Algorithme d'approximation pour le problème du k-fournisseur

Benoît. Rouits
Stage D.E.A. Informatique 1999
Université D'Evry-Val-d'Essonne

Responsable de stage : Christian Laforest

Table des matières

Introduction

De nombreux problèmes d'optimisation dans le monde réel s'expriment aisément dans des structures mathématiques telles que les graphes. Les graphes sont une structure simple composée de 2 ensembles : un ensemble de *sommets* ou points et un ensemble d'*arêtes* ou lignes. Les sommets et les arêtes peuvent être pondérés par plusieurs valeurs exprimant leurs caractéristiques. Par exemple, un réseau routier peut être décrit par un graphe dont les arêtes pondérées représentent les routes et leur coefficient de fréquentation et où les sommets représentent les croisements. On peut aussi transposer des topologies de réseau informatique sur les graphes où les sommets représentent les machines et où les arêtes représentent les liaisons physiques entre les machines, munies d'un coefficient de bande-passante par exemple.

Motivations

Dans le cadre des systèmes informatiques distribués et notament dans les environnements de travail de groupe [1], on est confronté à des problèmes de perfomance et de sûreté de l'infrastructure. On trouve entre autres des problèmes d'*élection* de sites particuliers jouant le rôle de lien entre les participants à l'environnement. Le problème exposé dans ce rapport est un problème d'élection : Considérons un réseau informatique constitué de sites et de liens de communication entre eux. Imaginons que plusieurs sites *clients* aient accès à des ressources informatiques communes. On aimerait pouvoir distribuer ces informations ou ces traitements sur un ensemble de sites *fournisseurs* tels qu'un client ait un fournisseur le plus proche possible de lui. On considère que l'on a un ensemble de sites proposé qui correspond aux fournisseurs potentiellement utilisables et que chaque fournisseur potentiel est marqué d'un coût d'utilisation. On veut alors trouver l'ensemble de fournisseurs dont le coût total est inférieur à une constante k donnée et qui permette à tout site client d'être le plus proche possible d'au moins un fournisseur élu. Ensuite, si de nouveaux clients arrivent ou bien si des clients se déconnectent, on veut pouvoir décider automatiquement de la migration des fournisseurs pour minimiser toujours la distance clients-fournisseurs.

Ce rapport propose un algorithme de faible complexité capable de donner une solution approchée à ce problème ; on le nommera par la suite le problème du k-fournisseur.

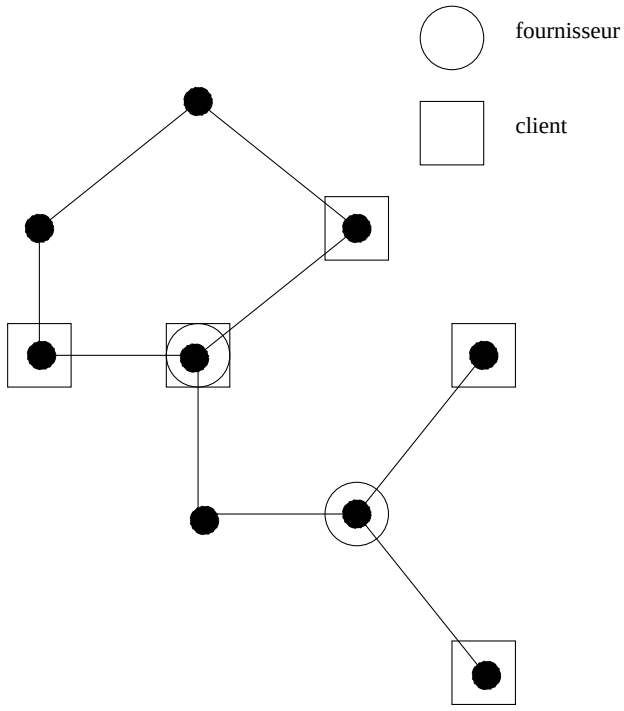


FIG. 1 – *un exemple de 2-fournisseur*

Modélisation

Pour donner un algorithme, nous allons d'abord formaliser le problème et donner un modèle de la structure sur laquelle il s'appuie. On modélisera le réseau par un graphe $G = (V, E)$ où V est l'ensemble des sites, et E l'ensemble des liens entre ces sites. On appelle V l'ensemble des *sommets* et E l'ensemble des *arêtes* du graphe. L'ensemble des sommets fournisseurs potentiellement utilisables est noté V_f et est inclus dans V. L'ensemble des clients est noté V_{cl} et est aussi inclus dans V. Le réseau correspond à un graphe quelconque. On peut obtenir à partir de ce graphe un graphe complet des distances entre tous les sommets. C'est sur ce graphe des distances que travailleront les algorithmes proposés. Une première partie développera le sujet du k-fournisseur statique, c'est-à-dire sans migration des fournisseurs, puis une seconde partie développera le sujet de la dynamicité. On pourra remarquer que les algorithmes utilisent un squelette générique[2] que l'on peut reprendre pour de nombreux problèmes.

1. courament appelé *groupware*
2. squelette d'algorithme *Bottleneck* proposé dans [5]

Chapitre 1

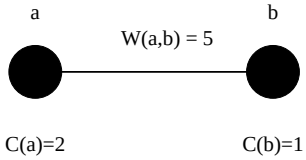
L'algorithme d'approximation pour le problème des fournisseurs

1.1 Contexte et définitions générales

Soit $G = (V, E)$ un graphe complet des distances où V est l'ensemble des sommets de G et E l'ensemble des arrêtes qui relient les sommets de V et dont les pondérations vérifient l'inégalité triangulaire. Nous définissons un ensemble de sommets *clients* $V_c \subseteq V$ et un ensemble de *fournisseurs potentiels* $V_f \subseteq V$. Tout sommet de V peut être à la fois client et fournisseur potentiel, ou seulement l'un des deux, ou encore ni l'un ni l'autre.

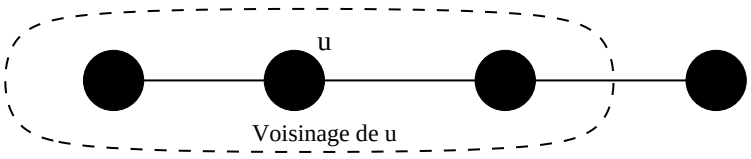
Au graphe $G = (V, E)$ sont associées deux fonctions :

- *Coût d'un sommet* de V_f : $C : V_f \to \mathbb{N}^+$
  Par extension, on notera le coût d'un ensemble $F \subseteq V_f$: $C(F) = \sum_{f \in F} C(f)$
- *Poids d'une arête* de E : $W : E \to \mathbb{N}^+$



Définition 1.1.1 ***Voisinage d'un sommet.***
On notera pour un sommet u dans un graphe quelconque $G = (V, E)$ *le* voisinage *de u comme étant :*

$$N_G(u) = \{v : (u, v) \in E\} \cup \{u\}$$



Définition 1.1.2 ***Excentricité d'un ensemble de fournisseurs.***
On associe à tout ensemble $F \subseteq V_f$ *une valeur* d'excentricité $Exc(F)$ *par rapport à l'ensemble des*

clients V_c *:*

$$Exc(F) = \max_{c \in Vc}[\min_{f \in F}[W(f,c)]]$$

Autrement dit, pour chaque sommet $c \in V_c$, on détermine le plus petit chemin pour aller à un fournisseur$f$ de F, puis on prend la valeur maximale de ces $|V_c|$ distances pour définir $Exc(F)$. Nous pouvons maintenant définir le problème du k-fournisseur optimal :

Problème 1.1.3 ***k-fournisseur optimal.***
Étant donné un entier k*, on cherche à déterminer un ensemble* $F^* \subseteq V_f$ *tel que :*

1. $C(F^*) \leq k$
2. $Exc(F^*) = \min\{Exc(F') : F' \subseteq V_f \text{ et } C(F') \leq k\}$

Un tel ensemble F^* *est appelé k-fournisseur optimal.*

Le problème du k-fournisseur est issu du problème nommé k-supplier énoncé par D.S. Hochbaum et D.B. Shmoys dans [5]. Nous ferons donc souvent référence aux auteurs cité précédemment.

Remarque 1.1.4 *Si* $V_c = V_f = V$ *et* $\forall f \in V_f$, $C(f) = 1$*, alors construire un k-fournisseur est équivalent à construire un k-centre tel qu'il est décrit dans [4].*

Remarque 1.1.5 *Si* $V_c \cap V_f = \emptyset$ *et* $V_c \cup V_f = V$ *alors construire un k-fournisseur est équivalent à construire un k-supplier tel qu'il est décrit dans [5].*

D.S. Hochbaum et D.B. Shmoys ont montré dans [4] que construire un k-centre optimal est un problème NP-complet, et dans [5] que construire un k-supplier optimal est aussi un problème NP-complet [1].

Corollaire 1.1.6 *Construire un k-fournisseur optimal est un problème NP-complet.*

En raison du caractère NP-complet du problème, on recherche un algorithme d'approximation qui trouve en temps polynômial un k-fournisseur approché. D.S. Hochbaum et D.B. Shmoys ont montré dans [5] que tout algorithme d'approximation qui résoud le problème du k-supplier a un rapport d'approximation d'au moins 3, c'est-à-dire que la solution trouvée par l'algorithme donne une excentricité inférieure ou égale à 3 fois l'excentricité d'un k-supplier optimal.

Corollaire 1.1.7 *Un algorithme d'approximation pour le problème du k-fournisseur optimal a un rapport d'approximation d'au moins* 3*.*

En effet, le problème du k-supplier est un cas particulier du k-fournisseur.

Remarque 1.1.8 *Le problème du k-centre est équivalent au problème du k-fournisseur où* $V_f = V_c = V$*. On peut, dans le cas où le k-fournisseur se ramène à un k-centre, obtenir un rapport d'approximation de* 2*. (voir [4]).*

1.2 Description de l'algorithme

Dans cette section, nous décrivons un algorithme d'approximation pour le problème du k-fournisseur. Cet algorithme peut être vu comme une généralisation de celui proposé dans l'article

1. l'annexe A montre que la recherche exhaustive est non polynomiale

[5] de D.S. Hochbaum et D.B. Shmoys. On note $E = \{e_1, \ldots, e_m\}$ l'ensemble des arêtes de G triées par ordre de poids *croissant*:

$$\forall i \in \{1, \ldots, m\}, W(e_i) \geq W(e_{i-1})$$

On note $G_i = (V, E_i)$ avec $E_i = \{e \in E : W(e) \leq W(e_i)\}$. En l'occurence, $G_0 = (V, \emptyset)$ et $G_m = (V, E)$.
Comme dans l'approche proposée par D.S Hochbaum et D.B Shmoys dans [5], nous allons construire une procédure itérative qui va appliquer sur les graphes $G_i : i \in \{0, \ldots, m\}$ en faisant croître i une fonction de test d'existence d'un k-fournisseur. Tant que la fonction de test renvoie un certificat d'impossibilité, la procédure poursuit son itération sur i, ce qui fait augmenter le nombre d'arêtes de G_i. Soit b l'itération pour laquelle la fonction de test a pu trouver un k-fournisseur faisable. On appelle G_b le graphe *seuil* ou *bottleneck*. Pour $i < b$, la fonction de test garantit qu'il n'existe pas de solution sur G_i. Pour tout $i > b$, il existe des solutions mais on ne peut déterminer leur rapport d'approximation. Pour $i = b$, la fonction test assure un rapport d'approximation d'au plus 3 avec la solution retournée. La fonction de test est donc le cœur de l'algorithme.
On devra prouver que la fonction TEST décrite plus bas vérifie effectivement ce qui vient d'être cité.

Procédure BOTTLENECK(G, V_c, V_f, k)

1. Si $\forall f \in V_f, C(f) > k$ alors {
2. il n'existe pas de solution
3. Stop.
4. }
5. Sinon {
6. $i = 0$
7. Répeter {
8. $out =$TEST(G_i, V_c, V_f, k)
9. $i = i + 1$
10. }
11. Jusqu'à ce que $out \neq$ "*impossible*"
12. Soit F l'ensemble retourné dans out avec un entier r, retourné par TEST
13. F est un k-fournisseur dans $G : C(F) \leq k$
14. r est le rapport d'approximation : $Exc(F) \leq r \times Exc(F^*)$ où F^* est la solution optimale.
15. }
16. Fin

La première ligne de la procédure BOTTLENECK traduit une condition suffisante pour garantir qu'il n'existe pas de solution dans $G = (V, E)$. En effet, dans le cas contraire, si $\exists u \in V_f : C(u) \leq k$ alors il existe $F \subseteq V_f$ qui vérifie l'énoncé 1.1.3, et qui est donc un k-fournisseur optimal.

Avant d'exposer la fonction de test d'existence d'un k-fournisseur dans G_i, nous allons donner quelques définitions utiles.

Définition 1.2.1 ***Puissance* n *d'un graphe.***
Soit $G = (V, E)$ un graphe quelconque. $G^n = (V, E^n)$ est le graphe à la puissance n de G, où $E^n = E^{n-1} \cup \{(u, v) : (u, w) \in E^{n-1}$ et $(w, v) \in E^{n-1}; [u, v, w \in V]\}$

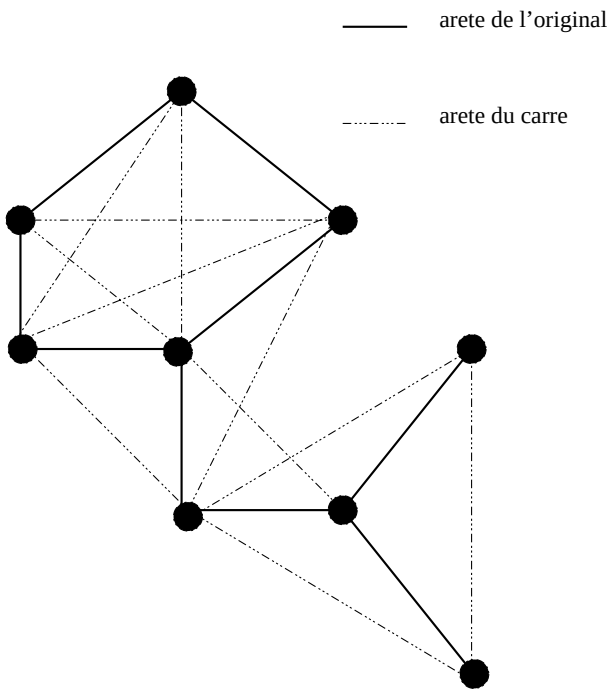


FIG. 1.1 – *Graphe carré*

Définition 1.2.2 ***Restriction d'un graphe à un ensemble de sommets.***
Soit un graphe $G = (V, E)$. Sa restriction à un ensemble de sommets $R \subseteq V$ est le graphe $G \backslash R = (V', E')$ tel que : $V' = R$ et $E' = \{(u, v) : u \in R, v \in R\}$.

Définition 1.2.3 ***Stable maximal.***
Soit un graphe $G = (V, E)$. $S \subseteq V$ est un ensemble stable maximal au sens de l'inclusion si :

- *$\forall u, v \in S, (u, v) \notin E$ (i.e: S est stable)*
- *$\forall v \notin S, S \cup \{v\}$ n'est pas stable.*

Remarque 1.2.4 *Construire un stable maximal dans un graphe peut être fait par un algorithme polynômial.*

Nous allons maintenant décrire la procédure centrale TEST qui est utilisée dans la procédure BOTTLENECK

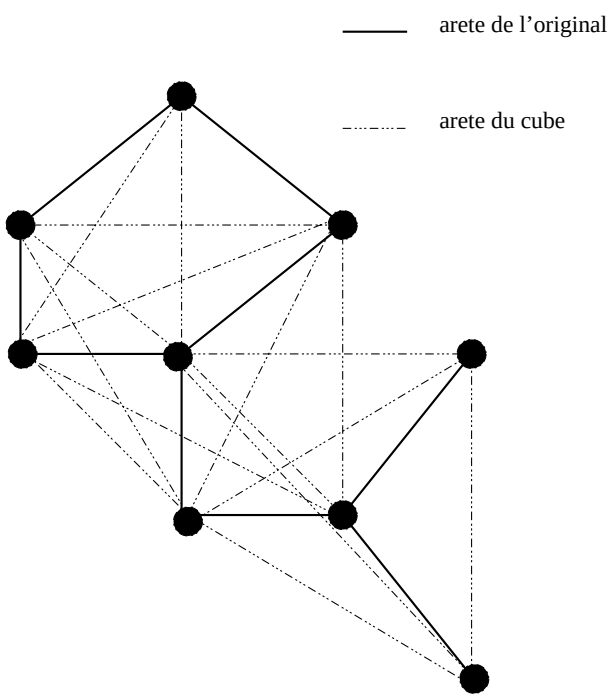


FIG. 1.2 – *Graphe cube*

Procédure TEST(G_i, V_c, V_f, k)

1. Si $\exists v \in V_c : N_{G_i}(v) \cap V_f = \emptyset$ alors Retourner "*impossible*".
2. Sinon {
3. $r = 2$
4. Construire un ensemble stable maximal S dans $G_i^2 \backslash V_c$
5. Pour tout $u \in S$ faire :
6. $C_{min} = \min\{C(v) : v \in V_f \cap N_{G_i}(u)\}$
7. $R = \{v : v \in V_f \cap N_{G_i}(u)$ et $C(v) = C_{min}\}$
8. Si $u \in R$ alors $f(u) = u$ /* $f(u)$ est fournisseur de u */
9. Sinon{
10. Choisir $f(u)$ quelconque dans R
11. $r = 3$
12. }
13. }
14. $F = \bigcup_{u \in S} f(u)$ /* F est l'ensemble des fournisseurs de V_c */
15. Si $C(F) > k$ alors Retourner "*impossible*"
16. Sinon Retourner (F, r)
17. }
18. Fin

La première ligne de la fonction TEST est justifiée par le propos suivant : si dans le graphe $G_i = (V, E_i)$ un sommet de V_c n'est voisin d'aucun fournisseur potentiel de V_f alors il n'existe pas de k-fournisseur dans G d'excentricité $W(E_i)$. TEST retourne alors "*impossible*" pour l'itération i.

Lemme 1.2.5 *Si la fonction* TEST(G_i, V_c, V_f, k) *retourne "impossible" alors il n'existe pas dans* G_i *d'ensemble* $F \subseteq V_f$ *tel que :*

1. $C(F) \leq k$ *et*
2. $\forall u \in V_c, \exists f \in V_f : (u, f) \in E_i$.

Preuve. Si à l'itération i la fonction TEST retourne "*impossible*" dès la première instruction (ligne 1) alors (2) n'est pas vérifié. En effet, $\exists v \in V_c : N_{G_i}(v) \cap V_f = \emptyset$.
Si à l'itération i la fonction TEST retourne "*impossible*" au niveau de la ligne 15, alors $C(F) > k$. Notons $F^* = \{f_1, \ldots, f_p\}$ un ensemble de fournisseurs de coût minimal vérifiant (2). Montrons que F^* ne vérifie pas (1). Chaque sommet de F^* est fournisseur d'un ensemble de sommets de V_c car F^* vérifie (2). Partitionnons V_c en $V_j : 1 \leq j \leq p$ tel que $V_j \subseteq N_G(f_j)$. Dans G_i^2, le sous-graphe induit dans chaque V_j est une clique ; le stable maximal S dans $G_i^2 \backslash V_c$ trouvé par TEST contient au plus un sommet de V_j. Soit r_j ce sommet, s'il existe. Comme f_j est adjacent à r_j, le sommet fournisseur $F(r_j)$ a au plus le coût $C(f_j)$; donc le coût total de $F = \bigcup f_j$ est au plus le coût total de F^*. Ainsi, $C(F) \leq C(F^*)$. Donc, $C(F) > k \Rightarrow C(F^*) > k$.
Si TEST trouve $C(F) > k$ il renvoie le message "*impossible*" car il n'existe pas d'ensemble F^* vérifiant (1). □

Lemme 1.2.6 *Soit* b *la première itération où* TEST *ne renvoie pas le message "impossible" mais un ensemble de fournisseurs, alors* $\forall e_b \in E_b, W(e_b) \leq Exc(F^*)$ *où* F^* *est un* k*-fournisseur optimal.*

Preuve. Si $b = 0$, alors l'ensemble F de fournisseurs est tel que $Exc(F) = 0$. Or $E_b = \emptyset$ donc $Exc(F^*) \geq W(e_b)$.
Si $b > 0$, supposons par l'absurde que $W(e_b) > Exc(F^*)$. Or $W(e_{b-1}) < Exc(F^*)$ car à l'itération $b-1$, F^* n'existe pas[2]. Donc $W(e_{b-1}) < Exc(F^*) < W(e_b)$ or ceci est absurde car $Exc(F^*)$ doit avoir pour valeur le poids d'une arête de G □

Théorème 1.2.7 *La procédure* BOTTLENECK *offre une solution* F *dont l'excentricité est :*

$$Exc(F) \leq 3 \times Exc(F^*)$$

Preuve. Soit F l'ensemble trouvé par l'algorithme et F^* la solution optimale. Montrons que :

1. $C(F) \leq k$ et
2. $Exc(F) \leq 3 \times Exc(F^*)$

vérification de (1) Soit b la première itération où TEST ne renvoie pas le message "*impossible*" mais un ensemble F. D'après la fonction TEST, $C(F) \leq k$ donc (1) est vérifié.

2. voir Lemme 1.2.5

vérification de (2) D'après le lemme 1.2.6, $W(e_b) \leq Exc(F^*)$. Montrons que $Exc(F) \leq 3 \times W(e_b)$: F contient des sommets qui sont adjacents à au moins un sommet du stable S de G_b^2 trouvé par TEST car $F \subseteq \bigcup_{u \in S} N_{G_i}(u)$; S contient des sommets qui dans G_b^2 sont adjacents à au moins un sommet de V_c car S est stable maximal dans $G_b^2 \backslash V_c$. F est donc une solution réalisable dans G_b^3. Comme G_b ne contient que des arêtes e_i de poids $W(e_i) \leq W(e_b)$, on en déduit que $Exc(F) \leq 3 \times W(e_b)$ donc (2) est vérifié par application du lemme 1.2.6.

□

Dans certains cas, on peut obtenir une 2-approximation du k-fournisseur comme on le voit dans la procédure TEST qui peut renvoyer $r = 2$. Ceci est dû au fait que chaque sommet du stable maximal S est inclus dans V_f. En fait, on peut élargir l'hypothèse à ce que V_{cl} soit inclus dans V_f.

Lemme 1.2.8 *Si l'ensemble F retourné par* TEST *est égal à l'ensemble stable maximal S sur $G_b^2 \backslash V_c$ et $C(F) \leq k$ alors la solution F est d'excentricité $Exc(F) \leq 2 \times Exc(F^*)$ où F^* est la solution optimale.*

Preuve. Chaque sommet de S est adjacent à au moins un sommet de V_c dans $G_b^2 \backslash V_c$; comme G_b contient des arêtes e_i de poids $W(e_i) \leq W(e_b)$, on en déduit que $Exc(F) = Exc(S)$ et donc $Exc(F) \leq 2 \times W(e_b)$. Or, d'après le lemme 1.2.6, $W(e_b) \leq Exc(F^*)$ où F^* est la solution optimale. Donc $Exc(F) \leq 2 \times Exc(F^*)$. □

Théorème 1.2.9 *Si la procédure* TEST *renvoie un ensemble F et un rapport $r = 2$ alors l'ensemble F trouvé par* TEST *vérifie les points suivants :*

- $C(F) \leq k$ *et*
- $Exc(F) \leq 2 \times Exc(F^*)$

Preuve.

- Le lemme 1.2.5 montre que si TEST renvoie "*impossible*" alors il n'existe pas de solution à l'itération i
- Soit b la première itération où TEST renvoie F. D'après TEST, $C(F) \leq k$. (1) est vérifié.
- Montrons que $Exc(F) \leq 2 \times Exc(F^*)$: lorsque $r = 2$, le test de la ligne 8 a toujours été vérifié. $F = \bigcup_{u \in S} F(u)$ où $F(u) = u$. Donc $F = S$. Par le lemme 1.2.8 il en découle que $Exc(F) \leq 2 \times D(F^*)$.

□

Il découle directement des théorèmes 1.2.7 et 1.2.9 que la procédure BOTTLENECK offre une r-approximation du problème du k-fournisseur, où $r \in \{2, 3\}$.

Chapitre 2

Problème du changement de la configuration

Dans le cadre des systèmes distribués en particulier, les algorithmes de base sont l'allocation et la distribution de ressources mais aussi l'*adaptation* du système lors des changements topologiques :

On suppose que dans le réseau informatique des clients sont connectés à des fournisseurs qui ont été choisis grâce à l'algorithme proposé au chapitre précédent. Si on observe un changement tel que la connexion d'un nouveau client ou la rupture d'une liaison, on doit pouvoir reconfigurer le k-fournisseur de manière à ce qu'il vérifie bien les propriétés qui lui sont attribuées tout en conservant une 3-approximation et si possible en modifiant le moins possible la configuration de fournisseurs précédente.

2.1 Exposé du modèle

Pour montrer les différentes étapes de changement, nous allons indicer les objets que nous manipulons par un numéro qui indique à quel instant il se trouve. Par exemple, $V_{cl}1$ est l'ensemble des sommets *clients* V_c à l'instant 1. L'ordre des nombres indique la séquence dans le temps. Lors d'un changement, les propriétés du graphe sont modifées. Par exemple, si un nouveau client s'ajoute, $|V_c|$ augmente. Avec cette nouvelle configuration, on ne sait pas si le k-fournisseur de départ est encore une 3-approximation. La figure 2.1 montre un cas où le k-fournisseur n'est plus une 3-approximation après la connexion d'un nouveau client.

Définition 2.1.1 ***Configuration***
Nous définissons la configuration i comme étant le triplet (V_ci, V_fi, Fi) où V_ci est l'ensemble des clients à l'instant i, V_fi est l'ensemble de fournisseurs potentiels à l'instant i, et $Fi \subseteq V_fi$ est un k-fournisseur de V_ci.

Définition 2.1.2 ***Changement.***
Un changement est le passage d'une configuration i à une configuration $i+1$ et correspond à la modification d'un ou plusieurs ensembles du triplet (V_ci, V_fi, Fi).

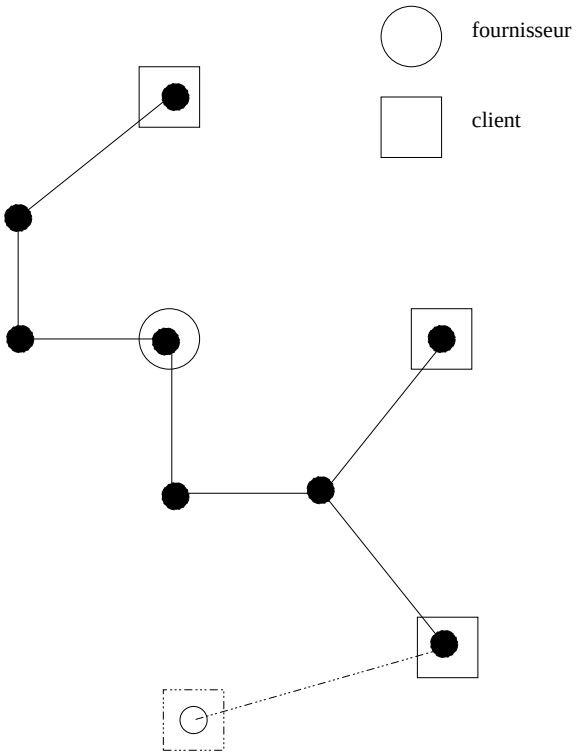


FIG. 2.1 – *Connexion d'un nouveau client*

Si l'on calcule un nouveau k-fournisseur avec la procédure présentée au chapitre précédent, on peut obtenir un ensemble très différent de celui qui était déjà en place. On devra donc faire *migrer* les fournisseurs d'une configuration à l'autre. On considère que la migration a un coût et on aimerait minimiser ce coût.

Définition 2.1.3 ***Coût de migration.***
Soit $F0$ l'ensemble des fournisseurs de la configuration de départ, et $F1$ l'ensemble des fournisseurs après un changement. Le coût de migration des fournisseurs M_{01} pour passer de la configuration 0 à la configuration 1 est égale au nombre de suppressions et de créations de fournisseurs qui ont été nécessaires pour passer de $F0$ à $F1$.

Problème 2.1.4 ***k-fournisseur dynamique***
Soit une configuration $C0 = (V_c0, V_f0, F0)$ où $F0 \subseteq V_f$, forme un k-fournisseur de V_c0. Soit $C1 = (V_c1, V_f1, F1)$ la nouvelle configuration après un changement. Le but est de trouver un k-fournisseur $F1$ de V_c1 tel que :

1. *La solution trouvée $F1$ est une 3-approximation de la solution optimale dans la nouvelle configuration.*
2. *La solution trouvée $F1$ implique le moins de migrations possibles de fournisseur pour passer de $F0$ à $F1$.*

Proposition 2.1.5 Heuristique de résolution du problème du changement
Considérons un changement d'une configuration 0 à une configuration 1. Vérifier que $F1$ est une 3-approximation du k-fournisseur optimal consiste à résoudre deux sous-problèmes :

1. *Calculer l'indice bottleneck du graphe G (voir section 1.2) pour lequel on sait qu'il existe une solution réalisable pour la configuration 1. Soit $b1$ cet indice. G_{b1} est le graphe* seuil *de la configuration 1.*
2. *Lorsque l'on connaît l'indice* bottleneck $b1$ *dans lequel il existe une solution faisable pour la configuration 1, on doit trouver un algorithme de* migration *qui minimise le coût M_{01} en trouvant un ensemble $F1$ tel que chaque sommet de V_c1 a au moins 1 voisin qui appartienne à $F1$ dans le graphe G^3_{b1}.*

La preuve du théorème 1.2.7 associée au lemme 1.2.6 montrent que ces deux conditions sont suffisantes pour garantir une 3-approximation.

2.2 Heuristique de migration

Avant de présenter une heuristique de migration, nous allons nous placer dans un cadre restreint : Nous considérons que les sommets de V_f ont tous un coût égal à 1 : $\forall f \in V_f, C(f) = 1$. Dans ce cas, un k-fournisseur compte au plus k sommets fournisseurs.

L'heuristique présentée prend en argument un graphe bottleneck. Ce graphe sera donné par la procédure BOTTLENECK qui a été présentée au chapitre précédent. On se souviendra que le graphe *bottleneck* G_b accepte un k-fournisseur d'excentricité au plus $3 \times b$ et que $\forall i < b$ il n'existe pas de solution dans G_i. Nous allons nous appuyer sur la procédure TEST décrite au chapitre précédent. Celle-ci recherche un stable maximal au sens de l'inclusion parmis les sommets clients du graphe (Cf: ligne 4 de la procédure TEST). L'idée ici est de trouver un stable qui ait la propriété d'être dans le voisinage des anciens fournisseurs. Pour les besoins de l'heuristique, nous introduisons les définitions suivantes.

Définition 2.2.1 ***Stable de cardinalité maximum***
Dans un graphe $G = (V, E)$, soit l'ensemble de sommets $A \subseteq V$ tel que :

- $\forall u, v \in S, (u, v) \notin E$ *(i.e: S est stable)*
- $|A|$ *est maximum*

Alors A est un stable de cardinalité maximum

Définition 2.2.2 ***Ensemble des candidats***
L'ensemble des clients de V_c1 qui ont pour voisin au moins 1 sommet de $F0$ dans G_b est l'ensemble C des candidats au stable.

Définition 2.2.3 ***Graphe de candidature***
Soit $G'_b = (V', E'_b)$ le graphe dérivé du graphe originel $G = (V, E)$ et C l'ensemble des candidats tel que :

- $V' = F0 \cup C$
- *et* $E'_b = \{(x, y) \in E_b^2 : x, y \in C\} \cup \{(x, z) \in E_b : x \in C \text{ et } z \in F0\}$

G'_b est appelé le graphe de candidature.

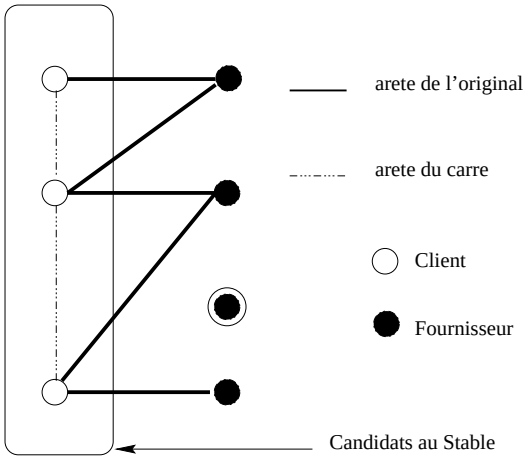


FIG. 2.2 – *graphe de candidature*

A partir du stable *maximum* A sur $G'\backslash C$ nous pourront construire le stable *maximal* $S1$ sur $G^2_b\backslash V_c1$ recherché par la procédure MIGRER [1] tout en conservant le maximum de fournisseurs de la configuration passée $F0$ pour l'inclure au nouvel ensemble de fournisseurs $F1$. Tout d'abord, nous allons montrer que trouver un stable *maximum* A sur $G'\backslash C$ permet de sauvegarder un maximum de fournisseurs de $F0$.

Lemme 2.2.4 *Soit une configuration $(V_c0, F0, V_f0)$ et un changement de V_c0 à V_c1. Soit C l'ensemble des candidats (définit dans 2.2.2) et A un stable dans le graphe de candidature $G'\backslash C$ (définit dans 2.2.3). Le nombre de fournisseurs de $F0$ sauvegardés dans $F1$ est au moins égal au cardinal de A.*

Preuve. Nous allons nous appuyer sur la nature du graphe de candidature G' pour prouver le lemme 2.2.4. Soit $x \in C$ et $z \in F0$. Le but est de montrer qu'à *un* sommet de A correspond *un et un seul* sommet de $F0$ sauvegardé dans $F1$. Nous avons trois cas à traiter :

- Par construction de G', tout sommet de A a pour voisin au moins *un* sommet de $F0$ car $A \subseteq C$.
- Si deux sommets de C ont pour voisin un même sommet de $F0$ alors, dû à la construction de G' , ils sont voisins entre eux dans G'. Si l'un de ces deux sommets de C appartient à A alors l'autre ne peut être dans A car A est un stable dans G'.
- Si un sommet de C a pour voisin dans $F0$ plus d'un seul sommet, alors un choix arbitraire entre ces sommets est fait pour n'en incorporer qu'un seul dans $F1$ (Cf : ligne 12 de la procédure MIGRER).

On constate bien qu'à 1 sommet x de A correspond au moins 1 sommet z de $F0$ sauvegardé. □

En définitive, pour tout stable A dans $G'\backslash C$, on a au moins $|A|$ anciens fournisseurs sauvegardés. Puisque $A \subseteq S1$ (voir la procédure MIGRER) et que A est le plus grand possible, on espère sauvegar-

1. Cf : ligne 7 de la procédure

der un maximum de sommets de $F0$ dans $F1$. On construit donc $S1$ stable maximal dans $G_b^2 \backslash V_c1$ à partir de A. La procédure ci-après est utilisée pour faire migrer les fournisseurs d'une configuration à l'autre. Elle correspond à la deuxième partie de la proposition 2.1.5. On considère donc que l'indice bottleneck b du graphe est déja donné.

Procédure MIGRER$(G_b, V_c1, V_f1, F0, k)$

1. $C = \{v \in V_c1 : N_{G_b}(v) \cap F0 \neq \emptyset\}$
2. /* Construction de $G' = (V', E')$ le graphe de candidature : */
3. $V' = C \cup F0$
4. $E' = \{(x, y) \in E^2 : x \in C, y \in C\} \cup \{(x, z) \in E : x \in C, z \in F0\}$
5. *Construire* un ensemble stable A le plus gros possible dans $G' \backslash C$
6. $V_c1' = V_c1 - \bigcup_{u \in A} N_{G_b^2}(u)$
7. *Construire* un stable maximal $S1'$ dans $G_b^2 \backslash V_c1'$
8. $S1 = A \cup S1'$ /* $S1$ est fait sur la base de A */
9. Pour tout $u \in S1$ faire {
10. $\quad R = V_f \cap N_{G_b}(u)$
11. $\quad$ Si $R \cap F0 \neq \emptyset$ alors {
12. $\quad\quad$ Choisir $f_1(u)$ quelconque dans $R \cap F0$ /* $f_1(u)$ est fournisseur de u */
13. $\quad$ }
14. $\quad$ Sinon Choisir $f_1(u)$ quelconque dans R
15. }
16. $F1 = \bigcup_{u \in S} f_1(u)$ /* $F1$ est l'ensemble des fournisseurs de V_c1 */
17. Si $|F1| > k$ {
18. $\quad$ recommencer la procédure avec b=b+1; /* indice bottleneck à affiner */
19. }
20. Retourner $(F1, 3)$
21. Fin

Comme on peut le voir à la ligne 18, la procédure de migration peut être relancée à un indice de graphe $b+1$ si le nombre de fournisseurs est supérieur à k. Or, à l'origine, ayant éxecuté la procédure BOTTLENECK, on est censé avoir trouvé cet indice. La problématique vient du fait que lorsque l'on trouve un indice *bottleneck* il existe alors une solution faisable *au pire* dans le cube du graphe et non forcément dans le graphe G_b lui-même. Entre G_b et G_b^3 il existe un espace sans solution et un espace de solutions réalisables. Au mieux, l'espace des solutions débute dès G_b, au pire, il est réduit à G_b^3 seulement. S'il s'avère que l'on trouve deux indices *bottleneck* comme à la figure 2.3 cela signifie que l'on a affiné la connaissance de l'espace des solutions réalisables.

Théorème 2.2.5 ***np-complétude du stable maximum***
Construire un stable de cardinalité maximum[2] *est un problème np-complet.*

2. Cf : définition 2.2.1

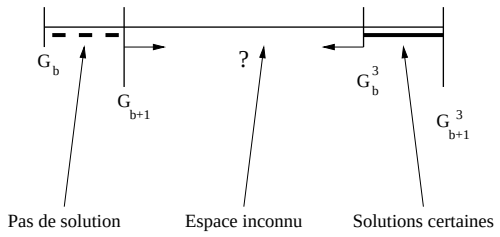


FIG. 2.3 – *Indices bottleneck et espace de solutions*

Afin de conserver le caractère polynômial de l'heuristique de recherche du k-fournisseur dynamique, nous devons utiliser un algorithme d'approximation du stable maximum (Cf: théorème 2.2.5).

Caractéristiques de l'heuristique

L'heuristique proposée considère que pour faire migrer le moins possible de fournisseur, il faut *orienter* la construction du stable maximal $S1'$ (ligne 7) de manière à donner au voisinage des sommets de $S1'$ le plus possible d'anciens fournisseurs. Cette orientation se situe exactement à la créatione du stable maximum A parmis les nouveaux clients qui ont pour voisin un ancien fournisseur (ligne 5). On voit que cette heuristique résoud donc un changement de clients mais il en est de même pour un changement de fournisseur (ex : panne) où cette fois-ci on prendra pour $F0$ le *nouvel* ensemble de fournisseurs. De plus, la coupure d'un lien de communication peut être représenté dans le graphe G sachant que la procédure BOTTLENECK doit auparavant trouver G_b.

La performance de l'heuristique présentée dans la procédure MIGRER dépend entre autre du rapport d'approximation de l'algorithme de recherche du stable[3] A car $|A|$ détermine le nombre de fournisseurs sauvegardés d'une configuration à l'autre. On trouve dans la littérature informatique[4] des algorithmes d'approximation pour le problème du stable maximum mais ceux-ci ont une piètre performance. En revanche, l'heuristique de migration proposée a pour qualité d'être robuste car elle peut gérer toute sorte de changement, à savoir :

- apparition/disparition de plusieurs clients: On utilise pour nouvel ensemble de clients l'ensemble V_c1
- apparition/disparition de plusieurs fournisseurs qu'ils soit de V_f ou de $F0$: On place dans $F0$ l'ensemble de fournisseurs modifié. Pour V_f modifié, on le prend en compte dès l'execution de la procédure BOTTLENECK
- apparition/disparition de liens entre sommets (i.e: panne de liaisons): La procédure BOTTLENECK étant executée à partir de G_0, elle prend en compte la disparition de liens.

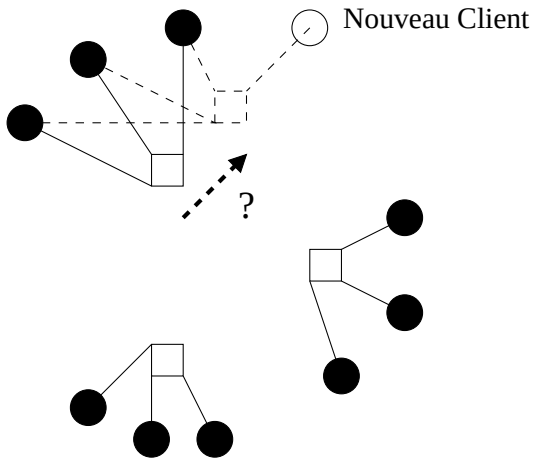


FIG. 2.4 – *L'ajout d'un client et la migration*

2.3 Cas d'étude : ajout d'un client

A défaut d'évaluer le rapport de performance de l'algorithme de migration, nous allons étudier dans le cas de l'ajout d'un simple client u dans V_c, comment obtenir un k-fournisseur qui ait entrainé le moins possible de migrations. Pour cela, en reprenant les notations introduites au début de ce chapitre, nous allons passer en revue cas par cas les conditions dans lesquelles le client u est ajouté sans pour autant réussir à donner une étude exhaustive.

Soit $F0$ le k-fournisseur de V_c0 et F^*0 un k-fournisseur optimal de V_c0. considérons que $F0$ est une 3-approximation de F^*0 pour V_c0.
Soit $c_0 = Exc(F0)$ et $c_0^* = Exc(F^*0)$ pour V_c0.
L'ajout d'un client se traduit par : $V_c1 = V_c0 \cup \{u\}$.
Soit $c_1^* = Exc(F^*1)$ pour V_c1 où F^*1 est un k-fournisseur optimal de V_c1.
Nous affirmons dans le lemme suivant que l'ajout d'un client ne peut donner lieu à obtenir un k-fournisseur d'excentricité meilleure que sans cet ajout.

Lemme 2.3.1 *Pour toute configuration* 1 *où* $V_c1 = V_c0 \cup \{u\}$, $c_0^* \leq c_1^*$.

Preuve. Supposons par contradiction que $c_1^* < c_0^*$. Donc il existe un k-fournisseur F^*1 de V_c1 d'excentricité c_1^*. C'est-à-dire : $\forall u \in V_c1, \exists f \in F^*1 : W(u,f) \leq c_1^*$. Comme $V_c0 \subseteq V_c1, \forall u \in V_c0, \exists f \in F^*1 : W(u,f) \leq c_1^* < c_0^*$. Or c_0^* est par définition l'excentricité optimale d'un k-fournisseur de V_c0 ; il y a donc contradiction avec l'hypothèse de départ. □

Rappel : par notation et définition, $c_0 \leq 3 \times c_0^*$.

Définition 2.3.2 ***Redondance.***
Un fournisseur f_0 *est redondant dans l'ensemble des fournisseurs* $F0$ *par rapport à* V_c0 *si* $Exc(F0 - \{f_0\}) = Exc(F0)$.
Soit donc u le client ajouté à V_c0 pour donner V_c1.

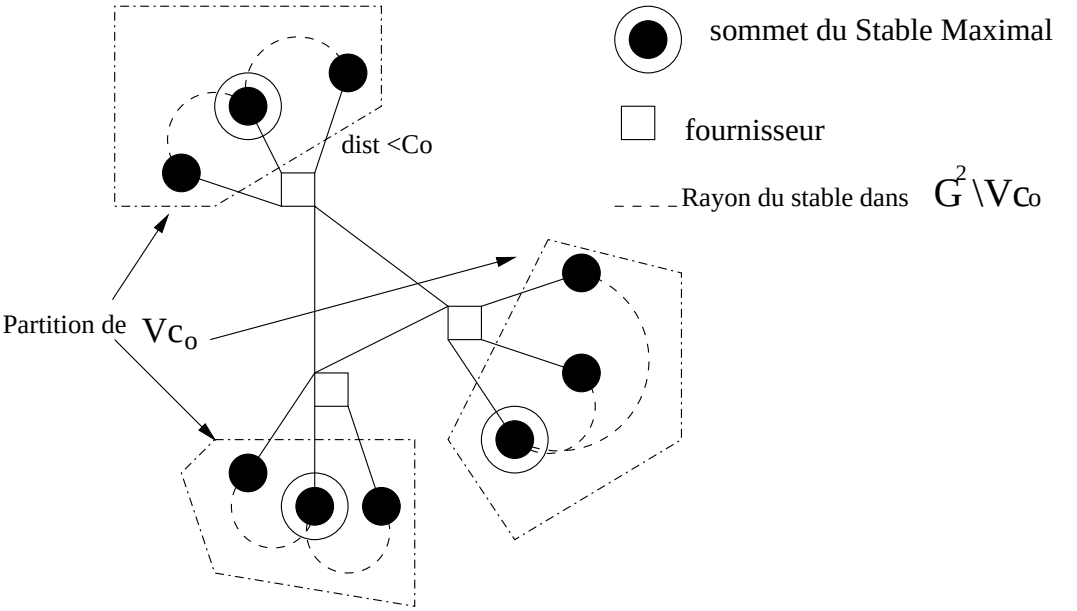
3. voir théorème 2.2.5
4. par exemple dans [7]

Si $W(u,F0) \leq c_0$ alors pour V_c1, $Exc(F0) \leq c_0 \leq 3 \times c_0^* \leq 3 \times c_1^*$. On prend donc $F1 = F0$ et l'on n'opère donc aucune migration de fournisseur. $Exc(F1) \leq 3 \times c_1^*$ et $|F1| \leq k$ donc $F1$ est une 3-approximation du k-fournisseur optimal de V_c1.

Si $W(u,F0) > c_0$ alors :

- si $|F0| < k$ et $\exists f \in V_f : W(u,f) \leq c_0$:
  On prend alors $F1 = F0 \cup \{f\}$. $|F1| \leq k$ et pour V_c1, $Exc(F1) \leq c_0 \leq 3 \times c_0^* \leq 3 \times c_1^*$. Donc $F1$ est une 3-approximation du k-fournisseur optimal de V_c1.
- si $|F0| < k$ et $\forall f \in V_f, W(u,f) > c_0$: Soit $f \in V_f : W(u,f) = d_1 = W(u,V_f)$, f est le fournisseur le plus proche de u. Dans ce cas, $c_1^* \geq d_1$. On prend alors $F1 = F0 \cup \{f\}$. $|F1| \leq k$ et pour V_c1, $Exc(F1) \leq d1 \leq c_1^*$. $Exc(F1) \leq 3 \times c_1^*$ donc $F1$ est une 3-approximation du k-fournisseur optimal de V_c1. A ce stade, nous avons étudié les cas où $|F0| < k$.
- Maintenant, si $|F0| = k$ et f_0 est redondant dans $F0$ par rapport à V_c0 :
  Soit f_1 le fournisseur le plus proche de u : $W(u,f_1) = d_1 = W(u,V_f)$; on prend alors $F1 = F0 \cup \{f_1\} - \{f_0\}$. $|F1| = k$ et :
  - si $d_1 \leq c_0$ alors pour V_c1, $Exc(F1) \leq c_0 \leq 3 \times c_0^* \leq 3 \times c_1^*$.
  - si $d_1 > c_0$, sachant que $c_1^* \geq d_1$ et pour V_c1, $Exc(F1) = d_1$ on a alors $Exc(F1) \leq c_1^*$.
- si $|F0| = k$ et $F0$ n'a pas de redondance .
  Dans cette situation, on va utiliser l'algorithme d'approximation : soit f_1 le fournisseur le plus proche de u : $W(u,f_1) = d_1 = W(u,V_f)$. Dans G_{c_0}, $N_{G_{c_0}}(u) \cap V_f = \emptyset$ par propriété. TEST renvoie donc *"impossible"* dans G_{c_0} et de même pour tous les cas dans $G_{c'}$ avec $c' < d_1$

Remarque 2.3.3 *Partitionnons V_{c_0} en k sous-ensembles tels que chaque ensemble de clients soit associé à son fournisseur de $F0$ le plus proche. Par hypothèse de l'inégalité triangulaire, nous avons au plus un seul client qui fasse partie à la fois au stable maximal S de $G^2_{c_0} \backslash V_{c_0}$ et a un sous-ensemble de la partition. En effet, si un client s fait partie du stable maximal, tout client associé au fournisseur de s est voisin de s dans $G^2_{c_0} \backslash V_{c_0}$ et ne peut donc appartenir au stable.*



Dans $G^3_{c_0}$ ces sous-ensembles sont des cliques et l'on a aussi en conséquence la propriété que $|S| \leq k$.

Soit donc $d_2$le nouveau niveau *bottleneck* trouvé après l'ajout du client u grâce à la procédure BOTTLENECK. On supposera que $d_2 \geq c_0$. La procédure a construit un stable maximal S dans $G^2_{d_2} \backslash V_c 1$. On peut assurer d'après la remarque 2.3.3 que $|S| \leq k+1$ car au pire, u est un élément ajouté au stable de l'ancienne configuration. Notons le stable maximal trouvé $S = \{s_1, \ldots, s_{|S|}\}$ avec $|S| \leq k+1$. $\forall s \in S$ et $s \neq u$, on prendra comme fournisseur celui qui est associé à la partition contenant s comme vu à la remarque 2.3.3. Il reste à trouver un fournisseur pour u :

- Si $u \notin S$ alors $W(u,S) \leq 2 \times d_2$: u appartient à une des partitions induites par S. En conséquence, $|\bigcup_{s \in S} f(s)| \leq k$ et $F = \bigcup_{s \in S} f(s)$ est une 3-approximation du k-fournisseur optimal. Comme $F \subseteq F0$ on prend $F1 = F0$ et l'on n'opère aucune migration. Comme $F1 = F0$, $|F1| \leq k$ et comme $F \subseteq F1$ on obtient bien une 3-approximation du k-fournisseur optimal de $V_c 1$.
- Si $u \in S$ alors deux cas se présentent : l'un où u est voisin d'un fournisseur de l'ancienne configuration dans G_{d_2}, et l'autre où il en est trop éloigné : $N_{G_{d_2}}(u) \cap F0 = \emptyset$.
  - Si $\exists f \in \bigcup_{s \in S} f(s) : f \in N_{G_{d_2}}(u)$ alors $f(u) = f$ et l'on peut prendre $F1 = F0$. $|F1| \leq k$ et d_2 est le premier indice *bottleneck* donc on obtient bien une 3-approximation du k-fournisseur optimal de $V_c 1$.
  - Si $\forall f \in \bigcup_{s \in S} f(s), f \notin N_{G_{d_2}}(u)$ alors choisir $f \in N_{G_{d_2}}(u)$. $F = \bigcup_{s \in S} f(s) \cup \{f\}$:
    - Si $|F| \leq k$ alors $\exists f_i \in F0 : f_i \notin F$ et l'on prend alors $F1 = F0 \cup \{f\} - \{f_i\}$: on a fait migrer un fournisseur. $|F1| = |F0|$ et d_2 est le premier indice *bottleneck* donc on obtient bien une 3-approximation du k-fournisseur optimal de $V_c 1$.
    - Si $|F| > k$ alors il n'existe pas de solution dans G_{d_2}. On doit continuer à progresser dans la procédure BOTTLENECK. L'indice suivant peut mener à des cas nouveaux, ces cas ne seront pas étudiés.

Le simple ajout d'un client donne donc lieu a de nombreux cas tels qu'il est difficile de les synthetiser en quelques lignes. En revanche, on en tire la conclusion que la configuration des fournisseurs n'en est finalement que très peu altérée (nous avons au plus fait migrer un seul fournisseur) mais qu'un algorithme de migration simpliste comme celui présenté au chapitre précédent ne saurait être assez fin pour optimiser au mieux le nombre de migrations impliqué par un changement particulier. En réalité, il y a necessité de classer les sortes de changement et d'appliquer différentes méthodes de décision de migration pour obtenir un résultat de qualité. Dans le cas général, tel qu'il est pris en compte par l'algorithme présenté au chapitre 2.2, le problème du k-fournisseur dynamique pose un nombre considérable de combinaisons de changements incluant les clients, les fournisseurs et les liaisons entre sommets. Il en résulte que l'étude de la dynamicité doit s'étendre sur deux dimensions : une première en terme de *variété* des changement et une seconde en terme de *qualité* de migration.

Conclusion

Dans le cadre général des problèmes associés à la communication de groupe, la particularité du problème du k-fournisseur nous a permis d'apprécier du point de vue algorithmique une méthode de résolution basée sur le phénomène *"bottleneck"* introduit dans [5] par D. hochbaum et B. Shmoys. L'étude approfondie de cette méthode de résolution nous a permis de découvrir, hors de son cadre d'application, quelque de ses propriétés singulières, ses avantages et ses pièges.

Par la nature même des algorithmes dits d'*approximation*, la motivation principale qui les accompagne est de donner la *preuve* de leur capacité a offrir une $f(n)$-approximation de la solution optimale qui n'est pas atteignable en temps polynômiale. Dans notre cas, nous avons pu garantir un rapport d'approximation constant et relativement faible.

Le problème du k-fournisseur dynamique est une nouveauté par rapport à celui du k-fournisseur simple déjà introduit par D. hochbaum et B. Shmoys et généralisé dans ce rapport. Peu de conclusions générales en ont été tirées car il s'avère que la dynamicité apporte de nombreux paramètres difficiles à maîtriser. Cependant, une heuristique de migration robuste a été proposée et permet déjà de traiter le problème mais son efficacité est à prouver.

Bibliographie

[1] G. Ausiello, P. Crescenzi, M. Protasi, Approximate solution of NP optimization problems, *Theoretical computer science* 150 (1995) 1-55.

[2] J.A. Bondy, U.S.R. Murty, *Graphs theory with applications* (North holland, 1976).

[3] M. Garey, D. Johnson, *Computers and intractability* (Freeman and company, 1979).

[4] D. hochbaum, B. Shmoys, A best possible heuristic for the k-center problem, *Mathematics of Operations Research* vol.10 n.2 (2 May 1985) 180-184.

[5] D. Hochbaum, B. Shmoys, A unified approach to approximation algorithms for bottleneck problems, *Journal of the Association for Computing Machinery* vol.33 n.3 (Jul 1986) 533-550.

[6] D. Hochbaum editor, *Approximation algorithms for NP-Hard problems* (PWS Publishing company, 1997).

[7] P. Crescenzi, V. Kann, *A compendium of NP optimization problems*, (`http://www.nada.kth.se/theory/problemlist.html`)

Annexe A

Complexité d'un algorithme exhaustif de recherche d'un k-fournisseur optimal

Tout d'abord, nous nous intéressons au calcul de l'excentricité d'un ensemble $S \subseteq V_f$ quel qu'il soit.

Complexité du calcul de l'excentricité d'un ensemble $S \subseteq V_f$

1. Pour chaque sommet c de V_c, on détermine la plus courte distance pour atteindre un sommet s de S :
   $\forall c \in V_c, \forall s \in S$ trouver $\min_{s \in S} W(s, c)$.
   Ce calcul se fait en $O(|S| \times |V_c|)$ en nombre de comparaisons de $W(s, c)$.
   On obtient une liste de $|V_c|$ distances W_i
2. On prend maintenant la valeur *maximale* de ces $|V_c|$ distances W_i. Ce calcul est linéaire, de complexité $O(S)$.

En conséquence, le calcul de l'excentricité d'un ensemble $S \subseteq V_f$ est au pire de complexité $O(|S| \times |V_c| + |V_c|)$ en nombre de comparaisons, ce qui est un calcul polynômial.

Calcul du k-fournisseur optimal

Le problème principal est de trouver l'ensemble $S \subseteq V_f$ tel que :
$Exc(S) = \min\{Exc(S') : S' \subseteq V_f \text{ et } C(S,) \leq k\}$ Nous avons donc un dénombrement de P ensembles S distincts tel que :

$$P = \sum_{i=0}^{k} \frac{|V_f|!}{(|V_f| - i)! \times i!}$$

Parmis ces P ensembles dont on doit calculer l'excentricité, il en existe au pire un seul dont l'excentricité $Exc(S^*)$ est minimale. Nous avons donc P comparaisons à ajouter au calcul. En conséquence, la complexité en nombre de comparaisons pour un algorithme de recherche exhaustive d'un

k-fournisseur optimal est :

$$C(V_c, V_f, k) = ([(|V_c| \times k + k) \times P] + P)$$

ce qui est de conplexité *non* polynômiale car P n'est pas un polynôme.