Ceci n'est possible que si $d(j) \leq d(i) - 1$ et si r_{ij} [5] ≥ 0, on aurait alors $(i, j) \in A_f$ ce qui signifie alors que l'algorithme aurait effectué une opération de poussage et non un réétiquetage. Il est donc impossible de trouver un nœud j tel que l'opération de réétiquetage diminue sa distance d.

Lemme 5.0.2
A n'importe quelle itération de l'algorithme, tout nœud actif i est relié au nœud source s par un chemin de i vers s dans le réseau résiduel.

Preuve
D'après le théorème de décomposition des flots [6], il est possible de décomposer n'importe quel préflot en des flots positifs le long de chemins reliant s à t [7], s aux nœuds actifs et le long de cycles. De plus par définition d'un préflot on a : $e(s) \leq 0$ et $e(i) \geq 0\ \forall i \in S - \{s\}$.

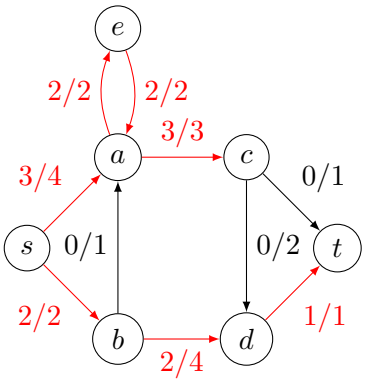
Considérons un nœud actif i, l'excès de flots $e(i)$ est strictement supérieur à 0. Or, lors de la considération des flots obtenus par le théorème de décomposition des flots, on observe que les flots pris le long des cycles et des chemins de s à t n'interviennent pas dans l'excédent de flot de i, ce qui implique qu'il existe un flot strictement positif coulant le long d'un chemin de s à i et donc que l'inverse de ce chemin se trouve dans le réseau résiduel. D'où l'existence d'un chemin de i vers s dans le réseau résiduel.

On peut directement déduire de ce lemme que, pour un nœud actif i, il existe toujours un nœud de son voisinage j tel que $d(j) \geq d(i)$ et donc que lors d'un réétiquetage, la minimisation de la hauteur des voisins ne se fait pas sur un ensemble vide. Autrement dit, pour un nœud que l'on réétiquette, il existe toujours un sommet de son voisinage dont la distance au puits est supérieure au nœud réétiqueté.

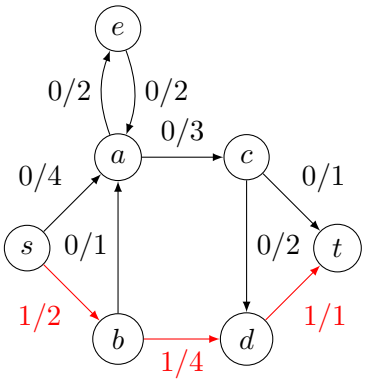
Lemme 5.0.3
Pour tout $i \in S,\ d(i) < 2n$

Preuve
Une opération de réétiquetage d'un sommet i n'affecte que la distance de i au nœud puits, par conséquent l'excédent de flot sur le nœud i après l'opération est le même qu'avant. D'après le lemme 5.0.2, il existe donc un chemin P de s à i [8] dans le réseau résiduel dont la longueur est

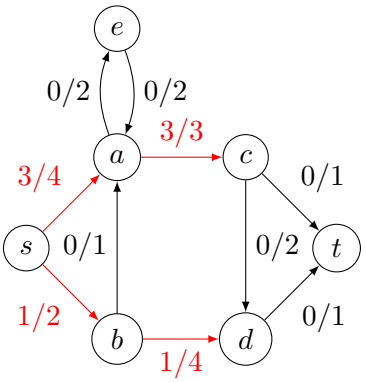
5. La capacité résiduelle de l'arc (i, j)
6. Un exemple d'application de ce théorème est donné à la figure 5
7. Le nœud puits
8. $i \neq t$



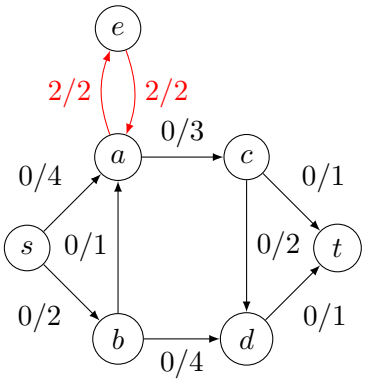
(a) Un préflot



(b) La décomposition en flots de s vers t



(c) La décomposition en flots de s vers les nœuds actifs



(d) La décomposition en cycles

FIGURE 5 – Un exemple de décomposition de préflot, d'après le théorème de décomposition des flots

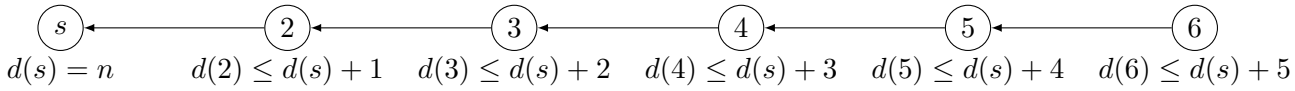


FIGURE 6 – Chemin de longueur $n-2$ sommets dans le graphe résiduel

au plus $n-2$ sommets[9]. À la fin de l'initialisation : $d(s) = |S| = n$, et pour chaque arc (k,l) du chemin de i à s, on a : $d(k) \leq d(l)+1$. Prenons un exemple concret, soit G un graphe à 8 sommets de sorte qu'après un certains nombre d'itérations de l'algorithme, le nœud 6 est actif[10]. Le chemin de 6 vers s est représenté à la fig. 6.

En partant de $d(s) = n$, on peut écrire :

$$d(2) \leq d(s) + 1$$

Or on a aussi :

$$\begin{array}{rrl} & d(3) & \leq \quad d(2)+1 \\ \Rightarrow & d(3) & \leq \quad d(s)+2 \end{array}$$

Donc pour le $n-2^e$ sommet on a :

$$\begin{array}{rrl} & d(n-2) & \leq \quad d(s)+(n-3) \\ \Rightarrow & d(n-2) & < \quad d(s)+n \\ \Rightarrow & d(n-2) & < \quad 2n \end{array}$$

Et comme il s'agit du pire des cas, on peut généraliser : $\forall i \in S, \quad d(i) < 2n$. Si l'on prend en compte le fait qu'à chaque opération de réétiquetage, la distance d est augmentée au minimum d'une unité, on obtient le lemme 5.0.4.

Lemme 5.0.4
Un nœud i subit au maximum $2n$ opération de réétiquetage, ce qui implique que le nombre total d'opération de réétiquetage est borné par $2n^2$

Lemme 5.0.5
Le nombre de poussages saturants[11] *exécutés pendant l'algorithme est inférieur à $2nm$.*

Preuve
Considérons deux sommets voisins i et j. Si l'on considère un poussage saturant de i vers j (respectivement de j vers i), alors l'arête (j,i) (resp. (i,j)) est une arête de A_f. Pour pouvoir effectuer le poussage saturant inverse, j (resp. i) doit augmenter sa distance de : $d(j) - d(i) + 1$ (resp. $d(i) - d(j) + 1$) ≥ 2. Donc chacun des sommets peut au maximum effectuer n poussages saturants vers l'autre[12]. Ce qui nous donne pour chaque arc, un nombre de poussages saturants inférieur à $2n$. Sur l'ensemble de l'algorithme ce dernier est bien majoré par $2nm$.

Lemme 5.0.6
Le nombre de poussage non saturants lors de l'exécution de l'algorithme est borné supérieurement par $O(n^2m)$.

9. Le chemin ne peut inclure t, de plus i ne peut être un voisin direct de t, sinon il n'y aurait pas eu d'opération de réétiquetage.
10. Il s'agit bien du pire des cas : une chaîne de $n-2$ sommets
11. Un poussage de i vers j est dit saturant si après poussage, $r((i,j)) = 0$
12. La distance de chaque sommet est bornée ainsi : $0 \leq d(i) \leq 2n$, le nombre maximum d'augmentation de d de 2 unités est donc n.

Preuve
Considérons la fonction Φ qui calcule la somme des distances des nœuds actifs :

$$\Phi = \sum_{i \in \{i \in S / e(i) > 0\}} d(i)$$

. Comme le nombre de nœuds actifs est inférieur à $|S|$ et que la distance de chaque somme est bornée par $2n$, la valeur initiale [13] de Φ est inférieure à $2n^2$, à la fin de l'algorithme, il n'existe plus de sommets actifs, et donc : $\Phi = 0$. Lors du déroulement de l'algorithme, trois choses peuvent se produire :

1. Poussage non saturant. Il a lieu lorsque l'excédent de préflot du sommet est inférieur à la capacité résiduelle de l'arc emprunté, donc une fois cette étape effectuée le nœud i ayant effectué le poussage n'est plus actif. Dans le cas ou j, le sommet cible du poussage n'appartenait pas à l'ensemble des nœuds débordants I, la fonction Φ est incrémentée de $d(j)$ et décrémentée de $d(i)$. Or $d(i) \geq d(j) + 1$, donc Φ diminue d'au moins une unité.
2. Poussage saturant. Il a lieu lorsque $e(i) \geq r(i, j)$, Dans le pire des cas, $j \notin I$ avant l'opération, on a alors une augmentation maximale de Φ égale à la hauteur max de j, soit $2n$ donc une augmentation de $2n^2m$ pour l'ensemble des poussages saturants.
3. Réétiquetage. Cette opération n'a lieu que lorsqu'aucun arc sortant de i n'est valide pour une opération de poussage, il est évident que chaque opération de réétiquetage augmente Φ de $2n$ unités au maximum par nœud [14], soit une augmentation totale de n^2 unités sur l'ensemble de l'algorithme.

La seule opération permettant de diminuer la valeur de Φ est un poussage non saturant, or la valeur finale de Φ est zéro, on en déduit donc que la valeur maximale de Φ nous donne le nombre de poussages non saturants. La valeur maximal de Φ nous est donnée par la somme des bornes de la valeur initiale et des augmentations dues aux opérations de poussages saturants et de réétiquetage soit :

$$\Phi = n^2 + n^2 + 2n^2m = 2n^2(m + 1) = O(n^2m)$$

On en déduit donc le théorème suivant :

Théorème 5.1 *L'algorithme de préflots générique s'exécute en un temps $O(n^2m)$.*

6 Algorithme FIFO

Les améliorations sur l'algorithme générique se font soit grâce à l'amélioration de la structure de donnée, soit à l'aide d'une amélioration sur la manière de sélectionner les nœuds actifs dans le réseau de départ.

Définissons à présent la notion d'*analyse de nœud*. Au cours de l'algorithme précédent, dans le cas d'une opération de poussage saturant, nous avons vu que le nœud réalisant l'opération reste, très probablement, un nœud actif, il est donc candidat à la sélection lors de la prochaine boucle de l'algorithme. Seulement rien ne garantit qu'il sera à nouveau choisi, il peut donc être intéressant dans ce genre de situation de réaliser plusieurs poussages saturants suivis soit d'un poussage non saturant, soit d'une opération de réétiquetage. C'est cette suite d'opérations que nous appellerons dorénavant *analyse de nœud*.

13. Après initialisation.
14. Limite donnée par la hauteur maximale d'un nœud.

Théorème 6.1 *L'algorithme de préflot FIFO s'exécute en un temps $O(n^3)$.*

Preuve
Afin de calculer la borne supérieure du temps d'exécution de l'algorithme, nous allons être amenés à introduire de le concept de phase. On appelle première phase, la partie de l'algorithme qui consiste à appliquer une analyse de nœud à chacun des sommets devenus actifs suite aux opérations de poussage effectuées sur s. Durant cette première phase, les nœuds devenus actifs suite aux analyse de nœud des sommets de la première phase, sont mis en queue de file et leur analyse constituera les opérations de la seconde phase, et ainsi de suite. L'objectif est donc de borner le nombre de phases.

Considérons la fonction $\Phi = max\{d(i) : i \text{ actif }\}$. Considérons à présent l'effet d'une phase sur la fonction Φ, on remarque alors plusieurs cas :

1. Lors de la phase aucune opération de réétiquetage n'a lieu. Dans ce cas, les sommets i de cette phase ont tous effectués une opération de poussage non saturant, ils ne sont donc plus actifs. Les nœuds actifs après la phase j ont tous une distance respectant cette loi : $d(j) < d(i) \quad \forall i, j$. Φ diminue alors d'au moins une unité.
2. Lors de la phase il y a au moins une opération de réétiquetage. Φ voit alors sa sa valeur augmenter de δ avec $\delta \geq 0$. Si $\delta = 0$, alors il existe un nœud qui voit sa distance augmenter au moins d'une unité, si $\delta > 0$ alors il existe un nœud dont la distance augmente d'au moins δ. De par le lemme 5.0.3 le nombre de phases pendant lesquelles Φ est inférieure à $2n^2$.

Comme la valeur initiale de Φ est n, le nombre de phases pendant lesquelles Φ diminue est donc bornée supérieurement par : $2n^2 + n$, le nombre total de phases est donc borné par $4n^2 + n$. De plus comme il y a au moins une opération de poussage non saturant par phase, le nombre de poussage non saturants est borné par $4n^3 + n$. L'algorithme s'exécute donc en $O(n^3)$.

7 Algorithme High Label

Comme vu précédemment, l'algorithme High Label analyse les nœuds en fonction de la valeur de leur fonction de distance. Cette sélection permet d'obtenir une borne maximale en $O(n^2\sqrt{m})$ [1], c'est ce que nous allons démontrer.

Soit une constante $K = \sqrt{m}$, soit la fonction $d' : S \longrightarrow \mathbb{N}$, définie par :

$$d'(i) = |\{j : d(j) \leq d(i)\}| \quad \forall i, j \in S$$

Autrement dit, $d'(i)$ représente le nombre de nœuds de S étant plus proche du puits que i. Cette fonctions vérifie alors quelques propriétés :

Lemme 7.0.7

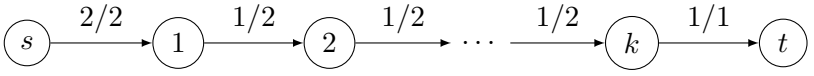
$$\forall i \in S, \quad d'(i) \leq n$$

Preuve :
L'ensemble des sommets dont la distance est inférieure à une valeur donnée est un sous-ensemble de l'ensemble des sommets du graphe. Son cardinal est donc majoré par ce dernier.

Introduisons la fonction de potentiel Φ définie comme suit :

$$\Phi = \sum_{i:e(i)>0} \frac{d'(i)}{K}$$

On peut remarquer qu'à n'importe quel moment de l'exécution de l'algorithme, $\Phi \geq 0$ et que après l'initialisation : $\Phi \leq n^2/K$. Cette borne est atteinte dans le cas d'un graphe complet



(a) Le graphe de flots

i	$d(i)$	$e(i)$	$d'(i)$
s	$k+2$	∞	$k+1$
1	$k+1$	1	1
2	$k+2$	0	$n+1$
...	...	...	...
n	$k+2$	0	$k+1$
t	0	1	0
			$\Phi = 1/K$

(b) Excédents, distances et fonctions de potentiel

FIGURE 7 – Exemple de l'influence du réétiquetage sur la fonction Φ

puisque chaque nœud ayant une distance de 1 et étant relié à la source, on obtient alors n nœuds actifs vérifiant tous $d'(n) = n$, d'où $\Phi = n^2/K$.

Regardons les effets des opérations de poussage et réétiquetage sur la fonction Φ.

Lemme 7.0.8 *Une opération de réétiquetage augmente la valeur de Φ d'au plus n/K.*

Preuve :
Dans le pire des cas, le sommet réétiqueté passe de la plus petite distance du graphe à la plus grande. Une représentation d'un de ces cas est donnée à la figure 7. Il s'agit là d'une chaîne dont la capacité de chacun des arcs est supérieure à la capacité du dernier arc de la chaîne, de façon à ce que l'on observe un phénomène de flux et reflux influant sur la distance de tous les sommets. Lors du dernier réétiquetage de l'algorithme on se retrouve dans le cas représenté. Aucun autre nœud n'étant actif, on a, avant réétiquetage, $\Phi = d'(1)/K = 1/K$, or après réétiquetage, le sommet 1 est le sommet présentant la plus haute distance du graphe. On observe dans ce cas une augmentation de k/K soit $(n-2)/k$.

Lemme 7.0.9 *Une opération de poussage saturant augmente la valeur de Φ d'au plus n/K.*

Preuve :
Chaque opération de poussage non saturant rend actif, au plus un seul sommet supplémentaire, appelons k ce sommet. Or on sait que $d'(k) \leq n$, donc l'augmentation de Φ par l'ajout d'un sommet actif est bornée par n/K.

Lemme 7.0.10 *Une opération de poussage non saturant n'augmente pas la valeur de Φ.*

Preuve :
Soient i et j deux sommets tels qu'il est possible de réaliser un poussage non saturant de i vers j. Après cette opération, le sommet i n'est plus actif et le sommet j le devient, or $d(i) = d(j)+1$ donc $d'(i) > d'(j)$, on observe même une diminution de la valeur de Φ d'au moins $1/K$.

Afin de quantifier le nombre de poussage non saturant, l'algorithme est scindée en phases définies comme suit : une phase est l'ensemble des opérations de poussage ayant lieu entre deux changements consécutifs de la valeur d^*. On définira d^* ainsi :

$$d^* = \max\{d(v) : v \text{ est actif }\} \tag{12}$$

Autrement dit il s'agit de l'ensemble des poussages ayant lieu sans qu'il y ait changement de la valeur maximale des distances des nœuds. Une phase présentant un nombre d'opérations de poussage non saturant inférieur à K sera dite *faible*, elle sera désignée comme *forte* sinon.

Lemme 7.0.11 *Le nombre de phases est inférieur à* $4n^2$

Preuve :
Au début et à la fin de l'algorithme, $d^* = 0$ dans la mesure où il n'existe aucun sommet actif dans le graphe. De plus la seule opération capable d'augmenter la valeur de d^* est un réétiquetage. Puisque d^* augmente au plus de $2n^2$ [15], sa valeur diminue au plus $2n^2$, on a alors au plus $4n^2$ changements, donc autant de phases.

Corollaire 7.0.1 *Le nombre maximum de poussages non saturants en phase* faible *est borné par* $4n^2K$.

Ceci est directement donné par la limite du nombre de phases : à supposer que toutes les phases sont des phases faibles, présentant au plus K poussages non saturants. Il y a alors, au maximum, $4n^2K$ poussages non saturants sur l'ensemble de l'algorithme.

Lemme 7.0.12 *Une phase forte présentant* $Q > K$ *poussages non saturants diminue la valeur de* Φ *d'au moins* Q.

Preuve :
Considérons une phase forte au cours de laquelle sont effectuées Q poussages non saturant. Dans la mesure où chacune de ces opération a lieu dans la même phase, la valeur de d^* est constante, ce qui implique que chacune de ces opérations est faite depuis un nœud i tel que $d(i) = d^*$.

La phase s'achève lorsque, tous les sommets de distance d^* sont sortis de leur état actif, ou lorsque l'un d'entre eux est réétiqueté impliquant une nouvelle valeur de d^*. Dans les deux cas, puisqu'il s'agit d'une phase forte, il y a au moins Q nœuds, qui ont vu leur excédent chuter à zéro, réduisant ainsi la valeur de Φ d'au moins Q/K à chaque poussage non saturant [16]. Or $Q > K$ donc chaque poussage diminue Φ d'au moins 1 et donc la phase diminue Φ d'au moins Q.

Théorème 7.1 *L'algorithme High Label s'exécute en* $O(n^2\sqrt{m})$.

L'augmentation maximale de la fonction de potentielle Φ est égale à $(2n^2 + 2nm)n/K$ [17]. Donc la valeur maximale de Φ se résume à sa valeur initiale maximale (n^2/K) à laquelle est ajoutée l'augmentation maximale $(2n^2 + 2nm)n/K$, on obtient alors :

$$\Phi_{\max} = \frac{2n^3 + n^2 + 2n^2m}{K}$$

Il s'agit là aussi de la borne supérieure du nombre de poussages non saturant qui ont lieu lors des phases fortes. En prenant en compte le nombre de poussages saturant maximum lors des phases faibles, on obtient :

$$N \leq \frac{2n^3 + n^2 + 2n^2m}{K} + 4n^2K \leq \frac{3n^3 + 2n^2m}{K} + 4n^2K = \frac{3n^3 + 2n^2m}{\sqrt{m}} + 4n^2\sqrt{m} = \frac{3n^3\sqrt{m}}{m} + 6n^2\sqrt{m}$$

Le nombre de poussages non saturant est en $O(n^2\sqrt{m})$ et donc l'algorithme s'exécute de la même manière en $O(n^2\sqrt{m})$.

15. Cette borne provient du nombre maximal de réétiquetage
16. Ceci est dû au fait qu'il y a au moins Q nœud ayant la même distance que le sommet effectuant le poussage. Si l'on appelle i ce sommet $d'(i) \geq Q$
17. Il y a au plus An^2 réétiquetage et $2nm$ poussages saturant augmentant la valeur de Φ d'au plus n/K

Quatrième partie

Tests

8 Modus Operandi et implémentation

Le développement a été réalisé sous vim en langage C, afin de profiter de la vitesse d'éxécution ainsi que des optimisations de gcc à la compilation. D'un point de vue collaboratif, git a été choisi pour sa facilité d'utilisation et la possibilité d'héberger le code source et les documents sur github.

Au niveau des choix d'implémentation, nous avons codé les graphes de manière à ce qu'il soit possible d'utiliser une réprésentation en mémoire sous forme de matrice ou de liste d'adjacence [18]. Chacun des algorithmes utilisés requiérant un accès rapide aux voisins d'un nœud, nous avons choisi de représenter les graphes sous forme de liste.

Nous avons créé un générateur de graphe aléatoire, garantissant la connexité du graphe généré. Il requiert pour s'éxécuter un nombre de nœuds et un nombre d'arêtes, et, à partir de ces données, crée dans un premier temps un arbre contenant l'ensemble des nœuds du graphe puis par une loi de probabilité défini le nombre de voisins de chacun des nœuds en s'assurant d'atteindre exactement le nombre d'arêtes fixées.

Avec l'aide du générateur il est donc possible de générer des graphes de densité fixée. Cette démarche permet l'expression du nombre d'arêtes en fonction du nœud et vice versa et donc la vérification des bornes calculées théoriquement précédemment.

Le programme de test génère un nombre donné de graphes aléatoires de densité fixée et applique chacun des algorithmes sur ces graphes. Il calcule ensuite la moyenne des temps d'exécutions.

9 Les résultats des tests

9.1 L'évolution du temps d'éxécution en fonction du nombre de nœuds

Nous présenterons ici des résultats pour, respectivement, une densité faible, moyenne et grande.

9.1.1 Densité $d = 2$:

Pour ce genre de densité, on a le nombre d'arêtes qui est dû même ordre de grandeur que le nombre de nœuds ce qui nous donne les complexités suivantes :

- Dinic $O(n^3)$
- FIFO $O(n^3)$
- High Label $O(n^2\sqrt{n})$

Malgré une complexité du même ordre de grandeur, l'agorithme FIFO s'exécute plus rapidement que l'algorithme de Dinic, ce qui est dû au fait que la borne supérieure de l'algorithme FIFO est très rarement atteinte et que les cas où ce dernier est dit "lent" sont des cas pathologiques. On peut remarquer aussi que l'algorithme High Label n'est un pas plus rapide que l'algorithme FIFO, contrairement à ce que l'on pouvait escompter. Il est possible d'expliquer cette "mauvaise" performance par un choix non adapté de structure de données.

18. Le choix de la représentation en mémoire est laissée à l'utilisateur et est défini par les procédures utilisées pour la création des graphes.

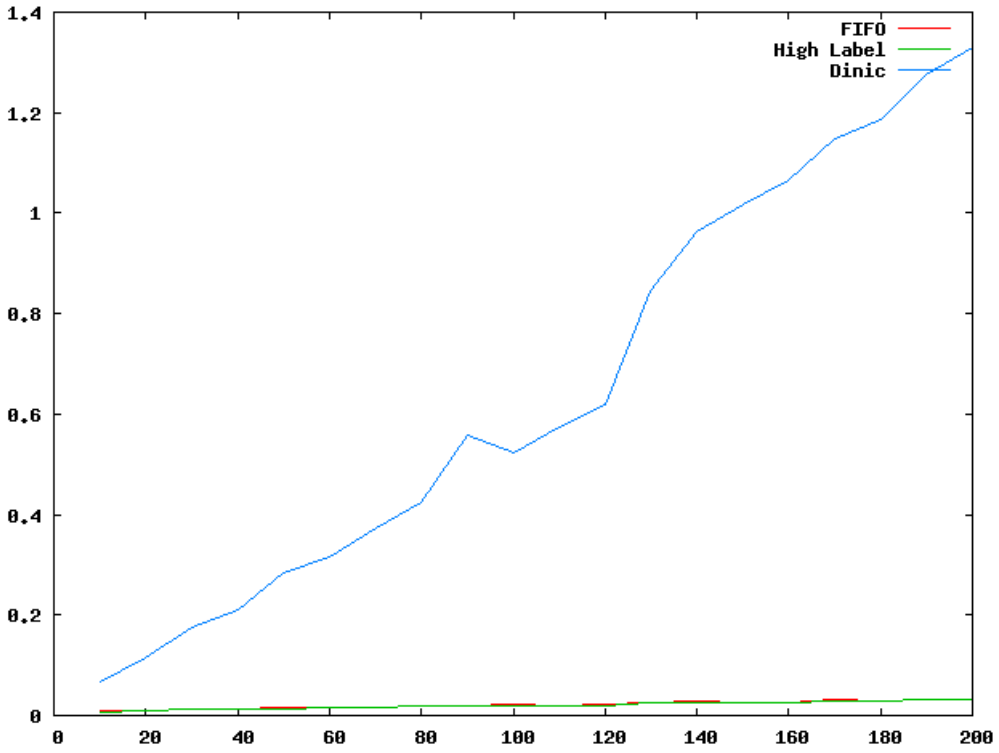


FIGURE 8 – Evolution du temps d'exécution des algorithmes en fonction du nombre de nœuds (densité = 2)

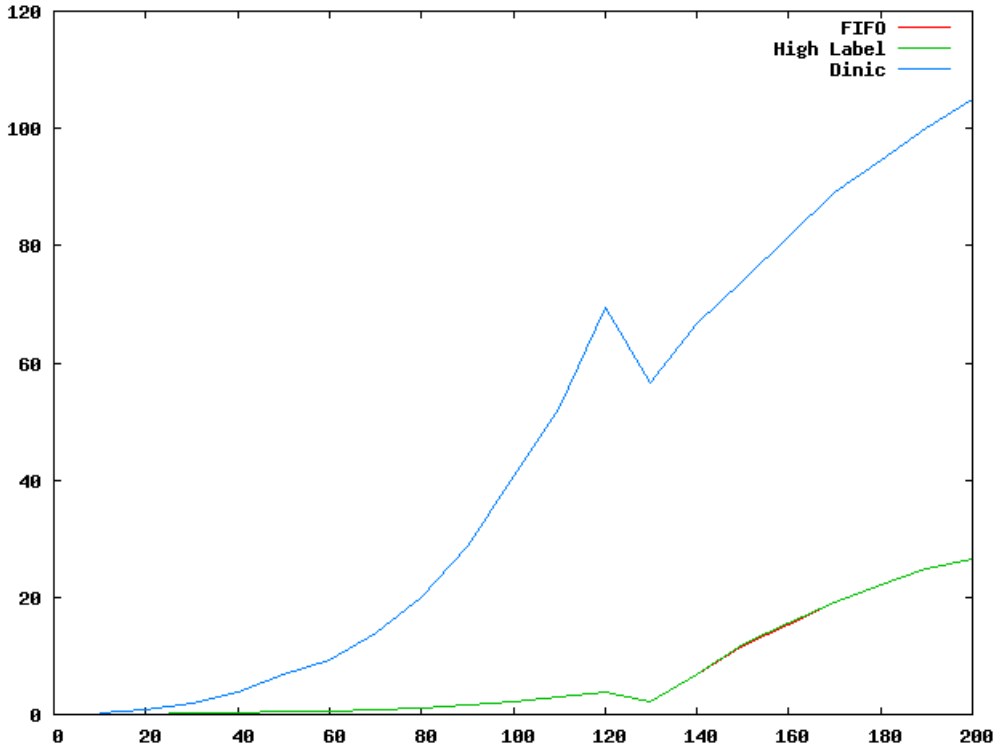


FIGURE 9 – Evolution du temps d'exécution des algorithmes en fonction du nombre de nœuds (densité = 130)

En effet, lors du calcul de la complexité de ce dernier, nous avons estimé (implicitement) que le temps d'accès au nœud de plus grande distance n'influe pas sur la complexité de ce dernier. Or ceci n'est vrai que si la structure de données utilisée pour stocker les sommets actifs est correctement choisie. Lors de l'implémentation, nous nous sommes tourné ves un tas permettant un accès en temps constant à la donnée recherchée. Seulement l'insertion et la suppression d'un nœud se font en $O(\log(n))$, ce qui augmente le temps d'exécution. Un choix judicieux aurait été un tableau de listes chaînées [1] t et l'utilisation d'une variable k sauvegardant une borne supérieure de la distance des nœuds. Ainsi, pour la sélection du nœud de plus grand distance, le tableau est parcouru en commençant par $t[k], t[k-1], \ldots$, ce qui permet une insertion et une suppression plus rapide.

9.1.2 Densités $d = 130$ et $d = 200$

Lorsque la densité est grande, le nombre d'arêtes du graphe devient de l'ordre de n^2. On peut alors espérer obtenir des complexités en $O(n^4)$ pour Dinic et $O(n^3)$ pour les algorithmes de préflots.

Ce sont effectivement les résultats obtenus lorsque $d = 200$, les courbes rajoutées correspondent aux approximations données par gnuplot. Elles sont de la forme $P_1(x) = ax^4 + bx^3 + cx^2 + dx + e$ pour Dinic et $P_2(x) = ax^3 + bx^2 + cx + d$ pour les algorithmes de préflots. Il est important de noter que le coefficient a est positif pour les deux polynômes ce qui prouve que la croissance de la valeur du polynôme est de l'ordre du degré le plus élevé de ce dernier.

On observe, dans la majorité des cas, lorsque la densité est de l'ordre de $\frac{n}{2}$, une brisure dans la courbe du temps des algorithmes. Nous n'avons aucune explication pour cette dernière.

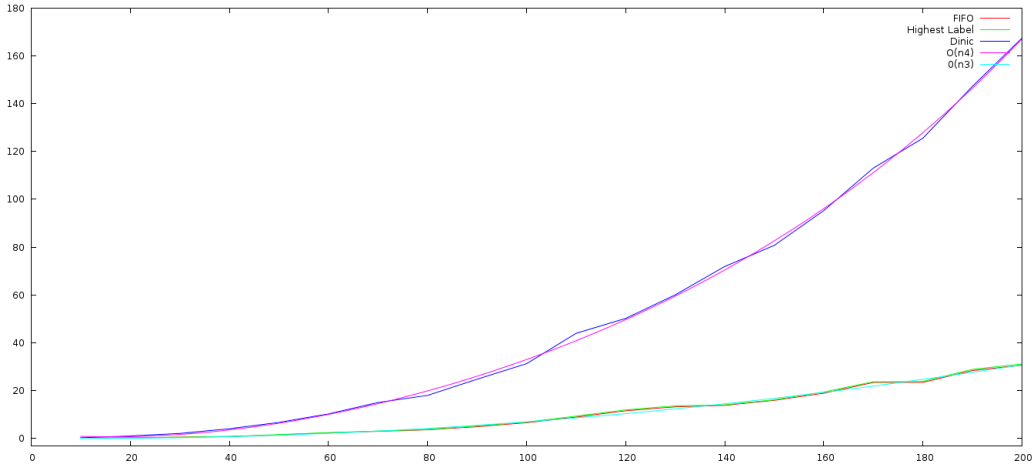


FIGURE 10 – Evolution du temps d'exécution des algorithmes en fonction du nombre de nœuds (densité = 200)

9.2 L'évolution du temps d'execution en fonction du nombre d'arêtes

Ici, le nombre de sommets du graphe est fixé [19] et l'on fait varier le nombre d'arêtes. On oberve, que l'influence du nombre d'arête sur le temps d'exécution est moindre pour les algorithmes de préflots ce qui confirme les résultats attendus.

Le point intéressant de ces résultats est la diminution du temps d'exécution pour des graphes dont la densité est élevée [20]. Ceci est dû au fait que les algorithmes de préflots sont très rapides pour ce genre de graphe puisque le préflot circule alors très rapidement vers les sommet puits diminuant considérablement le nombe d'évaluation des sommets actifs.

9.3 D'un point de vue général

Les résultats obtenus concordent pour la majorité avec les résultats attendus après une approche théorique. Il ne reste qu'à observer l'influence du choix de la structure de données pour l'algorithme High Label.

Ceci mis à part, les complexités théoriques sont confirmées par la pratique ainsi que les réactions des algorithmes aux différentes caractéristiques du graphe.

Cinquième partie
Conclusion

Comme vu précédemment, les algorithmes de préflots sont des algorithmes très rapides pour la résolution du problème de recherche de flot maximum. Et même s'il existe des méthodes dont la complexité théorique est inférieure à la leur [21], ils sont dans la pratique ceux offrant les meilleurs résultats ainsi qu'une grande facilité de mise en œuvre [22].

Mais au delà de cet aspect pratique, ils représentent une approche intuitive des algorithmes

19. $n = 100$
20. Proche de n.
21. On peut par exemple parler de l'algorihtme pousser vers l'avant avec un arbre dynamique
22. Relativement aux autres algorithmes dont les performances sont à peu de choses près équivalentes

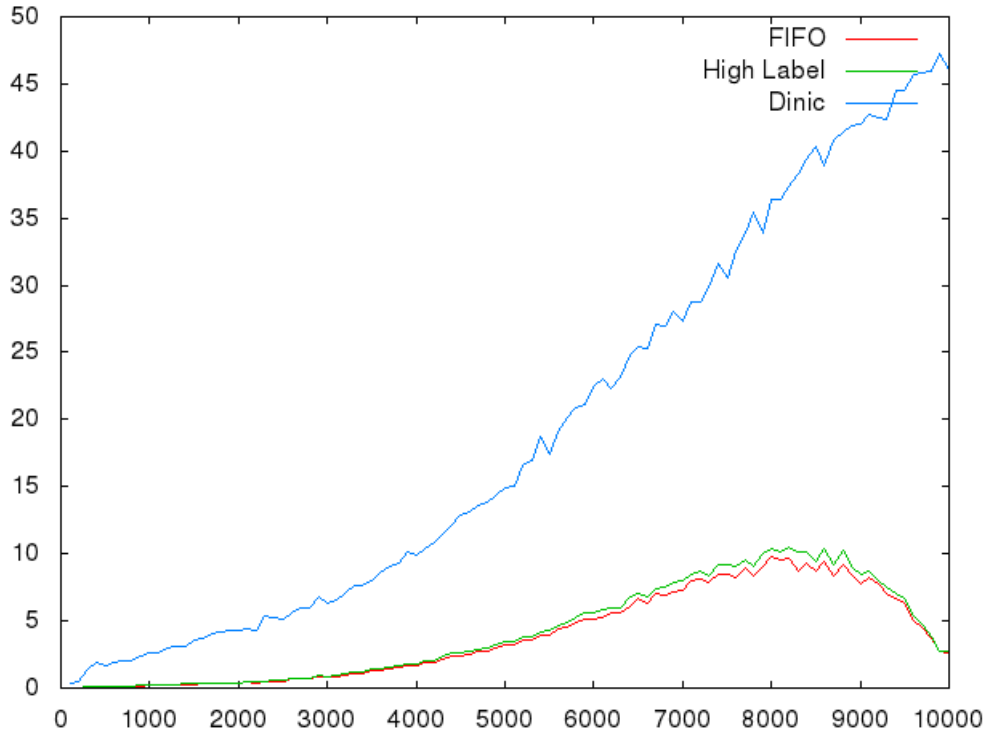


FIGURE 11 – Evolution du temps d'exécution des algorithmes en fonction du nombre d'arêtes (n=100)

de résolution de problème par relaxation de contraintes [23]. Ils sont en effet le parfait exemple de la puissance de ces méthodes, et mettent en avant combien il est coûteux de maintenir à n'importe quelle étape de l'algorithme une solution réalisable.

23. On peut par exemple citer dans cette famille, les méthodes arborescentes telles que le *Branch and Bound* ou *Branch and Cut*

Références

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows : Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] J. Cheriyan and K. Mehlhorn. An analysis of the highest-level selection rule in the preflow-push max-flow algorithm. *Information Processing Letters*, 69 :69–239, 1998.

[3] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction à l'algorithmique*. Dunod, 2001.

[4] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2) :248–264, April 1972.

[5] A. Goldberg and R. Tarjan. A new approach to the maximum flow problem. *Journal of the Association for Computing Machinery*, 35 :921 – 940, October 1988.

[6] A. Shrijver. On the history of the transportation and maximum flow problem. *International Symposium of the Mathematical Programming Society*, 91 :437 – 445, 2002.