Optimisation de réseaux SONET/SDH : éléments théoriques et résolution pratique

Pierre LEMAIRE
ENSIMAG / DEA Recherche Opérationnelle, Combinatoire et Optimisation
INPG
-
sous la direction de
Nadia BRAUNER et Gerd FINKE
Équipe de Recherche Opérationnelle
Laboratoire Leibniz-IMAG
Grenoble

janvier - juin 2001

Table des matières

Introduction

Ce travail a été réalisé dans le cadre du DEA de Recherche Opérationnelle, Combinatoire et Optimisation de l'INPG[1] ainsi qu'en tant que projet de fin d'études de l'ENSIMAG[2]. Il a été effectué sous la responsabilité de Nadia Brauner et Gerd Finke, de l'équipe de Recherche Opérationnelle du laboratoire Leibniz-IMAG[3] et porte sur la conception de réseaux de télécommunications, plus particulièrement les réseaux SONET/SDH de type SHR unidirectionnel, et est organisé comme suit.

Nous commençons par présenter informellement puis formellement le problème dans un premier chapitre. Nous donnons ensuite, dans le chapitre 2, quelques résultats théoriques tels que complexité du problème et bornes sur les valeurs optimales. Le chapitre 3 est dédié à la description d'algorithmes ayant des performances garanties et il trouve son prolongement dans le chapitre 4 qui présente, lui, des algorithmes gloutons de résolution. Cette partie de résolution se termine par le chapitre 5 où nous décrivons une méthode de recherche tabou pour notre problème. Un dernier chapitre conclut ce rapport et le lecteur trouvera de nombreux compléments dans les annexes.

[1]INPG : Institut National Polytechnique de Grenoble

[2]ENSIMAG : École Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble

[3]IMAG : Informatique et Mathématiques Appliquées de Grenoble

Chapitre 1

Présentation du problème

Dans ce chapitre, nous présentons brièvement les réseaux SDH qui constituent le cadre général de ce projet ; suivront une description formelle du problème puis des modélisations.

1.1 Les réseaux SDH

Avec l'explosion des télécommunications, une conception optimale des réseaux devient un enjeu crucial. Pour cela, l'un des critères essentiels est l'aptitude du réseau à résister aux pannes, *i.e.* sa capacité à rediriger les signaux lors de la défection d'un élément, si possible de manière automatique. Bien entendu cette résistance ne va pas sans un sur-coût qu'il faut maîtriser.

La technologie SONET/SDH[1] apporte une solution qui satisfait à la fois les critères techniques et économiques : elle consiste en l'élaboration d'*anneaux auto-cicatrisants*, connus dans la littérature sous le nom de *Self-Healing Ring* ou SHR : les nœuds du réseau sont regroupés en cycles et sont connectés selon ces cycles en utilisant au moins deux fibres séparées orientées dans des sens opposés. En chaque nœud se trouve un ADM[2] correctement dimensionné qui permet les communications et qui oriente automatiquement le signal sur une fibre ou l'autre. Comme l'information doit pouvoir circuler sur tout l'anneau, tous les ADMs ont la même aptitude de traitement, qui est donc une caractéristique de l'anneau : sa *capacité*.

Le standard européen, défini par l'ETSI[3], est d'utiliser des anneaux *unidirectionnels*, *i.e.* pour lesquels un sens de parcours est dédié au fonctionnement normal, l'autre étant réservé aux cas de défaillance. Comme l'illustre la figure 1.1, les ADMs sont capables de ré-orienter le signal sur la fibre de

[1] SONET (*Synchronous Optical NETwork*) est la terminologie américaine, SDH (*Synchronous Digital Hierarchy*) européenne.

[2] ADM : *Add/Drop Multiplexer* (Multiplexeur par Insertion Extraction ou MIE)

[3] ETSI : *European Telecommunications Standards Institute* (l'institut européen des standards de télécommunications)

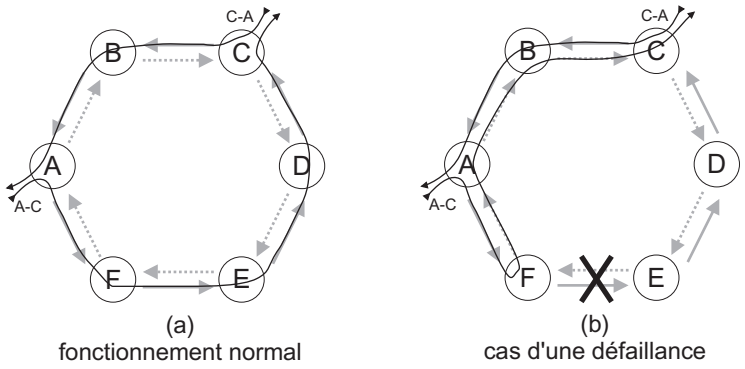

FIG. 1.1 – Un anneau de type SHR unidirectionnel.

sécurité lorsqu'une rupture de connexion survient, ce qui permet de continuer d'assurer le trafic même avec une défaillance de connexion par anneau.

Il existe d'autres standards, par exemple le SHR bidirectionnel pour lequel les signaux peuvent aller dans un sens comme dans l'autre, la moitié de la capacité de chaque fibre servant de sécurité. Ce standard permet en général un meilleur rendement (*i.e.* un plus grand débit pour un même dimensionnement) mais s'ajoute alors un problème de répartition des signaux sur chaque fibre pour tirer véritablement profit de cette propriété.

Pour plus de détails le lecteur est renvoyé au site internet *The SONET Home Page* [1]. Pour une approche plus technologique des réseaux SDH, le lecteur trouvera dans la page de Demetris Mili [18] une première introduction. Enfin l'ETSI publie de très nombreux documents (études, standards, références, etc) sur le fonctionnement, la réalisation pratique et tout ce qui touche les télécommunications ; pour notre sujet, nous citerons [5, 4] ainsi que leur site internet [3].

1.2 Description du problème

Notre problème s'inscrit dans le cadre général suivant : nous disposons de demandes entre des nœuds (une matrice $D = (d_{i,j})$) que nous devons répartir en anneaux. La création d'un anneau a un certain coût, de même que le raccord d'un nœud à un anneau. Le but est bien entendu de minimiser le coût global tout en satisfaisant les demandes.

Dans notre cas, nous considérons de plus les restrictions suivantes :

– la matrice D des demandes est symétrique et n'a que des 0 sur sa diagonale, *i.e.* que le graphe des demandes est non-orienté et simple[4] ;

[4]La forme du graphe des demandes revient à supposer que la demande de i vers j est la même que celle de j vers i, que toute la demande entre 2 sommets est groupée en une seule demande (pas d'arêtes parallèles) et qu'il n'y a pas de demande d'un sommet vers

- les anneaux sont tous identiques, de type SHR unidirectionnel, avec un coût de création fixé r et une capacité fixée C ;
- un nœud peut être raccordé à plusieurs anneaux (chaque raccord à un coût fixé l), mais une demande $d_{i,j}$, elle, ne peut pas être séparée sur plusieurs anneaux ;
- il n'y a pas de trafic hors des anneaux, *i.e.* que les anneaux doivent couvrir l'ensemble des demandes.

Nous garderons pour ce problème sa terminologie anglaise de ADR[5]. Des cas restreints apparaîtront aussi au fil de ce rapport, notamment le cas de demandes toutes unitaires que nous noterons AUDR[6] et son sous-cas pour lequel $r = 0$, noté ADRL[7] (on trouve parfois k-EP[8]).

Remarques sur les coûts : Le rapport entre le coût de création d'un anneau et celui du raccord d'un nœud dépend du type de réseau. Pour des réseaux locaux pour lesquels l'extension spatiale d'un anneau est faible, le coût d'un ADM prévaut sur celui de la fibre et on a $l > r$. Par contre, lorsqu'on passe à des réseaux de grande échelle, la longueur d'un anneau est telle qu'alors $r > l$. De plus, il faut noter que pour certains réseaux la différence entre les longueurs réelles des anneaux peut rendre abusif de supposer que r est constant.

1.3 Modélisations

Nous présentons ici deux modélisations différentes de notre problème, dues à Brauner, Crama et Wynants [2], correspondant à deux approches différentes du problème.

1.3.1 Programme linéaire en 0-1

Voici une modélisation par un programme linéaire du problème ADR. Nous avons n nœuds pour lesquels les demandes sont représentées par la matrice $D = (d_{ij})$. Comme D est symétrique à diagonale nulle, nous nous limitons aux couples (i, j) tels que $i < j$, et plus particulièrement à : $\Omega = \big\{(i, j) \in \{1, \ldots, n\}^2, i < j, d_{i,j} > 0\big\}$. Soit H le nombre maximum d'anneaux que l'on s'autorise[9] ; notons $R_h, 1 \leq h \leq H$ le $h^{ème}$ anneau et définissons les

lui-même (pas de boucles).

[5] ADR : *Affectation of Demands to Rings* (affectation de demandes à des anneaux)

[6] "U" pour *Unitary* (unitaire)

[7] "L" pour *Link* (lien)

[8] k-EP : *k-Edge Partitioning Problem* (partitionnement en ensembles d'au plus k arêtes)

[9] Cette borne n'est en fait pas une restriction : il suffit d'un anneau par demande ; ainsi en prenant, par exemple, $H = \frac{n(n-1)}{2}$, on conserve au problème toute sa généralité.

variables suivantes :

$$x_{ij}^h = \begin{cases} 1 & \text{si la demande } d_{ij} \text{ est affectée à } R_h \\ 0 & \text{sinon} \end{cases}$$

$$y^h = \begin{cases} 1 & \text{si au moins une demande est affectée à } R_h \\ 0 & \text{sinon} \end{cases}$$

$$z_i^h = \begin{cases} 1 & \text{si le nœud } i \text{ est raccordé à } R_h \\ 0 & \text{sinon} \end{cases}$$

L'objectif est alors :

$$\mathcal{FO} = \min \left(\sum_{h=1}^{H} r y^h + \sum_{h=1}^{H} \sum_{i=1}^{n} l z_i^h \right) \tag{1.1}$$

sous les contraintes :

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} x_{ij}^h d_{ij} \leq C y^h \qquad \forall h \in \{1 \dots H\} \tag{1.2}$$

$$\sum_{h=1}^{H} x_{ij}^h = 1 \qquad \forall (i,j) \in \Omega \tag{1.3}$$

$$x_{ij}^h \leq z_i^h \qquad \forall (i,j) \in \Omega, \forall h \in \{1 \dots H\} \tag{1.4}$$

$$x_{ij}^h \leq z_j^h \qquad \forall (i,j) \in \Omega, \forall h \in \{1 \dots H\} \tag{1.5}$$

$$x_{ij}^h, y^h, z_i^h \in \{0,1\} \qquad \forall (i,j) \in \Omega, \forall h \in \{1 \dots H\} \tag{1.6}$$

La fonction-objectif (1.1) minimise les coûts engendrés respectivement par la création de l'anneau R_h (quand $y^h = 1$) et par l'appartenance du nœud i à l'anneau R_h (quand $z_i^h = 1$).

La contrainte (1.2) traduit non seulement que la somme des demandes affectées à l'anneau R_h ne peut dépasser sa capacité C, mais force aussi la création de cet anneau si une demande non nulle lui a été affectée. La contrainte (1.3) oblige chaque demande à être satisfaite une et une seule fois. Les contraintes (1.4) et (1.5) forcent les valeurs des variables z_i^h à 1 si une demande d'extrémité i est affectée à l'anneau R_h. Dans une solution optimale, où on gagne à avoir $z_i^h = 0$ quand $l > 0$ (ce qui est le cas dans la réalité), on aura donc $z_i^h = 1$ si et seulement si l'anneau R_h contient le nœud i, et donc z_i^h a bien, à l'optimum, la signification voulue. Les contraintes d'intégralité (1.6) assurent tout d'abord que les demandes ne sont pas réparties sur plusieurs anneaux ($x_{ij} \in \{0,1\}$) ; ensuite, elles assurent la cohérence du sens des variables y^h et z_i^h (un anneau existe totalement ou pas du tout, et un nœud est raccordé à un anneau ou pas). Remarquons que, dans ce modèle, pour toute solution réalisable, l'ensemble des anneaux créés est une partition des arêtes du graphe des demandes.

Cette modélisation permet de bien poser le problème, cependant elle ne permet pas une résolution pratique d'instances réelles : pour un graphe de n nœuds, nous avons $\mathcal{O}(n^2)$ contraintes et surtout nous voulons des solutions en nombres entiers, ce qui fait très vite exploser les temps de calcul.

1.3.2 Problème de recouvrement

Nous présentons maintenant une autre modélisation sous forme de problème de recouvrement du graphe des demandes. Cette formulation est bien adaptée, entre autre, pour une approche de type "génération de colonnes". Nous notons E l'ensembles des arêtes du graphe des demandes et d_e la demande sur l'arête $e \in E$ et nous posons :

$$\begin{aligned} \mathcal{R} &= \left\{ R \subset G, \sum_{e \in R} d_e \leq C \right\} \\ c_R &= \text{nombre de sommets dans le sous-graphe } R \in \mathcal{R} \\ a_{eR} &= \begin{cases} 1 & \text{si l'arête } e \in E \text{ appartient à } R \in \mathcal{R} \\ 0 & \text{sinon} \end{cases} \end{aligned}$$

L'objectif est alors :

$$\mathcal{FO} = \min \sum_{R \in \mathcal{R}} (lc_R + r) x_R \tag{1.7}$$

avec les contraintes :

$$\sum_{R \in \mathcal{R}} a_{eR}.x_R = 1 \qquad \forall e \in E \tag{1.8}$$

$$x_R \in \{0, 1\} \qquad \forall R \in \mathcal{R} \tag{1.9}$$

Notons tout d'abord que l'ensemble $\mathcal{R}$ est tout simplement l'ensemble de tous les anneaux possibles. À partir de là, l'objectif (1.7) revient à trouver un ensemble d'anneaux de coût global minimum, chaque anneau R ayant un coût de $lc_R + r$.

La contrainte (1.8) assure que chaque arête est couverte exactement une fois et donc que chaque demande sera satisfaite. La contrainte (1.9) traduit qu'un candidat pour être un anneau est pris complètement ou pas du tout. Cela assure notamment que les demandes ne sont pas séparées sur plusieurs anneaux et comme pour le modèle précédent une solution réalisable sera une partitions des arêtes du graphe des demandes.

Chapitre 2

Propriétés du problème et de ses solutions

Dans ce chapitre, nous établissons tout d'abord la complexité du problème ADR et de certains de ses sous-problèmes. Nous présentons ensuite certaines propriétés des solutions, notamment des bornes sur les coûts pour différents types de solutions qui seront utilisées au chapitre 3 pour garantir des performances d'algorithmes.

2.1 Complexité du problème

Dans le cas où $l = 0$, la structure du graphe des demandes n'a aucune influence sur la valeur de la solution. Ainsi, seules les valeurs des demandes importent. Dans le cas où les demandes sont toutes identiques, de valeur d, nous obtenons trivialement une solution optimale en mettant $\lfloor C/d \rfloor$ arêtes par anneau (si on a m arêtes, la valeur optimale est alors de $\left\lceil \frac{m}{\lfloor C/d \rfloor} \right\rceil r$). Dans le cas où les demandes ne sont pas uniformes, le problème se réduit immédiatement à un problème de Bin-Packing, connu pour être fortement $\mathcal{NP}$-difficile dans le cas général [7]. Pour un nombre d'anneaux fixé le problème reste $\mathcal{NP}$-difficile mais il existe des algorithmes pseudo-polynomiaux pour déterminer si une capacité permet de satisfaire ce nombre d'anneaux ; pour une capacité C fixée le problème devient polynomial par recherche exhaustive [7].

Pour le cas ADRL (*i.e.* demandes unitaires et $r = 0$), Brauner, Crama et Wynants [2] ont montré le résultat suivant[1] :

Lemme 2.1 *Le problème de décision :*
Instance : *un graphe G, une capacité C, une borne B*
Question : *est-il possible de couvrir G avec des sous-graphes G_i =*

[1] On trouve un résultat similaire dans [12].

(V_i, E_i) de G tels que les E_i forment une partition de E et que $\forall i$: $|E_i| \leq C$ et que $\sum_i |V_i| \leq B$?
est $\mathcal{NP}$-complet pour $C = 3$.

preuve :
Leur démonstration est une réduction à **EPIT**[2], qui a été montré $\mathcal{NP}$-complet par Holyer [15]. •

De ce résultat découle la complexité du problème ADRL :

Lemme 2.2 *Le problème ADRL (i.e. $r = 0$, demandes unitaires) est fortement $\mathcal{NP}$-complet dans le cas général.*

preuve :
Pour le problème de décision du lemme 2.1, nous avons : $\sum_i |V_i| \leq 2|E|$ pour toute instance. En conséquent nous pouvons borner B polynomialement en fonction de la taille de l'instance sans, en fait, limiter le problème. De plus la capacité C est elle aussi bornée polynomialement puisqu'elle est fixée à 3.
En d'autres termes, le problème de décision du lemme 2.1 est une restriction polynomiale de ADRL et elle est $\mathcal{NP}$-complète. Il s'en suit que ADRL est fortement $\mathcal{NP}$-complet dans le cas général. •

Nous aboutissons ainsi au résultat général pour ADR :

Théorème 2.1 *Le problème ADR est fortement $\mathcal{NP}$-complet dans le cas général.*

preuve :
Le problème ADRL est une restriction polynomiale de ADR qui est $\mathcal{NP}$-complète (lemme 2.2). En conséquent ADR est fortement $\mathcal{NP}$-complet. •

Le tableau 2.1 résume les résultats de complexité que nous avons obtenus pour ADR et ses sous-problèmes[3]. Rappelons que lorsque les demandes ne sont pas unitaires, le problème est une généralisation du Bin-Packing et qu'il est ainsi immédiatement difficile.

2.2 Bornes pour le coût d'une solution

Soit $G = (V, E)$ un graphe de demandes entières, strictement positives. $|E|$ dénote le nombre d'arêtes de G ; pour une demande $e \in E$, d_e est la demande sur e ; pour un sommet $i \in V$, $deg(i)$ est son degré, *i.e.* le nombre d'arêtes adjacentes à i.

[2]EPIT ou EPT : *Edge Partition Into Triangles* (partition d'arêtes en triangles).
[3]Les démonstrations des résultats de complexité qui ne sont pas données dans le présent chapitre appararaissent dans l'annexe A.

Soit une solution S composée de R anneaux (non vides) : $R_1 \dots R_R$. Pour un anneau R_i nous notons $L(R_i)$ le nombre de nœuds (distincts) dans R_i. Chacun de ces nœuds constitue un *raccord*. Ainsi, $L = \sum_{i=1}^{R} L(R_i)$ est le nombre total de raccords dans la solution S. Par définition, le coût de la solution S est :

$$c(S) = R.r + L.l$$

Il existe une borne triviale sur R, donnée par la propriété suivante :

Propriété 2.1 *Le nombre d'anneaux R dans une solution de ADR sur un graphe de demandes $G = (V, E)$ est borné par :*

$$R \geq \left\lceil \frac{\sum_{e \in E} d_e}{C} \right\rceil \geq \left\lceil \frac{|E|}{C} \right\rceil$$

preuve :

Pour la première inégalité, il suffit de constater que R anneaux peuvent satisfaire au plus une demande de $R.C$, que la demande totale est de $\sum_{e \in E} d_e$, et que R doit être entier. La deuxième inégalité est justifiée par l'hypothèse de demandes entières strictement positives. •

Remarque : dans le cas de demandes unitaires, les 2 bornes sont égales et il existe toujours des solutions réalisables qui atteignent cette borne (indépendamment de la topologie du graphe : il suffit de mettre C demandes par anneau, avec un dernier anneau éventuellement incomplet).

Passons maintenant à des bornes sur le nombre de raccords. Nous avons :

Propriété 2.2 *Le nombre de raccords à un anneau satisfaisant d demandes distinctes vérifie :*

$$\beta_d \quad \geq \quad \left\lceil \frac{1 + \sqrt{1 + 8d}}{2} \right\rceil$$

preuve :

Par définition d'un raccord, les demandes affectées à un anneau sont nécessairement des arêtes du graphe complet engendré par les raccords à cet anneau. Il s'en suit que ce graphe complet doit avoir au minimum d arêtes distinctes pour que d demandes soient satisfaites. Or, un graphe complet de v sommets a $\frac{v(v-1)}{2}$ arêtes ; pour nos β_d raccords cela donne donc :

$$\beta_d \geq \min \left\{ v \geq 0, \frac{v(v-1)}{2} \geq d \right\} = \left\lceil \frac{1 + \sqrt{1 + 8d}}{2} \right\rceil$$

•

Propriété 2.3 *Pour un graphe de demandes $G = (V, E)$, le nombre de raccords L dans une solution est borné par :*

$$L \geq |E|\sqrt{\frac{2}{C}} \tag{2.1}$$

$$L \geq \left\lceil \frac{1+\sqrt{1+8C}}{2} \right\rceil \frac{|E|}{C} \tag{2.2}$$

$$L \geq \sum_{i \in V} \left\lceil \frac{deg(i)}{C} \right\rceil \tag{2.3}$$

preuve :

La borne (2.1) est due à Goldschmidt, Hochbaum et Olinick [12] et n'est qu'une version faible de la borne (2.2).

Un anneau satisfaisant C demandes nécessite un minimum de β_C raccords (*cf* propriété 2.2). Comme le rapport $\frac{\beta_x}{x}$ est une fonction décroissante de x, le coût moyen minimum pour une arête est $\frac{\beta_C}{C}$. De ce fait, une solution coûte au moins $\frac{\beta_C}{C}|E|$, ce qui justifie la borne (2.2).

Il y a $deg(i)$ arêtes adjacentes à un sommet $i \in V$, donc ce sommet appartient au minimum à $\left\lceil \frac{deg(i)}{C} \right\rceil$ anneaux. Cela étant valable pour chaque sommet $i \in V$, il suffit de faire la somme sur V pour obtenir la borne (2.3).
•

Remarque : la borne (2.1) est toujours moins bonne que la borne (2.2), mais son écriture plus simple la rend plus pratique pour des calculs. De plus, cette différence tend vers 0 lorsque C tend vers l'infini et les bornes sont asymptotiquement équivalentes.

Pour chacune des bornes ci-dessus, il existe des graphes pour lesquels le nombre minimum de raccords pour une capacité donnée est cette borne. Cependant on constate, par exemple pour les bornes (2.1) et (2.2) qui ne sont atteintes que si le graphe de demandes est partitionnable en cliques de C arêtes, que cela est rare — voire n'arrive jamais en pratique. De ce fait, ces bornes sont souvent "assez loin" de la valeur réelle. Des bornes plus précises seront parfois données pour des graphes plus particuliers, par exemple :

Propriété 2.4 *Soit G un graphe connexe ayant n sommets dont le degré minimum est d'au moins 2. Soit une solution S de ADR sur G pour une certaine capacité. Soit R le nombre d'anneaux dans S. Le nombre de raccords L dans S vérifie :*

$$L \geq n + 2(R-1)$$

preuve :

Chaque nœud de G doit être raccordé à au moins un anneau, ce qui donne $L \geq n$. À cela il faut ajouter les raccords dus au partitionnement en anneaux : l'existence de R anneaux implique un minimum de $R-1$ *frontières* entre eux, *i.e.* que pour parcourir toutes les arêtes on doit changer au moins $R-1$ fois d'anneaux. Comme le graphe est connexe et que tous les degrés sont au moins de deux, chaque frontière doit séparer en deux la demande d'au moins deux sommets. De ce fait, chaque frontière implique 2 raccords supplémentaires par rapport aux n raccords obtenus si l'on pouvait ne raccorder un nœud qu'à un seul anneau. Le nombre total de raccords est donc bien, au minimum, de $L \geq n + 2(R-1)$. •

2.3 Solutions forescentes

Nous nous plaçons ici essentiellement dans le cadre ADRL, *i.e.* avec un coût de création d'anneau $r = 0$ et des demandes unitaires.

Lemme 2.3 *Soit un graphe G ayant m arêtes et une partition de ses arêtes en p arbres : $S = \{T_1 \dots T_p\}$. On note $|T_i|$ le nombre d'arêtes de l'arbre T_i et $K = \max\{|T_i|, 1 \leq i \leq p\}$. On a : S est une solution réalisable de ADRL sur G pour tout $C \geq K$ et son coût est de : $(m+p)l$*

preuve :

Chaque T_i est un anneau valable d'une solution de ADRL sur G, puisque $|T_i| \leq C, \forall i \in \{1, \dots, p\}$. De plus, par définition, S est une partition des arêtes de G. Donc S est une solution réalisable de ADRL sur G.

Un arbre de x arêtes coûte $x+1$ raccords. En conséquence, le coût de la solution est : $\sum_i (|T_i|+1)l = (p + \sum_i |T_i|)l$. Comme les T_i forment une partition des arêtes de G : $\sum_i |T_i| = m$, d'où le résultat : $L = (p+m)l$. •

Si on se place dans le cadre ADR, *i.e.* en tenant compte du coût de création d'un nœud, on obtient simplement une borne : le coût de la solution est $c(S) \leq (n+p)l + r.p$. En effet rien n'empêche, *a priori*, que plusieurs arbres puissent être rassemblés en un seul anneau. Sur une même forescence on peut ainsi avoir plusieurs solutions de coût distinct pour ADR. N'ayant pas ce problème lorsque $r = 0$, on obtient :

Théorème 2.2 *Dans un graphe G de demandes unitaires sans cycle de longueur inférieure (ou égale) à C, une solution optimale de ADRL est une forêt avec un nombre minimum de composantes connexes.*

preuve :

Si G n'a pas de cycle de longueur inférieure à C, alors un anneau ne peut pas contenir de cycle, et donc chaque anneau est une forêt. Dans une

solution S, il y a $p(S)$ arbres — *i.e.* $p(S)$ composantes connexes — pour un coût de $n + p(S)$, d'après le lemme 2.3. Or n, le nombre d'arêtes de G, est une constante indépendante de la solution envisagée, donc S optimale minimise bien $p(S)$ parmi les solutions réalisables, d'où le résultat. •

Corollaire 2.2.1 *Dans un graphe G de demandes unitaires sans cycle de longueur inférieure (ou égale) à C, le nombre de raccords L dans une solution optimale de ADRL est borné par :*

$$L \geq \frac{(C+1)|E|}{C}$$

preuve :

D'après le corollaire précédent, une solution optimale d'un tel graphe est une forêt avec un nombre minimum de composantes connexes. Ce minimum est au moins égal au nombre minimum d'anneaux dans une solution, $\left\lceil \frac{|E|}{C} \right\rceil$ (*cf* propriété 2.1), qui ne peut être atteint que si chaque anneau est un arbre. Dans ce cas, le nombre moyen minimum de raccords est de $\frac{C+1}{C}$ par arête, d'où le résultat. •

2.4 Anneau de coût minimum et solution optimale

Un anneau coûte d'autant moins qu'il contient peu de sommets. De ce fait, les cliques sont peu coûteuses. Cependant la recherche systématique de tels anneaux n'amène pas toujours à des solutions optimales, comme le montre l'exemple de la figure 2.1(a). De plus, même un anneau de coût minimum qui ne déconnecte pas le graphe n'appartient pas toujours à une solution optimale. La figure 2.1(b) montre un tel exemple, sur un arbre, pour $C = 3$ (dans ce cas, un anneau de coût minimum est un arbre de 3 arêtes).

Il y a ici une perspective intéressante, résumée par la question : "Quand pouvons-nous, être sûrs qu'un anneau appartient (ou n'appartient pas) à une solution optimale ?"[4].

2.5 Nombre d'anneaux dans une solution optimale

Une solution optimale du problème ADR doit trouver un équilibre entre minimiser le nombre d'anneaux et minimiser le nombre de raccords. Nous pouvons légitimement nous demander dans quel cas il existe une solution optimale qui minimise le nombre d'anneaux parmi toutes les solutions réalisables. Nous allons nous intéresser ici au cas de demandes unitaires (AUDR), et d'abord lorsque $C = 3$:

[4] L'étude de la partition d'un arbre en sous-arbres de 3 arêtes (*cf* annexe B.1) donnent quelques réponses dans des cas restreints.

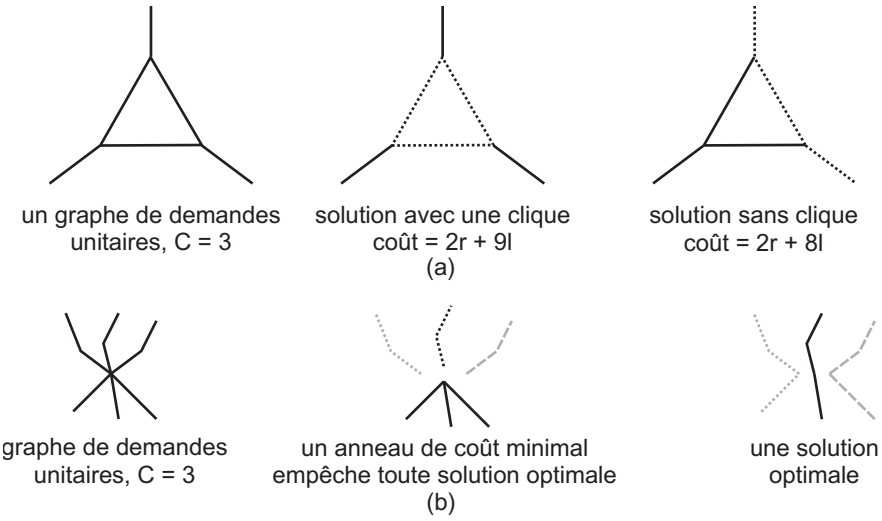

FIG. 2.1 – Exemples où (a) la densité d'un anneau empêche d'atteindre l'optimal ; (b) la sélection d'un anneau de coût minimum ne déconnectant pas le graphe empêche d'atteindre l'optimal.

Propriété 2.5 *Pour $C = 3$ et $r > l$, les solutions optimales ont un nombre d'anneaux minimum parmi toutes les solutions réalisables.*

preuve :

Si S ne minimise pas le nombre d'anneaux parmi toutes les solutions réalisables, c'est qu'il existe $p \geq 2$ anneaux contenant chacun $r_i \leq 2$ arêtes.

S'il existe un anneau i tel que $r_i = 1$ alors, comme il existe au moins un autre anneau incomplet j et nous pouvons fusionner i et j pour obtenir une solution réalisable avec un gain de $r > 0$ par rapport à S, ce qui contredit son optimalité. Donc : $\forall i, 1 \leq i \leq p : r_i = 2$.

Comme chaque anneau incomplet a 2 arêtes et qu'on peut diminuer le nombre total d'anneaux, il y a au moins 3 anneaux incomplets : i, j, k. Coupons i en 2 et mettons une arêtes dans j, l'autre dans k. La solution ainsi obtenue est toujours réalisable. De plus, si i était formé de 2 arêtes non adjacentes, nous aurions un gain d'au moins r, ce qui contredirait S optimale ; si i était formé de 2 arêtes adjacentes, nous aurions un gain d'au moins $r - l > 0$, ce qui contredirait aussi que S est optimale.

En définitive, si S est optimale, S minimise le nombre d'anneaux. •

Il faut malheureusement constater que cela n'est pas toujours le cas, comme l'affirme la propriété suivante :

Propriété 2.6 *Soit une capacité $C \geq 3$. Si $r < C.l$ alors il existe un graphe connexe G de demandes unitaires tel qu'aucune solution optimale de ADR sur G ne minimise le nombre d'anneaux.*

preuve :

Prenons comme graphe G un graphe construit comme suit : nous partons d'un cycle c_0 de C arêtes ; sur chacun des sommets de ce cycle, nous greffons un cycle de C arêtes, et sur un des sommets de chaque cycle, autre que celui commun à c_0, nous greffons une $(C-1)$-chaîne. Nous obtenons ainsi un graphe formé de $C+1$ cycles de C arêtes et C chaînes de $C-1$ arêtes, soit un total de $2C^2$ arêtes (*cf* figure 2.2).

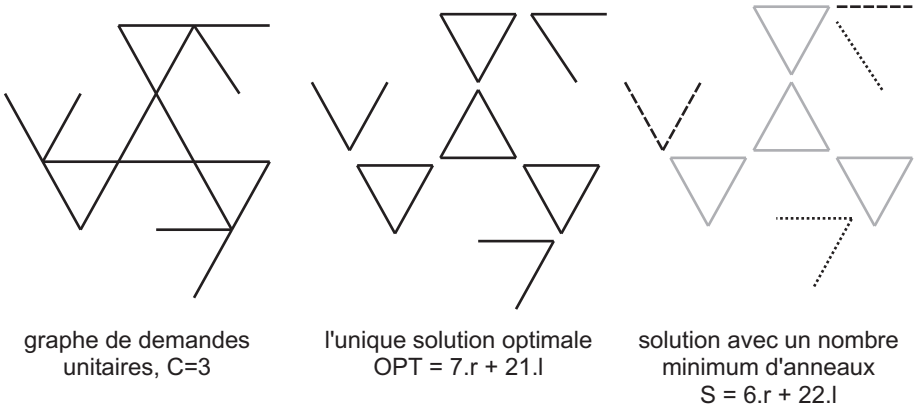

FIG. 2.2 – Un exemple où une solution optimale de ADR ne minimise pas le nombre d'anneaux ($r < 3l$).

Une solution réalisable est composée d'au moins $2C$ anneaux, et ce nombre est le minimum n'atteignable qu'en mettant C arêtes par anneau. Les seuls cycles (simples) de G ont C arêtes et ainsi si la solution contient un cycle, celui-ci forme un anneau complet de coût $r + lC$. Tout anneau qui n'est pas un cycle est une forêt de p composantes (connexes) et a donc au moins $C + p$ raccords s'il contient C arêtes (*cf* lemme 2.3).

Prenons une solution réalisable sur G qui minimise le nombre d'anneaux et notons-la $S_i, i \in \{0, \ldots, C+1\}$, où i est le nombre de cycles dans S.

Si S_i prend le cycle central comme anneau (donc $i \geq 1$), cela déconnecte le graphe en C composantes connexes de $2C-1$ arêtes. Les arêtes de ces composantes sont regroupées en $2C-1$ anneaux dont au plus C peuvent être formés d'une seule composante connexe (puisqu'aucune composante connexe n'a $2C$ arêtes) et parmis ceux-ci il y a $i-1$ cycles. S_i est donc composée au mieux de i cycles de coût $r + lC$, de $C - (i-1) = C - i + 1$ arbres de C arêtes de coût $r + (C+1)l$ et de $2C - i - (C - i + 1) = C - 1$ forêts d'au moins 2 composantes de coût au moins $r + l(C+2)$. Globalement, le coût de S_i vérifie : $c(S_i) \geq r(i + C - i + 1 + C - 1) + l\,(i.C + (C - i + 1)(C + 1) + (C - 1)(C + 2)) = 2rC + l(2C^2 + 3C - i - 1)$, d'où, pour une S_i ayant le cycle central comme anneau : $c(S_i) \geq 2rC + l(3C^2 - 2C - 2)$.

Si S_i n'a pas le cycle central comme anneau (donc $i \leq C$), ses i cycles déconnectent G en $i+1$ composantes connexes : i chaînes de $(C-1)$ arêtes et une composante de $2C^2 - iC - i(C-1) = 2C^2 - 2iC + i$ arêtes. Cette dernière est la seule sur laquelle on peut avoir des anneaux formés d'une seule composante, en dehors des cycles, et on peut en avoir un maximum de $\left\lfloor \frac{2C^2-2iC+i}{C} \right\rfloor = 2(C-i)$ si $i < C$. Si $i = C$, cela reste valable car alors la seule composante connexe d'au moins C arêtes est justement le cycle central interdit, ce qui fait qu'il n'y aucun anneau autre que les i cycles qui soit connexe dans S_i. De ce fait une S_i qui n'a pas le cycle central comme anneau est donc au mieux composée de i cycles, $2(C-i)$ arbres de C arêtes et $2C - i - 2(C-i) = i$ forêts d'au moins 2 composantes ; le coût d'une telle S_i vérifie donc : $c(S_i) \geq 2rC + l(iC + 2(C-i)(C+1) + i(C+2)) = 2rC + l(2C^2 + 2C)$.

Pour $C \geq 3$, $3C^2 - 2C - 2 \geq 2C^2 + 2C$ et donc, pour tout S_i, quelle utilise ou non le cycle central comme anneau (*i.e.* pour toute solution minimisant le nombre d'anneaux), on a : $c(S_i) \geq 2rC + l(2C^2 + 2C)$.

Considérons maintenat la solution S consistant à prendre pour anneau chaque cycle et chaque chaîne. Le coût de cette solution est : $c(S_1) = (C + 1 + C)r + ((C+1)C + C.C)l = (2C+1)r + (2C^2 + C)l$.

On a : $c(S_i) - c(S) \geq 2rC + (2C^2 + 2C)l - r(2C+1) - (2C^2 + C)l = lC - r$, et donc, si $r < lC$, S est meilleure que S_i, *i.e.* qu'une solution qui minimise le nombre d'anneaux n'est pas optimale. •

La propriété suivante donne un critère permettant de savoir s'il est avantageux de supprimer un anneau — voire de minimiser le nombre d'anneaux :

Propriété 2.7 *Soit G un graphe de demandes unitaires ; soit S une solution de ADR sur G pour une capacité C qui ne minimise pas le nombre d'anneaux. Soit $R_\alpha \in S$ un anneau de $\alpha < C$ arêtes. Si :*

$$r \quad > \quad \left(2\alpha - \left\lceil \frac{1+\sqrt{1+8\alpha}}{2} \right\rceil\right) l \tag{2.4}$$

alors il est intéressant de supprimer R_α. En particulier, si l'inégalité (2.4) est vraie pour $\alpha = C - 1$, une solution qui ne minimise pas le nombre d'anneaux n'est pas optimale.

preuve :

Par hypothèse S ne minimise pas le nombre d'anneaux et donc nous pouvons déplacer toutes les arêtes d'un anneau incomplet $R_\alpha, \alpha < C$, dans d'autres anneaux afin de gagner r. Une telle opération a un coût : d'après la propriété 2.2 il y avait un minimum de $\beta_\alpha = \left\lceil \frac{1+\sqrt{1+8\alpha}}{2} \right\rceil$ raccords à l'anneau R_α, ce qui s'ajoute au gain de r. D'un autre coté chacune des α arêtes

peut engendrer jusqu'à deux nouveaux raccords dans l'anneau où elle est déplacée, ce qui fait une perte pouvant aller jusqu'à 2α. Globalement le gain (éventuellement négatif) de l'opération est donc au minimum de : $r + (\beta_\alpha - 2\alpha)l$. Cette opération est donc toujours intéressante si $r > (2\alpha - \beta_\alpha)l$.

De plus, une solution ne minimisant pas le nombre d'anneaux à au moins un anneau ne contenant que $\alpha \leq C - 1$ arêtes. Donc, si l'inégalité (2.4) est vraie pour $\alpha = C-1$, il sera toujours intéressant de supprimer un tel anneau, tant que cela est possible. Dans ce cas, une solution optimale minimisera le nombre d'anneaux. •

Restrictions					**Complexité**
C	r	l	demandes	graphe	
-	-	-	unitaires	chaîne	polynomial (trivial)
-	-	-	unitaires	cycle	polynomial (trivial)
-	-	-	unitaires	étoile	polynomial (trivial)
-	-	$l = 0$	unitaires	-	polynomial (trivial)
$C = 2$	-	-	-	-	polynomial - $\mathcal{O}(m)$
$C = 3$	-	-	unitaires	grille	polynomial - $\mathcal{O}(m)$
$C = 3$	-	-	unitaires	$K_{n,p}$	polynomial - $\mathcal{O}(m)$
$C = 3$	-	-	unitaires	arbre	polynomial - $\mathcal{O}(m)$
C pair	-	-	unitaires	$K_{2,p}$	polynomial - $\mathcal{O}(m)$
$C \geq 3$	-	$l = 0$	-	-	$\mathcal{NP}$-complet
$C \geq 3$	$r = 0$	-	unitaires	-	fortement $\mathcal{NP}$-complet
$C \geq 3$	-	-	-	-	fortement $\mathcal{NP}$-complet

TAB. 2.1 – Synthèse des résultats de complexité pour ADR (m est le nombre d'arêtes du graphe des demandes).

Chapitre 3

Algorithmes à performances garanties pour ADRL

Ce chapitre présente des approximations avec garantie de performances pour le problème ADRL (*i.e.* demandes unitaires et coût de création d'un anneau $r = 0$).

3.1 Couverture par des 2-chaînes

Brauner, Crama et Wynants [2][1] se sont basés sur l'algorithme de Masuyama et Ibaraki [17] de partitionnement d'un graphe en 2-chaînes[2] pour établir une première heuristique avec garantie de performance :

Lemme 3.1 *L'algorithme de Masuyama et Ibaraki de partitionnement des arêtes d'un graphe en 2-chaînes est une α-approximation de ADRL pour tout graphe G connexe et une capacité $C \geq 3$, avec :*

$$\alpha = \frac{3C}{2\left\lceil \frac{1+\sqrt{1+8C}}{2} \right\rceil} \leq \frac{3}{2}\sqrt{\frac{C}{2}}$$

De plus : cette heuristique a un temps d'exécution linéaire en le nombre d'arêtes.

preuve :

D'après les travaux de Masuyama et Ibaraki[2], nous savons que l'ensemble E des arêtes d'un graphe connexe est partitionnable en $\left\lfloor \frac{|E|}{2} \right\rfloor$ 2-chaînes et nous connaissons un algorithme linéaire qui effectue cette partition.

[1] Ce résultat se retrouve aussi dans [12]

[2] Les résultats de Masuyama et Ibaraki sont rappelés dans l'annexe A.1 où il est montré qu'ils permettent une résolution exacte de ADR lorsque $C = 2$.

Le coût d'une arête dans une 2-chaîne est de $\frac{3}{2}$; ainsi, en notant $\mathcal{MI}$ le coût de la solution donnée par l'algorithme de Masuyama et Ibaraki, si $|E|$ est pair le graphe est couvert exclusivement par des 2-chaînes et alors $\mathcal{MI} = \frac{3}{2}|E|$. Si $|E|$ est impair il y a une arête solitaire e et des 2-chaînes. Comme le graphe est connexe, e est adjacente à une 2-chaîne et ajouter e à cette 2-chaîne pour créer un anneau R fait toujours une solution réalisable ($C \geq 3$). Le coût de cet anneau R est alors au plus de $\frac{4}{3}$ par arête. Nous avons alors : $\mathcal{MI} = \frac{3}{2}(|E| - 3) + 3\frac{4}{3} \leq \frac{3}{2}|E|$.

Dans tous les cas, donc, $\mathcal{MI} \leq \frac{3}{2}|E|$. Or le coût minimum d'une solution est : $\mathcal{OPT} \geq \left\lceil \frac{1+\sqrt{1+8C}}{2} \right\rceil \frac{|E|}{C}$ (*cf* inégalité (2.2) de la propriété 2.3 page 16). De ce fait, nous avons bien une α-approximation avec pour α la valeur maximale pour toutes les instances du rapport $\frac{\mathcal{MI}}{\mathcal{OPT}}$, soit $\alpha = \frac{3|E|}{2} \frac{C}{|E| \left\lceil \frac{1+\sqrt{1+8C}}{2} \right\rceil}$.

•

3.2 Couverture par des arbres

Afin d'améliorer l'approximation, au lieu de se contenter de 2-chaînes, comme au paragraphe précédent, nous allons maintenant couvrir le graphe des demandes G par des arbres. Pour appliquer les algorithmes suivants, il ne faudra pas travailler sur G lui-même, mais sur $T(G)$: un arbre obtenu en ouvrant les cycles de G et dont les arêtes sont en bijection avec celles de G de manière à ce qu'à toute solution de ADR sur $T(G)$ corresponde une solution sur G de coût au pire égal[3].

Tout d'abord il existe un algorithme linéaire de résolution exacte de AUDR (*i.e.* demandes unitaires) pour un arbre de m demandes et une capacité $C = 3$ qui garantit que les solutions obtenues n'ont pas plus de $\frac{3}{2}m + \frac{1}{2}$ raccords[4]. Nous pouvons nous servir de ce résultat en tant qu'approximation pour un graphe G quelconque et une capacité $C \geq 3$, en résolvant ADR sur $T(G)$ pour $C = 3$. Nous obtenons :

Lemme 3.2 *L'algorithme de partitionnement d'un arbre en 3-arbres étendu aux graphes quelconques donne une α-approximation de ADRL pour tout graphe connexe et une capacité $C \geq 3$, avec :*

$$\alpha = \frac{3C}{2\left\lceil \frac{1+\sqrt{1+8C}}{2} \right\rceil} + \frac{C}{2m\left\lceil \frac{1+\sqrt{1+8C}}{2} \right\rceil} \leq \left(\frac{3}{2} + \frac{1}{2m}\right)\sqrt{\frac{C}{2}}$$

De plus : cette heuristique a un temps d'exécution linéaire en le nombre d'arêtes m.

[3] Un tel $T(G)$ se calcule en temps linéaire ; une construction possible est donnée dans l'annexe C.

[4] Un tel algorithme est donné dans l'annexe A.6 ; la borne sur le nombre de raccords provient du corollaire A.2.1.

preuve :

À toute solution réalisable sur $T(G)$ correspond une solution réalisable sur G, et comme $C \geq 3$ à la solution renvoyée par l'algorithme de partitionnement correspond bien une solution réalisable sur G pour C.

Nous venons de rappeler que le coût de cette solution est inférieur à $\frac{3}{2}m + \frac{1}{2}$; de plus la propriété 2.3 page 16 nous donne une borne inférieure pour ce coût : $\left\lceil \frac{1+\sqrt{1+8C}}{2} \right\rceil \frac{m}{C} \leq m\sqrt{\frac{2}{C}}$, ce qui termine la démonstration. •

Remarque : Le ratio de cette heuristique est légèrement moins bon que celui de l'heuristique basée sur des 2-chaînes. Cependant, cela ne traduit que le pire cas (atteignable) et il faut s'attendre à de meilleures performances en pratique pour cette nouvelle heuristique.

Pour une capacité $C \geq 4$, nous pouvons obtenir de meilleurs résultats. En effet, il existe un algorithme qui partitionne un arbre en 3-arbres et 4-arbres (plus une éventuelle 2-chaîne ou arête)[5] et qui fournit une nouvelle approximation avec performances garanties :

Lemme 3.3 *L'algorithme de partitionnement d'un arbre en 3- ou 4-arbres étendu aux graphes quelconques donne une α-approximation de ADRL pour tout graphe connexe et une capacité $C \geq 4$, avec :*

$$\alpha = \frac{4C}{3\left\lceil \frac{1+\sqrt{1+8C}}{2} \right\rceil} + \frac{2C}{3m\left\lceil \frac{1+\sqrt{1+8C}}{2} \right\rceil} \leq \left(\frac{4}{3} + \frac{2}{3m}\right)\sqrt{\frac{C}{2}}$$

De plus : cette heuristique a un temps d'exécution linéaire en le nombre d'arêtes m.

preuve :

Une solution donnée par un algorithme de couverture par des 3- ou 4-arbres est au pire formée par des 3-arbres plus une 2-chaîne ou arête solitaire. Cela nous donne une borne supérieure sur le nombre de raccords : $L \leq \min\left\{3 + \frac{4}{3}(m-2), 2 + \frac{4}{3}(m-1)\right\} = \frac{4}{3}m + \frac{2}{3}$. La propriété 2.3 page 16 donne une borne inférieure sur le coût d'une solution optimale, d'où nous déduisons la borne annoncée. •

Goldschmidt, Hochbaum et Olinick [12] ont généralisé cette démarche de couverture par des arbres en concevant une procédure de couverture d'un arbre par des arbres ayant p arêtes, $\frac{C}{2} \leq p \leq C$. Leurs résultats sont résumés dans le lemme suivant :

[5] Un tel algorithme est donné dans l'annexe B.2.

Lemme 3.4 *L'algorithme de partitionnement en arbres ayant p arêtes,* $\frac{C}{2} \leq p \leq C$*, étendu aux graphes quelconques donne une* α*-approximation de ADRL pour tout graphe connexe et une capacité* $C \geq 3$*, avec :*

$$\alpha = \sqrt{\frac{C}{2}} + \sqrt{\frac{2}{C}}$$

De plus : cette heuristique a un temps d'exécution linéaire en le nombre d'arêtes.

3.3 Heuristique eulérienne

Nous étudions ici les performances d'une heuristique basée sur la recherche de chemin eulérien. L'idée de base est simple : puisqu'un chemin eulérien, s'il existe, couvre toutes les arêtes du graphe, il suffit de le partitionner en sous-chaînes d'au plus C arêtes pour obtenir une solution réalisable de ADRL.

Plaçons nous d'abord, naturellement, dans le cas où le graphe des demandes est eulérien. Dans ce cas, il existe un chemin de longueur m passant une et une seule fois par chaque arête. Nous pouvons ensuite couper ce chemin de façon optimale[6] pour obtenir une solution de ADRL. Replongée dans le graphe, cette solution peut voir son coût baisser, en raison de cycles, et dans le pire des cas son nombre de raccords vérifie : $L \leq m + \lceil \frac{m}{C} \rceil$.

De cela on déduit le résultat suivant :

Lemme 3.5 *Pour un graphe de demandes eulérien et une capacité C, il existe une* α*-approximation de ADRL avec :*

$$\alpha = \frac{\left(m + \lceil \frac{m}{C} \rceil\right) C}{m \left\lceil \frac{1+\sqrt{1+8C}}{2} \right\rceil} \leq \left(1 + \frac{1}{C} + \frac{1}{m.C}\right) \sqrt{\frac{C}{2}}$$

De plus : cette heuristique a un temps d'exécution linéaire en le nombre d'arêtes m.

Ce résultat s'étend aux graphes connexes quelconques. Soit δ le nombre de sommets de degré impair. Dans la recherche d'un chemin eulérien, les sommets impairs impliqueront un maximum de $\frac{\delta}{2} - 1$ coupures[7] si nous prenons soin de commencer après chaque coupure, ainsi qu'au début, par un sommet de degré impair s'il en reste dans le graphe des demandes insatisfaites. Cela donne le résultat suivant :

[6] La résolution exacte de ADR sur une chaîne de demandes unitaire est donné dans l'annexe A.2. La propriété A.1 donne le nombre de raccords dans une solution optimale : $m + \lceil \frac{m}{C} \rceil$.

[7] Rappelons que δ, nombre de sommets de degré impair, est toujours pair.

Lemme 3.6 *Pour un graphe de demandes connexe avec δ sommets de degré impair et une capacité C, l'heuristique basée sur la recherche de chemin eulérien permet d'obtenir une α-approximation de ADRL avec :*

$$\begin{aligned} \alpha &\leq \left(1 + \frac{1}{C} + \frac{1}{m.C} + \frac{\delta}{2m} - \frac{1}{m}\right)\sqrt{\frac{C}{2}} \\ &\leq \left(1 + \frac{1}{C} + \frac{1}{m.C} + \sqrt{\frac{2}{m}} - \frac{1}{m}\right)\sqrt{\frac{C}{2}} \end{aligned}$$

De plus : cette heuristique a un temps d'exécution linéaire en le nombre d'arêtes m.

3.4 Synthèse

Le tableau 3.1 résume les résultats d'approximation avec garantie de performance pour ADRL. Certains résultats n'apparaissent pas explicitement dans ce qui précède, mais tous les résultats nécessaires pour les établir sont dans ce rapport[8].

Ces approximations sont à la base de plusieurs heuristiques gloutonnes de résolution d'ADR, comme nous le verrons au chapitre 4. Nous pouvons déjà remarquer que les performances d'algorithmes partitionnant un graphe en anneaux de petite taille, tels 2-chaînes ou 3-arbres, peuvent être facilement et significativement améliorées en pratique, tout simplement en fusionnant ces anneaux entre eux[9].

[8] Pour mieux se rendre compte de la qualité de chaque approximation, on trouvera dans l'annexe E des valeurs numériques pour α dans différents cas.

[9] Une méthode pour une telle fusion est décrite dans l'annexe D.

Approximation	Ratio et commentaires	
Couverture par des 2-chaînes	$\alpha = \frac{3}{2}\sqrt{\frac{C}{2}}$	(1.a)
	$\alpha = \frac{3}{2}\frac{C}{C+1}$	(2.a)
Couverture par des 3-arbres	$\alpha = \left(\frac{3}{2} + \frac{1}{2m}\right)\sqrt{\frac{C}{2}}$	(1.a)
	$\alpha = \frac{3}{2}\left(1 + \frac{1}{3m}\right)\frac{C}{C+1}$	(2.a)
Couverture par des 3-arbres et des 4-arbres	$\alpha = \left(\frac{4}{3} + \frac{2}{3m}\right)\sqrt{\frac{C}{2}}$	(1.b)
	$\alpha = \frac{4}{3}\left(1 + \frac{2}{m}\right)\frac{C}{C+1}$	(2.b)
Couverture par des arbres de $\frac{C}{2} \leq p \leq C$ arêtes	$\alpha = \sqrt{\frac{C}{2}} + \sqrt{\frac{2}{C}}$	(1.a)
	$\alpha = \frac{C+2}{C+1} + \frac{C}{(C+1)m}$	(2.a)
Heuristique eulérienne	$\alpha = \left(1 + \frac{1}{C} + \frac{1}{m.C} + \sqrt{\frac{2}{m}} - \frac{1}{m}\right)\sqrt{\frac{C}{2}}$	(1.a)
	$\alpha = \left(1 + \frac{1}{C} + \frac{1}{m.C}\right)\sqrt{\frac{C}{2}}$	(3.a)
	$\alpha = \left(1 + \frac{1}{C} + \frac{1}{m.C} + \frac{\delta}{2m} - \frac{1}{m}\right)\sqrt{\frac{C}{2}}$	(4.a)

(1) graphes connexes
(2) graphes connexes, sans cycles de C (ou moins) arêtes
(3) graphes eulériens
(4) graphes connexes avec δ sommets de degré impair

(a) $C \geq 3$ (b) $C \geq 4$

TAB. 3.1 – Synthèse des résultats d'approximation avec garantie de performance pour ADRL ; m est le nombre d'arêtes du graphe des demandes et toutes ces heuristiques sont en $\mathcal{O}(m)$.

Chapitre 4

Algorithmes gloutons de résolution de ADR

Dans ce chapitre, nous présentons quelques algorithmes gloutonnes pour la résolution de ADR. Nous discutons ensuite de leurs performances expérimentales.

4.1 Description des algorithmes

4.1.1 Adaptations d'un algorithme pour le Bin-Packing

En partant du constat que pour $l = 0$ (et donc approximativement quand $l << r$) le problème ADR se réduit à un Bin-Packing, nous avons réalisé nos premières heuristiques gloutonnes à partir du célèbre algorithme FFD[1] pour le Bin-Packing dont les performances ont été étudiées par Johnson *et al.* [16]. Le principe est à chaque fois de trier les arêtes par demande décroissante, puis de les ranger dans des anneaux selon la règle : "on met une arête dans le premier anneau qui peut la contenir" (Johnson *et al.* [16] ont montré qu'avec cette règle le nombre d'anneaux obtenus était au plus de $\frac{11}{9}x + 4$, où x est le nombre minimum d'anneaux dans une solution réalisable[2]).

Pour intégrer le coût de raccord d'un nœud, l, des variantes ont été essayées. La première consiste en l'affinement du tri : au sein des ensembles d'arêtes de même demande, les sommets sont triés selon un ordre arbitraire[3], dans l'espoir que des arêtes de même extrémité seront ainsi mises dans le même anneau.

[1] FFD : *First Fit Decreasing* (le premier qui convient, en décroissant).

[2] Ce ratio de $\frac{11}{9}$ reste l'un des meilleurs connus. À notre connaissance, le meilleur ratio actuel est obtenu par une version modifiée de FFD due à Garey et Johnson [8] avec une garantie de $\frac{71}{60}x + \frac{31}{6}$.

[3] Dans notre implantation les sommets sont nommés avec des entiers et ainsi c'est l'ordre sur les noms que nous avons pris.

La deuxième amélioration consiste à ne pas respecter scrupuleusement la règle FFD, mais de choisir, parmi les anneaux pouvant recevoir l'arête, le premier anneau qui maximise le nombre de nœuds communs.

En combinant les 2 tris possibles (raffiné ou non) et les 2 règles d'insertion possible (FFD modifiée ou non), nous obtenons ainsi 4 variantes d'algorithmes gloutons :

- `bin_packing1` : tri normal, règle FFD
- `bin_packing2` : tri raffiné, règle FFD
- `bin_packing3` : tri normal, règle FFD modifiée
- `bin_packing4` : tri raffiné, règle FFD modifiée

Remarque 1 : Il y a peu de garanties sur la qualité des solutions rendues par ces algorithmes, mais ceux-ci étant bâtis sur des algorithmes de Bin-Packing, ils sont *a priori* plus adaptés pour des cas où le coût de création d'un anneau domine celui de raccord d'un nœud (*i.e.* $r > l$).

Remarque 2 : Dans le cas où les demandes sont uniformes, les 4 versions renverront des solutions avec le même nombre d'anneaux et tous aussi remplis. Dans ce cas l'ordre de traitement des demandes devient primordial ; ainsi pour `bin_packing2` et `bin_packing4` les noms attribués arbitrairement aux sommets déterminerons le nombre de raccords, et donc la qualité de la solution !

4.1.2 Recherche du meilleur nœud

Tandis que les algorithmes dérivés du Bin-Packing privilégient plutôt un faible nombre d'anneaux, la deuxième approche adoptée s'attache plus à un faible nombre de raccords en ajoutant à chaque itération qu'un seul nouveau nœud, prometteur, à un anneau. Avant de donner cet algorithme précisons que la demande associée à un nœud est la somme des demandes incidentes à ce nœud ; de plus, lorsque nous parlons d'ajouter des arêtes à un anneau, nous le faisons toujours dans la limite de la capacité de cet anneau. Ceci étant dit, l'algorithme se déroule comme suit :

Algorithme 4.1

1. $i = 1$
2. tant qu'il reste des demandes non satisfaites faire :
   (a) créer l'anneau R_i en choisissant le nœud n_j ayant la plus grande demande insatisfaite et en affectant à R_i les demandes incidentes à n_j.
   (b) tant que R_i n'est pas plein, choisir le nœud n_k ayant la plus grande demande insatisfaite avec les nœuds déjà dans R_i et affecter les demandes $d_{n_k n_j}, n_j \in R_i$ à R_i.

(c) $i = i + 1$

Cet algorithme n'engendre aucun anneau satisfaisant une demande supérieure à C et ne s'arrête que lorsque toutes les demandes sont satisfaites. Comme il s'arrête toujours, il fournit bien, toujours, une solution réalisable de ADR.

4.1.3 Recherche de la meilleure demande

Tout comme l'heuristique précédente, l'algorithme de recherche de la meilleure demande est plutôt dédié à un faible nombre de raccords. En notant *occurs* la fonction qui à une arête (ou à la demande correspondante) associe le nombre d'arêtes qui lui sont incidentes dans l'anneau en construction, cet algorithme ajoute la demande insatisfaite qui maximise *occurs* parmi celles qui ne créent pas de dépassement de capacité. Lorsqu'un anneau est plein, un nouvel est créé, auquel est affectée une première demande arbitraire et on continue ainsi jusqu'à ce que toutes les demandes soient satisfaites, ce qui fait qu'on obtient bien une solution réalisable.

4.1.4 Adaptation de l'algorithme de Masuyama et Ibaraki

Rappelons que l'algorithme de Masuyama et Ibaraki de partition d'un graphe en 2-chaînes est une α-approximation de ADRL (*i.e.* demandes unitaires et $r = 0$) avec $\alpha = \frac{3}{2}\sqrt{\frac{C}{2}}$ (*cf* section 3.1). Cet algorithme s'adapte à des demandes quelconques et nous pouvons améliorer ses performances en effectuant une deuxième étape : concaténer les anneaux entre eux, dans la limite de la capacité C et de manière à diminuer au mieux le nombre de raccords[4].

4.1.5 Adaptation de l'heuristique eulérienne

En nous basant sur la construction d'un chemin eulérien dans un graphe, nous avons réalisé une nouvelle heuristique. Bien que respecter scrupuleusement la construction d'un chemin eulérien (autant qu'il est possible dans un graphe quelconque) permette de garantir certaines performances (*cf* section 3.3), nous avons sacrifié ces certitudes à l'espoir d'obtenir de bien meilleures performances en pratique en modifiant les règles de sélection d'une arête : nous ne sommes pas regardant sur la parité du sommet où on va, mais nous nous efforçons de re-utiliser les nœuds déjà utilisés dans l'anneau en construction. Nous obtenons l'algorithme suivant (nous notons V l'ensemble des nœuds du graphe et $occurs_r(n)$ le nombre d'arêtes incidentes au nœud n dans l'anneau r et $libre(r)$ la place libre dans l'anneau r) :

Algorithme 4.2

[4]Un algorithme pour une telle concaténation est donné dans l'annexe D.

1. $i = 1$
2. affecter une demande non satisfaite d_{kj} à l'anneau R_i.
   $cur = j$
3. tant que R_i n'est pas plein et qu'il y a une demande insatisfaite partant du nœud cur :
   (a) soit $\mathcal{Q} = \{j \in V | d_{cur,j}$ insatisfaite et $d_{cur,j} \leq libre(r)\}$
   soit j qui maximise $occurs_r(j), j \in \mathcal{Q}$.
   (b) $R_i = R_i \cup \{d_{cur,j}\}$
   $cur = j$
4. $i = i + 1$
   retourner en (2) s'il reste des demandes insatisfaites.

4.2 Performances comparées des algorithmes

Nous effectuons dans cette section une comparaison des solutions renvoyées par nos heuristiques gloutonnes ; lorsque nous parlons de "meilleure solution", il faudra donc comprendre "meilleure solution retournée par l'un des algorithmes gloutons", qui n'est pas nécessairement optimale. Cette comparaison est faite sur la base de tests sur des instances combinant différents types de graphes (aléatoires plus ou moins réalistes, $K_{n,p}$, etc) de demandes unitaires ou non et divers paramètres C, r, l ; ce que nous énonçons traduit des tendances mais peut être mis en défaut sur des instances particulières.

La figure 4.1 donne un exemple des différentes solutions obtenues selon l'algorithme glouton utilisé. Sur de petits graphes, il y a très peu, voire pas du tout, d'écarts entre les différentes solutions. Cependant, sur des graphes plus conséquents, les écarts s'accroissent.

L'heuristique pseudo-eulérienne et la recherche de la meilleure demande ont un très bon comportement (avec un avantage pour la première, surtout dans le cas de demandes unitaires) : les solutions quelles fournissent sont très souvent les meilleures et sont très rarement éloignées de la meilleure de plus de 10%, et cela quels que soient les demandes, la capacité, les coûts. Elles ont pour autre qualité d'être particulièrement performantes sur les instances les plus réalistes.

La recherche du meilleur nœud a un comportement beaucoup moins prévisible, rivalisant tantôt avec les meilleurs, tantôt avec les pires. Elle donne ses meilleurs résultats lorsque le coût de création d'un anneau domine celui de raccord d'un nœud (*i.e.* $r >> l$), et cela d'autant plus que la capacité C est petite.

Avec des écarts de 15% à 25% par rapport à la meilleure solution, les algorithmes adaptés de FFD sont juste corrects lorsque le coût de création d'un anneau domine celui de raccord d'un nœud (*i.e.* $r >> l$). Quand le

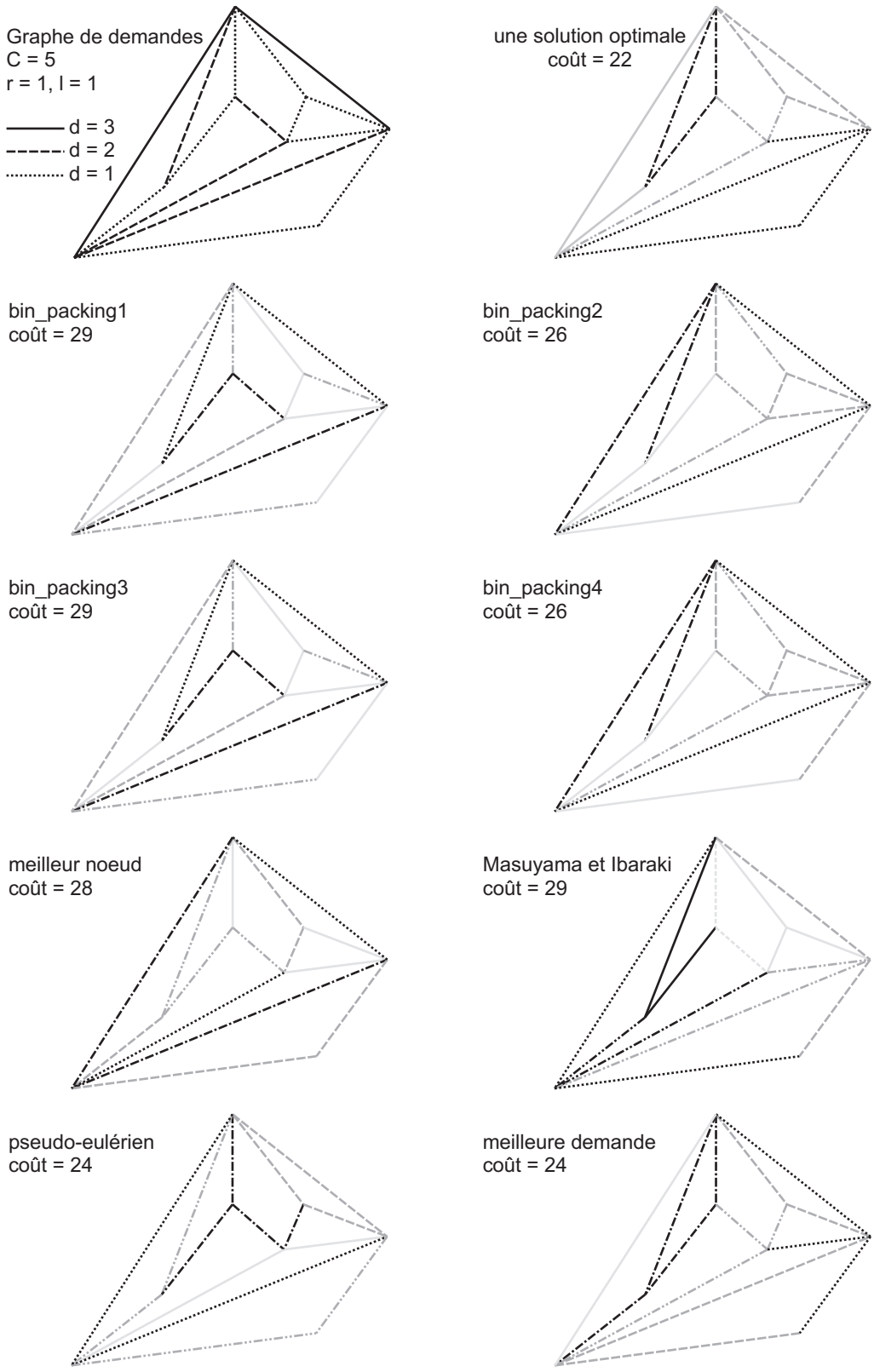

FIG. 4.1 – Solutions renvoyées par les différents algorithmes gloutons pour un petit exemple.

rapport entre ces coûts n'est plus à l'avantage de r, les solutions données sont généralement médiocres avec des écarts de l'ordre de 75%. C'est généralement pour de petites capacité C que les moins mauvais résultats sont obtenus et `bin_packing1` et `bin_packing3` ont les mêmes performances, qui sont moins bonnes que celles, à peu près identiques, de `bin_packing2` et `bin_packing4`.

Nous avons aussi testé les heuristiques sur des instances pour lesquelles nous connaissons la valeur de l'optimum, tels que des grilles ou des $K_{n,p}$ pour $C = 3$, des $K_{2,p}$ pour C pair, des étoiles. Dans ces cas-là les meilleures heuristiques trouvent presque systématiquement une solution optimale.

Chapitre 5

Une méthode de recherche tabou

Les algorithmes gloutons donnent des solutions pour ADR qui, bien que souvent optimales sur des cas simples, peuvent être nettement améliorées sur de grandes instances. C'est pour cela que nous envisageons maintenant d'utiliser une méta-heuristique, la recherche tabou, afin de reduire les coûts des solutions déjà obtenues.

Les premières idées sur la recherche tabou furent introduites par Hansen [14] et Glover [9]. C'est à ce dernier qu'on doit la forme actuelle de cette méthode qu'il a développée conjointement avec Laguna [10, 11].

Une recherche de minimum par la méthode tabou se base sur 2 grands principes. Tout d'abord on autorise une détérioration de la solution courante ; cela permet de sortir des puits que forment les minimums locaux. Ensuite on utilise une "mémoire" des solutions déjà explorées afin de ne pas y retourner systématiquement. Cette mémoire consiste en un certain nombre de solutions, dites *tabou*, qu'on interdit pendant un certain temps. La volonté derrière ces deux principes est de forcer une recherche dans des régions de l'espace des solutions qui n'ont pas encore été explorées.

À partir de ces idées de base de nombreuses variations peuvent être prises en compte mais le schémas de fonctionnement d'une recherche tabou est toujours sensiblement le même. Typiquement, on a une solution courante S ayant un certain voisinage $v(S)$. La nouvelle solution courante est alors, tout simplement, le meilleur élément de $v(S)$ qui n'est pas tabou. Cet élément devient alors tabou pour un certain nombre d'itérations. Ce nombre d'itérations est un des paramètres primordiaux d'une recherche tabou.

Généralement on passe d'une solution à un de ses voisins par une opération simple : un *mouvement* (par exemple : échanger les arêtes e_1 et e_2), et alors c'est le mouvement inverse que l'on interdit. Une telle gestion des solutions tabou peut interdire des solutions jamais explorées, ce qu'on compense par le phénomène d'*aspiration* : on s'autorise à choisir une solution

tabou si elle est meilleure que la meilleure trouvée jusque là.

Du fait de sa mémoire limitée, une recherche tabou peut boucler ou ne jamais sortir de certaines zones. Il existe différentes façons de faire face à ce problème. La plus simple consiste à forcer une *diversification* après un certain nombre d'étapes infructueuses voire de détérioration. Cette diversification consiste à choisir une solution hors du voisinage de la solution courante.

5.1 Description de notre méthode

5.1.1 Voisinage d'une solution

Le voisinage d'une solution est défini de manière simple par l'ensemble des solutions réalisables atteignables par un mouvement unique, ou *Move*, définit comme l'échange d'une arête d'un anneau contre une (ou zéro) arête d'un autre anneau.

Le voisinage ainsi défini a pour qualité première sa simplicité et sa relativement petite taille. De ce fait l'ensemble des voisins d'une solution peut être rapidement calculé, ce qui est essentiel pour une application pratique : pour m demandes, une capacité C et une liste de n_t tabous, le temps de calcul du voisinage d'une solution ayant R anneaux est de l'ordre de $\mathcal{O}(m.C.n_t.R)$, avec une valeur de R généralement de l'ordre de $\frac{m}{C}$ et une faible constante multiplicative.

Un autre point fort de ce voisinage est qu'à partir d'une solution quelconque on peut atteindre par une suite de *Move*s n'importe quelle solution n'ayant pas plus d'anneaux. Aussi, même si nous nous interdisons certaines solutions, nous gardons les plus prometteuses (il faut cependant mettre deux bémols à cela : tout d'abord, la propriété 2.6 page 19 montre qu'il existe des cas où minimiser le nombre d'anneaux empêche de trouver une solution optimale ; ensuite, il existe des exemples pour lesquels passer d'une solution donnée à une solution meilleure nécessite autant de *Move*s que ce qu'il y a de demandes).

5.1.2 Listes des tabous et aspiration

Nous avons réalisé trois gestions différentes de la liste des tabous. Supposons ainsi que nous effectuons le mouvement $Move(e_1 \in R_1, e_2 \in R_2)$, c'est-à-dire que nous enlevons e_1 de R_1 pour le mettre dans R_2 et que nous enlevons e_2 de R_2 pour le mettre dans R_1 (e_2 peut être éventuellement nul, et alors e_1 est échangé pour rien). Comme l'illustre la figure 5.1, la première méthode interdit de bouger e_1 ou e_2 ; la deuxième méthode interdit de re-échanger e_1 et e_2 ; la troisième méthode interdit de remettre e_1 and R_1 ou e_2 dans R_2. Dans tous les cas, les mouvements sont interdits pendant un nombre d'itérations constant donné en paramètre à notre programme.

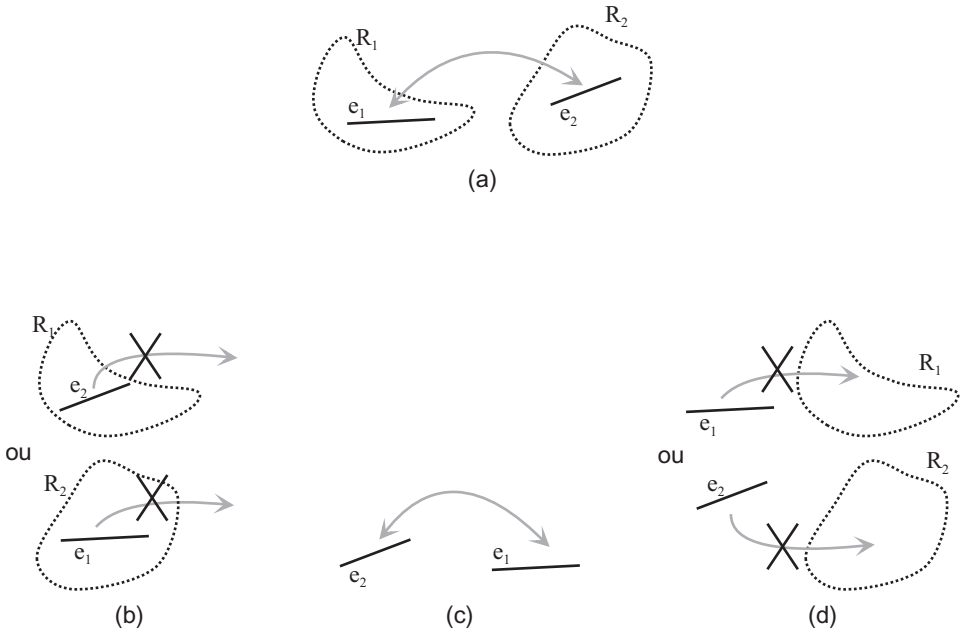

FIG. 5.1 – Les trois gestions différentes des éléments tabous : pour (a) un mouvement on interdit (b) de bouger l'une ou l'autre des arêtes ; (c) de re-échanger les arêtes ; (d) de remettre l'une ou l'autre des arêtes dans son anneau d'origine.

Nos gestions des solutions tabou interdisant plus que les solutions explorées, nous autorisons le phénomène d'aspiration, c'est-à-dire que si un voisin est tabou mais est meilleur que la meilleure solution trouvée jusque là, nous le prenons quand même comme solution courante. En pratique notre programme permet deux manières de réaliser cette aspiration : soit on sélectionne la première qui se présente, soit on explore tout le voisinage pour choisir celle qui apporte le meilleur gain.

5.1.3 Diversification

Une bonne diversification, pour bien relancer une recherche, est un point essentiel d'une recherche tabou mais assez délicat. Notre diversification se passe en deux temps : une première étape coupe chaque anneau R_i en deux anneaux R_i^1 et R_i^2, une arête $e_j \in R_i$ ayant la probabilité $\frac{1}{2}$ d'être dans chacun. Une deuxième étape concatène gloutonnement les anneaux entre eux, de manière à diminuer au mieux le nombre de raccords et d'anneaux[1].

Comme nous l'avons vu précédemment, le défaut majeur de notre définition du voisinage d'une solution et qu'elle ne permet pas à notre recherche tabou d'augmenter le nombre d'anneaux, ce qui peut être très handicapant.

[1] L'algorithme utilisé est décrit dans l'annexe D.

Notre diversification palie à ceci et se pose donc comme un bon complément. En effet, l'algorithme de concaténation d'anneaux utilisé n'est pas optimal[2] et en moyenne on peut s'attendre à une légère augmentation du nombre d'anneaux dans la solution après une diversification.

5.1.4 Solution initiale

La solution initiale donnée à une recherche tabou influence grandement toute la recherche mais malheureusement de manière presque imprévisible : certaines solutions initiales très mauvaises peuvent être facilement et grandement améliorées pour aboutir à de meilleures solutions finales qu'à partir de bonnes solutions initiales.

Pour notre recherche tabou, nous pouvons choisir l'algorithme d'initialisation parmi les algorithmes gloutons décrits au chapitre 4. Nous avons aussi rajouté un paramètre, dont l'idée nous vient de Fortz, Soriano, Wynants[6], qui précise la place à laisser vacante dans un anneau lors du calcul de la solution initiale. Bien entendu, cela détériore la solution initiale mais donne plus de liberté à la recherche tabou ce qui, au final, peut s'avérer fructueux.

5.1.5 Les différents paramètres

Une option de compilation permet de choisir, lors de la création du programme exécutable, parmi les 3 implantations des listes des tabous. Indépendamment de ce choix plusieurs autres paramètres sont réglables à l'exécution, ce qui permet de s'adapter à l'instance à traiter :

- la longueur de la liste des tabous ;
- le nombre maximum d'itérations avant l'arrêt de la recherche ;
- le nombre maximum d'itérations avec une dégradation de la solution courante avant une diversification ;
- le nombre maximum d'itérations sans améliorer la meilleure solution avant une diversification ;
- l'algorithme glouton d'initialisation utilisé parmi ceux décrits au chapitre 4). Pour une heuristique de type Bin-Packing, on peut aussi préciser une limitation du remplissage d'un anneau ;
- un commutateur permet de changer de mode d'aspiration : "dès qu'une meilleure solution est trouvée, on la choisit" ou bien "on explore tout puis on prend la meilleure".

5.2 Résultats expérimentaux

Nos expérimentations avaient un double but : elles devaient permettre d'évaluer les performances que l'on peut attendre de notre recherche tabou et

[2] Ce problème est de toute façon $\mathcal{NP}$-complet puisqu'en négligeant les raccords nous obtenons exactement un Bin-Packing.

aussi de cerner au mieux son comportement, avec pour finalité d'être capable de choisir, pour une instance donnée, un jeu de paramètres qui assure, sinon le meilleur, du moins un bon résultat.

5.2.1 Comportement

Pour à peu près n'importe quelle instance et n'importe quel jeu de paramètres, c'est dans les premières itérations que l'essentiel de l'amélioration est fait, comme l'illustre la figure 5.2. Une heuristique de type "2-OPT"[3]

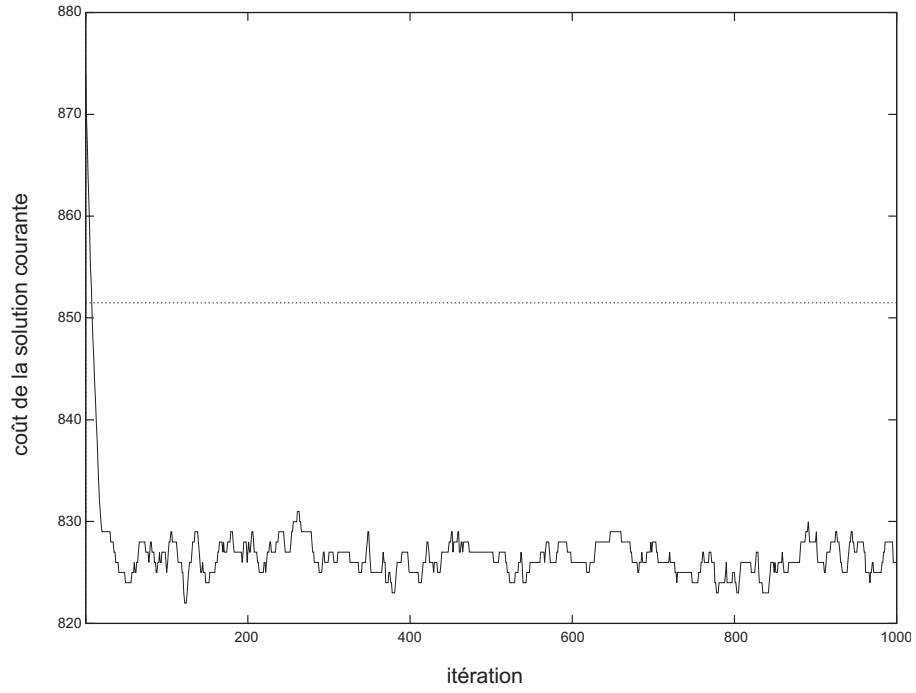

FIG. 5.2 – Un exemple d'évolution du coût de la solution courante lors d'une recherche tabou.

donnerait cependant des résultats bien moins bons car, avant même les premières détériorations de la valeur de la solution courante, il y a des paliers qu'une telle heuristique ne franchirait pas.

Après cette première et relativement brève phase, la recherche oscille entre améliorations et détériorations de la solution courante, en trouvant parfois une nouvelle meilleure solution. Toutefois, le gain obtenu dans cette deuxième phase est nettement inférieur à celui obtenu dans la première.

Un tel comportement, classique pour une recherche tabou, est à peu près

[3] "2-OPT" : cette méthode de recherche d'extrémum local consiste à faire des échanges d'arêtes apportant un gain strictement positif, et cela tant que de tels échanges existent.

général, mais cela n'empêche pas les résultats d'être très différents, pour une même instance, selon les paramètres utilisés.

5.2.2 Influence des différents paramètres

Gestion de la liste des tabous

La qualité de la meilleure solution trouvée varie selon la gestion de la liste des tabous employée. Ainsi, à peu près systématiquement, la gestion 1 (interdiction de bouger l'une ou l'autre des arêtes) aboutit à de significativement moins bonnes solutions et cela quels que soient les autre réglages. Les gestions 2 (interdiction de re-échanger les 2 arêtes) et 3 (interdiction de remettre les arêtes dans leur anneau d'origine) fournissent des résultats semblables, avec un léger avantage pour 3.

En ce qui concerne la longueur de la liste des tabous, une valeur de l'ordre de $3\sqrt{m}$, où m est le nombre de demandes, nous a toujours donné des résultats parmi les meilleurs. Une nette amélioration est cependant envisageable en n'imposant pas une longueur constante à cette liste, mais en la faisant varier en fonction du numéro de l'itération dans la recherche[4], voire en infligeant des pénalités aux arêtes souvent bougées, *i.e.* en imposant à celles-ci un nombre d'itérations tabou croissant avec leur nombre de mouvements.

Diversification

La diversification est assez décevante. En effet, il est très rare qu'elle amène un gain et, pire, elle dégrade presque toujours la meilleure solution trouvée.

Toutefois, dans le cas de demandes non uniformes au moins, on note que forcer une diversification au début de la recherche (après 10 ou 50 itérations, avant même d'avoir atteint un premier minimum local), peut s'avérer profitable : les solutions ainsi obtenues sont assez souvent meilleures et rarement nettement moins bonnes que sans diversification.

Solution initiale

La recherche tabou ressert les écarts obtenus entre les initialisations par différentes méthodes gloutonnes. Dans d'assez nombreux cas, même, la meilleure valeur trouvée est la même pour toutes les initialisations.

Cependant, en dehors de ces cas, les meilleures initialisations engendrent souvent les meilleures solutions ou en sont proches. Il faut toutefois signaler l'existence d'un nombre non négligeable de cas, principalement pour des

[4]Quelques tests ont été réalisés dans ce sens ; des résultats positifs s'obtiennent assez facilement, mais une étude plus poussée est nécessaire pour déterminer les fonctions véritablement efficaces.

demandes unitaires, où les meilleures initialisations aboutissent aux pires solutions, et vice versa. Malgré cela, utiliser une bonne initialisation permet d'avoir des garanties sur la qualité de la solution finale. En effet, on constate sur plusieurs exemples que la meilleure solution obtenue par la recherche tabou à partir d'une mauvaise initialisation peut être moins bonne qu'une bonne initialisation avant même la recherche tabou.

Autres paramètres

D'autres paramètres ont été brièvement testés. Ainsi, entre les deux méthodes d'aspiration : explorer tout le voisinage puis choisir son meilleur élément ou bien effectuer une aspiration dès qu'un voisin le permettant est trouvé, la deuxième façon de faire, un peu moins coûteuse en temps de calcul, permet d'obtenir d'un peu meilleurs résultats si les demandes sont unitaires, mais un peu moins bons dans le cas contraire.

En ce qui concerne la possibilité de laisser une place vide dans les anneaux au moment de la méthode gloutonne, cela aboutit parfois, pour de faibles valeurs, à un léger gain. Cependant, la perte de qualité devient très vite irrécupérable pour la méthode tabou et il semble finalement préférable de ne laisser aucun espace libre.

Quant au nombre d'itérations de la recherche, il est évident que, si on peut se le permettre, il faut lui donner une grande valeur. Cependant, comme le comportement d'une recherche tabou l'annonce, c'est essentiellement au début de la recherche que l'amélioration est obtenue. Aussi, pour un graphe de m demandes, il ne faut plus s'attendre à de véritable gain passé quelques m itérations (il semble que pour un graphe assez régulier quelques m suffisent, mais il faut aller jusqu'à quelques $10m$ pour des graphes plus chaotiques).

5.2.3 Conclusions

La méthode de recherche tabou que nous avons implantée donne des résultats suffisamment satisfaisants, si les paramètres sont réglés de manière cohérente, pour être utilisée en pratique : non seulement elle est applicable sur des instances assez grandes mais elle permet aussi des gains conséquents par rapport aux meilleurs algorithmes gloutons[5]. Il faut aussi signaler que sur les cas dont nous connaissons la valeur optimale ou une borne assez proche, les résultats sont bons, la solution à la fin de la recherche tabou n'étant jamais à plus de quelques pourcents de la borne.

Toutefois, de nombreuses améliorations sont envisageables qui peuvent engendrer des gains importants. Comme nous l'avons déjà signalé, c'est tout

[5]Sur des instances simples, sur lesquelles les algorithmes gloutons sont performants, on obtient des gains de 3-4% ; sur des instances plus complexes, qui mettent en défaut les gloutons, les gains peuvent être de plus de 20%.

d'abord au niveau de la gestion de la liste des tabous que des modifications sont possibles, notamment une gestion dynamique de sa longueur, voire une gestion individualisée, un mouvement étant d'autant plus longtemps tabou qu'il est souvent effectué. C'est ensuite au niveau de la diversification que du travail peut être fait, par exemple en y intégrant une "mémoire" des configurations intéressantes afin de diriger la recherche vers des zones prometteuses.

Conclusions et perspectives

La littérature portant sur notre problème est généralement consacrée à l'un ou l'autre de ses extrêmes, c'est-à-dire à un coût de création d'un anneau nul ($r = 0$) ou à un coût de raccord d'un nœud négligeable ($l = 0$). En nous positionnant dans l'entre deux, nous avons à la fois synthétisés et développés ces résultats.

Le problème général ADR est ainsi formellement posé et sa forte $\mathcal{NP}$-complétude établie. Étant donnés ces premiers résultats, nous nous sommes efforcés d'établir différentes propriétés afin de permettre une résolution au mieux. De là l'exploration du cas des solutions arborescentes qui, bien que rarement optimales, permettent d'aboutir à plusieurs algorithmes avec des garanties de performances.

Cette étude théorique a été suivie d'une méthode de résolution pratique du problème basée sur un algorithme de recherche tabou qui s'avère effectivement utilisable en pratique, mais qui peut être encore grandement améliorée. En outre, plusieurs algorithmes gloutons de résolution ont été proposés.

Parallèlement à cette étude du problème général, des cas particuliers, dont plusieurs correspondent à des exemples concrets de réseaux, ont été résolus de manière exacte et polynomiale[1]. Il reste cependant au moins un cas particulier intéressant en pratique, celui des graphes complets K_n, que nous n'avons pas résolu et qui mérite de l'attention.

D'un point de vue théorique, un éclairage nouveau peut être apporté au problème en privilégiant les nœuds du graphe des demandes à ses arêtes[2]. Il serait aussi intéressant de développer l'étude des cas où l'on peut affirmer qu'un anneau donné appartient à une solution optimale, ou au moins borner la valeur des solutions auxquelles il appartient. Ceci permettrait une meilleur orientation des algorithmes de résolution, notamment dans une optique de résolution exacte par une méthode de type "Branch & Bound".

Pour la résolution pratique d'autres approches sont possibles, par exemple une méthode de génération de colonnes pour laquelle nous disposons déjà d'un modèle. L'étude d'une autre technique de résolution aurait de plus l'avantage de pouvoir mieux qualifier les résultats obtenus par notre

[1] Ces cas particuliers de résolutions sont donnés en annexe A.

[2] L'annexe H propose un approche où une solution n'est plus une partition des arêtes, mais une valuation des sommets.

recherche tabou.

Plusieurs développements de notre problème sont envisageables. L'un des principaux serait de considérer le coût de création r comme une fonction de l'anneau, l'hypothèse de son indépendance étant généralement irréaliste. Dans le même ordre d'idée, le coût de raccord l peut lui aussi être considéré comme une fonction de la capacité réelle transitant le long d'un anneau, si on assimile un raccord à l'ADM qu'il nécessite.

Le problème ADR peut aussi être étendu à d'autres standards, comme les réseaux SHR bidirectionnels pour lequel il faut aussi traiter le problème de répartition des signaux sur l'une ou l'autre fibre.

Annexe A

Exemples de résolution d'ADR

Cette annexe présente quelques cas pour lesquels nous arrivons non seulement à déterminer polynomialement la valeur optimale de ADR, mais aussi à trouver des configurations qui réalisent une telle valeur. D'une manière générale, dans les exemples présentés, l'idée est qu'il existe un type d'anneaux plus intéressant que tous les autres et qu'il est possible de faire des solutions avec seulement ce type d'anneau-là, qui se trouvent ainsi être optimales (des démonstrations plus précises seront faites au cas par cas).

Mis à part le cas d'une capacité $C = 2$ que nous traitons dans toute sa généralité, nous ne nous intéressons ici qu'à des cas de demandes unitaires. En effet, rappelons que pour $l = 0$ le problème est indépendant de la topologie du graphe et se réduit au Bin-Packing. De ce fait, pour des demandes non uniformes, il est fortement $\mathcal{NP}$-complet, et cela indépendamment du graphe. Il est donc peu probable de trouver un algorithme efficace donnant la valeur de l'optimum.

A.1 Les demandes sont quelconques, $C = 2$

Brauner, Crama et Wynants [2], en se basant sur les travaux de Masuyama et Ibaraki portant sur la partition d'un graphe quelconque en 2-chaînes [17], ont tracé les grandes lignes de la résolution polynomiale d'ADR pour $C = 2$. Ce qui suit présente en 3 étapes leurs résultats[1] complétés : tout d'abord, nous nous occupons de graphes connexes avec des demandes unitaires, puis nous généralisons à des graphes non nécessairement connexes, pour finir avec le résultat général pour des graphes de demandes quelconques.

[1] On trouve des résultats similaires dans [12].

Traitons tout d'abord le cas, un peu simplifié, lorsque toutes les demandes sont de 1. Nous avons le résultat suivant :

Lemme A.1 *Pour $C = 2$, le problème ADR peut être résolu en temps polynomial (en fait : linéaire par rapport au nombre de demandes) pour tout graphe connexe G de demandes unitaires.*

preuve :

Masuyama et Ibaraki ont montré [17] qu'on peut trouver en temps polynomial une couverture par des 2-chaînes de tout graphe connexe ayant un nombre pair d'arêtes. En d'autres termes : on peut trouver en temps polynomial une solution S, réalisable, n'ayant que des 2-chaînes comme anneaux (avec éventuellement une arête seule s'il y a un nombre impair de demandes). Une telle solution minimise le nombre de composantes connexes et donc, d'après le théorème 2.2 page 17, elle minimise le nombre de raccords. Comme elle minimise aussi le nombre d'anneaux, elle est optimale pour ADR. •

Voici maintenant, basé là encore sur les mêmes travaux de Masuyama et Ibaraki, un algorithme permettant cette partition d'un graphe G connexe en 2-chaînes :

Algorithme A.1 de Masuyama et Ibaraki

1. `choisir une racine` r `dans` G`.`
2. `numéroter toutes les arêtes de` G `selon un parcours en largeur. On obtient ainsi un ‘‘arbre couvrant ’’ de toutes les arêtes de` G`.`
3. `tant qu'il reste des arêtes, choisir celle de plus grand numéro et la regrouper avec un de ses frères si elle en a, son père sinon.`

La figure A.1 illustre l'exécution de cet algorithme. Sa validité vient du parcours en largeur : l'arête de plus grand numéro est aussi la plus profonde. De ce fait, la choisir ne déconnecte pas le graphe ; de même pour son éventuel frère puisqu'il ne peut pas avoir de successeur (qui serait, sinon, de numéro plus grand que l'arête sélectionnée). De plus, cet algorithme est clairement linéaire par rapport au nombre d'arêtes du graphe.

Nous généralisons maintenant le lemme A.1 aux graphes non nécessairement connexes :

Lemme A.2 *Pour $C = 2$, le problème ADR peut être résolu en temps polynomial (en fait : linéaire par rapport au nombre de demandes) pour tout graphe G de demandes unitaires.*

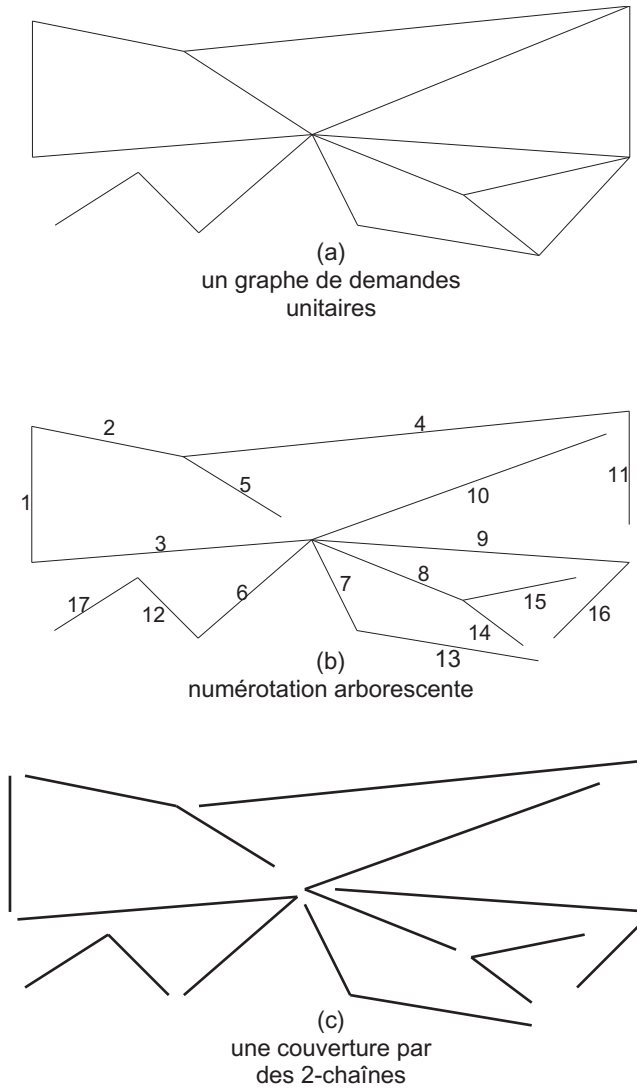

FIG. A.1 – Un exemple d'exécution de l'algorithme de Masuyama et Ibaraki.

preuve :

Si une solution a plus d'une arête solitaire, nous pouvons les regrouper par paires pour former des anneaux de deux arêtes et ainsi minimiser le nombre d'anneaux. Prenons donc une solution S ayant une arête solitaire si G a un nombre impair d'arêtes et aucune sinon. S minimisera le nombre de raccords — et ainsi sera optimale pour ADRL mais aussi ADR — si elle minimise le nombre de composantes connexes.

Or en appliquant l'algorithme A.1 à chaque composante connexe $G_i, i = 1 \dots k$ de G, nous obtenons, en temps linéaire[2], un ensemble de 2-chaînes ainsi que des arêtes solitaires $e_1 \dots e_p$ $(p \leq k)$ que nous regroupons arbitrairement par paires. Cela donne une solution S optimale. En effet, l'ensemble des 2-chaînes est maximal dans la mesure où la couverture est maximale sur chaque G_i et qu'il ne peut pas y avoir de 2-chaînes adjacentes à 2 G_i distincts. De même, chaque arête e_j appartenant à une composante connexe différente, il ne peut s'en trouver 2 adjacentes, et donc quel que soit l'arrangement fait, le nombre de composantes connexes sera le même. •

Voici maintenant le résultat général :

Théorème A.1 *Pour $C = 2$, le problème ADR peut être résolu en temps polynomial (en fait : linéaire par rapport au nombre de demandes).*

preuve :

En plus des demandes de 1, il faut maintenant envisager des demandes de 2. Cependant, ces demandes ne laissent aucun choix quant à leur affectation dans des anneaux : chaque demande de 2 constitue, seule, un anneau. Cette affectation est polynomiale (linéaire, même, en le nombre d'arêtes) et transforme le graphe G en un graphe G' n'ayant que des demandes unitaires, et pour lequel on peut donc appliquer le lemme A.2. Il s'en suit que ADR est soluble polynomialement pour $C = 2$. •

A.2 Le graphe est une chaîne ou un cycle

La résolution de ADR pour une chaîne de demandes unitaires est triviale, comme l'illustre la figure A.2 et l'énonce la propriété suivante :

Propriété A.1 *Dans le cas où le graphe des demandes G est une chaîne de n demandes unitaires, ADR se résoud polynomialement et la valeur de l'optimum est :*

$$\mathcal{OPT} = \left\lceil \frac{n}{C} \right\rceil r + \left(\left\lceil \frac{n}{C} \right\rceil + n \right) l$$

[2]Déterminer les composantes connexes d'un graphe est aussi linéaire en le nombre d'arêtes ; en fait, on peut déterminer ces composantes lors de la phase de numérotation de l'algorithme A.1.

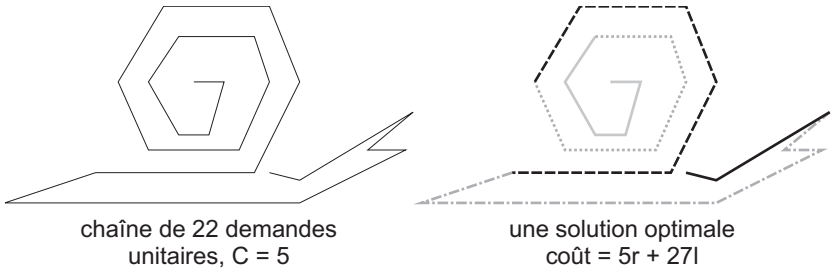

FIG. A.2 – Un exemple de résolution de ADR sur une chaîne de demandes unitaires.

preuve :

Soit la solution S formée en coupant la chaîne toutes les C arêtes. Cela se fait en temps linéaire par rapport au nombre d'arêtes. Chaque sous-chaîne de S constitue un anneau valide et il y en a exactement $\lceil \frac{n}{C} \rceil$, *i.e.* le nombre minimum d'anneaux (*cf* propriété 2.1 page 15). De plus chaque anneau est connexe et donc S minimise le nombre de composantes connexes pour une solution réalisable. Le théorème 2.2 page 17 permet donc d'affirmer que S minimise aussi le nombre de raccords. Il s'en suit que S est optimale, puisqu'elle minimise les deux paramètres de coût. Le lemme 2.3 page 17 nous donne son coût, en nombre de raccords : $(n + \lceil \frac{n}{C} \rceil)l$, ce qui fait un coût global de la solution de $\lceil \frac{n}{C} \rceil r + \left(\lceil \frac{n}{C} \rceil + n \right) l$. •

Remarque : Ce qui précède est aussi valable si G est un cycle, mis à part si $n \leq C$: dans ce cas, la solution optimale est constituée d'un unique anneau et son coût est de $r + n.l$.

A.3 Le graphe est une étoile $K_{1,p}$

Dans le cas d'une étoile (graphe biparti-complet $K_{1,p}$) nous pouvons remarquer que toutes les arêtes ont un rôle identique et qu'elles sont toutes adjacentes en le même nœud central. Le coût de la solution ne dépend ainsi que du nombre d'anneaux R auxquels appartient ce nœud central : le coût d'une solution est de $rR + l(p + R)$. Il faut donc minimiser R, ce qui fait que résoudre ADR sur une étoile revient exactement à un BIN-PACKING sur les demandes.

De ce fait si les demandes ne sont pas uniformes le problème est $\mathcal{NP}$-complet. Si les demandes sont uniformes le problème devient trivial : pour des demandes unitaires n'importe quelle partition des arêtes en $\lceil \frac{p}{C} \rceil$ anneaux donne une solution optimale puisque c'est le nombre minimum d'anneaux possible (*cf* propriété 2.1), et il est trivial de construire une telle solution (*cf* figure A.3), d'où :

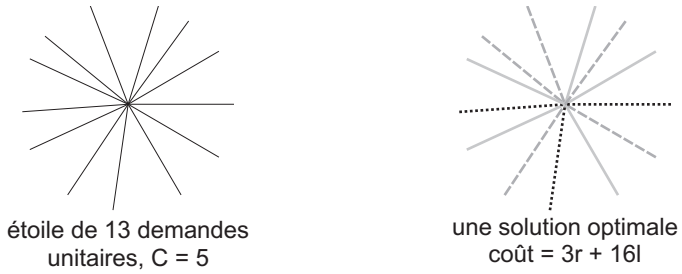

FIG. A.3 – Un exemple de résolution de ADR sur une étoile de demandes unitaires.

Propriété A.2 *Dans le cas où le graphe des demandes G est une étoile $K_{1,p}$ de demandes unitaires, ADR se résoud polynomialement et la valeur de l'optimum est :*

$$\mathcal{OPT} = \left\lceil \frac{p}{C} \right\rceil r + \left(\left\lceil \frac{p}{C} \right\rceil + p \right) l$$

A.4 Le graphe est une grille, $C = 3$

Lorsque le graphe G de demandes unitaires est biparti — ce qui est le cas d'une grille — il ne contient pas de cycles impairs, et donc pas de cycle de longueur inférieure à 3. Le théorème 2.2 page 17 permet donc d'affirmer qu'une solution optimale de ADRL (*i.e.* $r = 0$) sur un tel graphe est une forêt qui minimise le nombre de composantes connexes, qui sont ici des arbres d'au plus 3 arêtes. De ce fait une couverture du graphe avec seulement des 3-arbres (plus éventuellement une 2-chaîne ou une arête seule, selon le nombre total d'arêtes) est une solution optimale de ADRL sur G pour $C = 3$. De plus une telle solution a $\left\lceil \frac{|E|}{C} \right\rceil$ anneaux et ainsi elle minimise aussi le nombre d'anneaux, d'après la propriété 2.1 page 15. Une telle couverture est donc optimale pour ADR, quels que soient r et l.

Pour une grille une telle couverture est facile à construire. Il existe, par exemple, l'algorithme suivant (linéaire en le nombre d'arêtes) pour une grille de taille $n \times p$:

Algorithme A.2 Couverture d'une grille par des 3-arbres

1. numéroter, dans l'ordre d'adjacence, les lignes de 1 à n et les colonnes de 1 à p ;
   $i = 1, j = 1$
2. pour le nœud courant (i, j) prendre, sous réserve d'existence, les arêtes dans l'ordre : (a) $n_{i-1,j}n_{i,j}$, (b) $n_{i,j}n_{i,j+\varepsilon}$ avec $\varepsilon = 1$ si i est impair, -1 sinon ;
   trois arêtes prises consécutivement forment un anneau.

3. `si` $1 \leq j + \varepsilon \leq p$ `alors` $j = j + \varepsilon$ `sinon` $i = i + 1$;
`si` $i < n$ `retourner en (2) sinon STOP.`

Cet algorithme consiste, en fait, simplement à parcourir la grille le long de ses lignes, alternativement de gauche à droite puis de droite à gauche, selon la parité du numéro de ligne, et d'empaqueter au passage les arêtes auprès desquelles on passe (*cf* figure A.4). De par les déplacements, 2 arêtes choisies

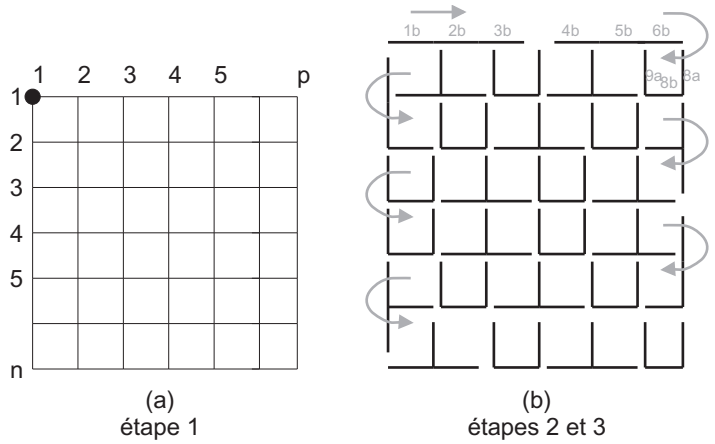

FIG. A.4 – Un exemple d'exécution de l'algorithme de résolution optimale d'ADR sur une grille, $C = 3$.

consécutivement sont adjacentes et donc la solution construite n'a bien que des 3-arbres (le dernier pouvant être incomplet).

De plus, nous déduisons de ce qui précède la valeur de l'optimum :

Propriété A.3 *Pour une capacité $C = 3$ et une grille G de taille $n \times p$ de demandes unitaires, ADR se résoud en temps linéaire et la valeur optimale d'ADR est :*

$$\mathcal{OPT} = \begin{cases} \frac{x}{3}(r + 4l) & \text{si } x \mod 3 = 0 \\ \left\lceil \frac{x}{3} \right\rceil r + \left(4 \left\lfloor \frac{x}{3} \right\rfloor + (x \mod 3) + 1\right) l & \text{sinon} \end{cases}$$

avec : $x = 2np - n - p$ (nombre d'arêtes pour une grille).

A.5 Le graphe est un biparti-complet $K_{n,p}$, $C = 3$

Les principes évoqués dans la section A.4 sont valables pour un graphe biparti-complet $K_{n,p}$. Il suffit donc de donner un algorithme qui décompose un graphe $K_{n,p}$ en 3-arbres (à un arbre près) pour avoir une résolution optimale de ADR pour $C = 3$ sur $K_{n,p}$. Un tel algorithme existe : en notant $\{n_1, \ldots, n_n\}$ et $\{p_1, \ldots, p_p\}$ les sommets de $K_{n,p}$, R_k les anneaux de la solution S renvoyée, et (x, y) l'arête entre les sommets x et y, nous avons par exemple :

Algorithme A.3 Couverture de $K_{n,p}$ par des 3-arbres

1. $k = 1$
   $\beta = \lfloor \frac{n}{3} \rfloor$
   pour $i \in \{1, \ldots, \beta\}$ **faire :**
     pour $j \in \{1, \ldots, p\}$ **faire :**
       $R_k = \{(n_i, p_j), (n_{\beta+i}, p_j), (n_{2\beta+i}, p_j)\}$
       $k = k + 1$
2. **si** $n = 1 \mod 3$ **alors :**
     pour $j \in \{1, \ldots, p\}$ **faire :**
       $R_k = R_k \cup \{(n_n, p_j)\}$
       si $|R_k| = 3$ **alors :** $k = k + 1$
3. **sinon** $n = 2 \mod 3$ **alors :**
     pour j de 1 à p **faire :**
       $R_k = R_k \cup \{(n_{n-1}, p_j)\}$
       si $|R_k| = 3$ **alors :** $k = k + 1$
     pour j de p à 1 **faire :**
       $R_k = R_k \cup \{(n_n, p_j)\}$
       si $|R_k| = 3$ **alors :** $k = k + 1$

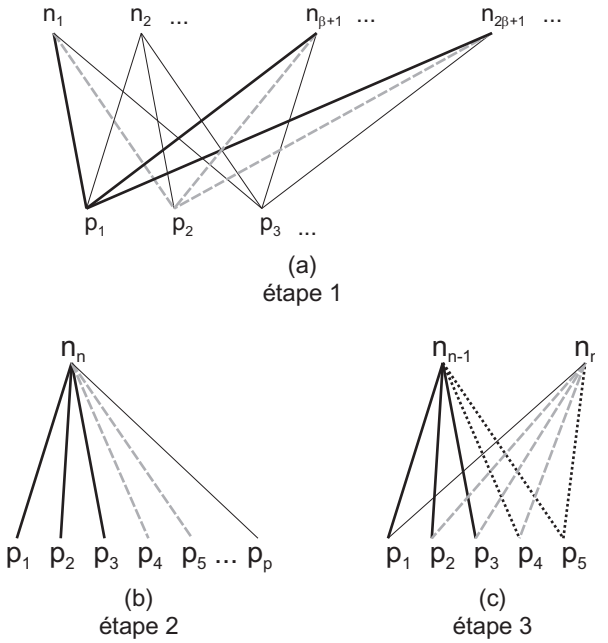

FIG. A.5 – Un exemple d'exécution de l'algorithme de résolution optimale d'ADR sur $K_{n,p}$, $C = 3$.

Cet algorithme se déroule comme suit (nous appelons 3-étoile de racine x un 3-arbre pour lequel les 3 arêtes sont adjacentes en x) :

L'étape (1) de l'algorithme (*cf* figure A.5(a)) construit autant de 3-étoiles de racine p_j qu'on peut en faire. Si n est un multiple de 3, alors cette étape permet de couvrir tout le graphe et l'algorithme peut se terminer.

Si n est congru à 1 modulo 3, il reste après l'étape (1) un nœud, n_n, relié à tous les p_j. L'étape (2) (*cf* figure A.5(b))construit alors autant de 3-étoiles de racine n_n qu'on peut en faire avec les arêtes restantes, la dernière étant éventuellement incomplète.

Si n est congru à 2 modulo 3, il reste après l'étape (1) deux nœuds, n_{n-1} et n_n, reliés à tous les p_j. L'étape (3) (*cf* figure A.5(c)) construit alors autant de 3-étoiles de racine n_{n-1} qu'on peut en faire, la dernière étant éventuellement incomplète. Ensuite, elle complète si nécessaire cette étoile avec des arêtes adjacentes à n_n (cela donne un 3-arbre) puis fini de rassembler les arêtes restantes en 3-étoiles de racine n_n. Le fait d'effectuer la deuxième boucle à l'envers permet d'assurer la connexité d'un R_k commencé dans la première boucle et fini dans la deuxième.

Cet algorithme construit une partition des arêtes de G uniquement composée de 3-arbres (à un près si le nombre d'arêtes ne le permet pas). Il donne donc une solution optimale de ADR pour $C = 3$. De plus, comme pour une grille, nous déduisons de ceci la valeur de l'optimum :

Propriété A.4 *Pour une capacité $C = 3$ et un graphe biparti-complet $K_{n,p}$ de demandes unitaires, ADR se résoud en temps linéaire et la valeur optimale d'ADR est :*

$$\mathcal{OPT} = \begin{cases} \frac{x}{3}(r+4l) & \text{si } x \mod 3 = 0 \\ \left\lceil \frac{x}{3} \right\rceil r + \left(4\left\lfloor \frac{x}{3} \right\rfloor + (x \mod 3) + 1\right) l & \text{sinon} \end{cases}$$

avec : $x = 2np$ (nombre d'arêtes de $K_{n,p}$).

A.6 Le graphe est un arbre, $C = 3$

Dans cette section, nous résolvons polynomialement le problème ADR pour un arbre de demandes unitaires et une capacité $C = 3$. Notre méthode se base sur un algorithme de décomposition optimale d'un arbre en 3-arbres décrit dans l'annexe B.1. Le théorème B.2 de cette même annexe nous indique aussi que cette méthode n'est pas extensible de manière exacte pour un graphe quelconque si $\mathcal{P} \neq \mathcal{NP}$.

Considérons un arbre de demandes unitaires et une capacité $C = 3$ (la figure A.6 donne un exemple d'instance avec une solution optimale calculée par notre algorithme).

Tout d'abord, si le coût de création d'un anneau $r = 0$ (cas ADRL), alors le nombre d'anneaux est indifférent et c'est le nombre de composantes connexes qui implique le coût de la solution, comme cela est traduit par le théorème 2.2 page 17. Il s'en suit directement le résultat suivant pour la résolution de ADRL :

FIG. A.6 – Un exemple de résolution de ADR sur un arbre de demandes unitaires, $C = 3$.

Lemme A.3 *Le problème ADRL (demandes unitaires et $r = 0$) pour un arbre et une capacité de $C = 3$ est soluble à l'optimum en temps polynomial (linéaire en le nombre de demandes).*

preuve :

Toute solution de ADRL sur un arbre T est une partition des arêtes de T en arbres. D'après le théorème 2.2 une solution sera optimale si et seulement si elle minimise le nombre de composantes connexes. Le corollaire B.1.1 nous assure que l'algorithme B.1, linéaire, permet une décomposition d'un arbre en un nombre minimum de composantes connexes d'au plus 3 arêtes. En d'autres termes, il résoud optimalement ADRL sur T pour $C = 3$. •

Maintenant, pour résoudre en tenant compte d'un coût de création d'un anneau $r > 0$, il suffit d'étudier un peu plus en détails la partition d'un arbre qu'engendre l'algorithme B.1. Ainsi, nous constatons qu'à toutes les étapes ne sont créés que des 3-arbres, excepté lors de l'application du lemme B.3 où des arêtes sont laissées de côté et à la toute fin de l'algorithme, lorsqu'il reste moins de 3 arêtes et où donc une arête ou une 2-chaîne peut éventuellement être laissée de côté. Ainsi, en regroupant cette éventuelle 2-chaîne et ces arêtes en anneaux de 3 arêtes (ce qui se fait facilement, linéairement en le nombre d'arêtes), la solution obtenue minimise le nombre de raccords (elles est toujours optimale pour ADRL) et de plus, elle minimise le nombre d'anneaux. Elle est donc tout simplement optimale pour AUDR, ce qui se résume par le théorème suivant :

Théorème A.2 *Le problème AUDR pour un arbre et une capacité de $C = 3$ est soluble à l'optimum en temps polynomial (linéaire en le nombre de demandes).*

Corollaire A.2.1 *Soit T un arbre de demandes unitaires. Le nombre de raccords dans une solution optimale de ADR (ou ADRL) sur T et une capacité $C = 3$ est borné par :*

$$L \leq \frac{3}{2}|E| + \frac{1}{2}$$

preuve :

Nous venons de voir, dans la justification du théorème précédent, qu'il existe une solution optimale qui minimise le nombre d'anneaux et le nombre de raccords parmi toutes les solutions réalisables possibles. De ce fait, toute solution optimale a ce même nombre d'anneaux, et aussi ce même nombre de raccords L.

De plus nous avons vu que la solution (optimale) construite par l'algorithme B.1 avait une forme très particulière. Dans cette solution, chaque fois qu'une arête est laissée toute seule, un 3-arbre est aussi créé (application du lemme B.3), avec l'exception de la dernière itération qui, par manque d'arêtes, peut laisser une arête ou une 2-chaîne seule. De ce fait, à cette exception près, il y a au plus $\frac{|E|}{4}$ arêtes solitaires pour $\frac{3|E|}{4}$ 3-arbres, soit un coût moyen de $\left(\frac{1}{4} \times 2 + \frac{3}{4} \times \frac{4}{3}\right) = \frac{3}{2}$ raccords par arête. De là nous tirons une borne sur le nombre de raccords, en tenant compte de ce qui reste éventuellement à la dernière itération :

$$\begin{aligned} L &\leq \min\left\{3 + \frac{3}{2}(|E| - 2), 2 + \frac{3}{2}(|E| - 1), \frac{3}{2}|E|\right\} \\ &\leq \frac{3}{2}|E| + \frac{1}{2} \end{aligned}$$

•

A.7 Cas des graphes bipartis, $C = 3$

Dans les sections précédentes ont été présenté des exemples de résolutions de ADR pour $C = 3$ sur des graphes bipartis particuliers. Les principes évoqués (principalement à la section A.4) restent valables pour n'importe quel graphe biparti, mais il n'existe pas toujours de couverture par seulement des 3-arbres (à un arbre près), qui est une garantie d'optimalité. Une telle couverture n'existe pas, par exemple dans le cas des arbres, pour lequel la résolution exacte est plus compliquée (*cf* section A.6 page 55). Il y a aussi de nombreuses autres familles de graphes bipartis pour lesquels nous ne savons rien dire.

A.8 Le graphe est biparti-complet $K_{2,p}$, C pair

Prenons un graphe G biparti-complet $K_{2,p}$: chacun des $p + 2$ sommets du graphe G a un degré supérieur ou égal à 2. Nous pouvons donc déduire

de la propriété 2.4 page 16 que le nombre de raccords L pour une solution ayant R anneaux vérifie $L \geq p + 2 + 2(R-1) = p + 2R$. Or la propriété 2.1 page 15 nous donne une borne sur le nombre minimum d'anneaux pour une capacité C : $R \geq \left\lceil \frac{2p}{C} \right\rceil$. Nous en déduisons que le coût minimum d'une solution réalisable de ADR sur G pour une capacité C est de :

$$c(S) \quad \geq \quad \left\lceil \frac{2p}{C} \right\rceil r + \left(p + 2 \left\lceil \frac{2p}{C} \right\rceil \right) l \tag{A.1}$$

Plaçons nous maintenant dans le cas où la capacité C est paire. Notons $\{n_1, n_n\}$ et $\{p_1, p_2, \ldots, p_p\}$ les deux ensembles formant les sommets de $K_{2,p}$ et $libre(r)$ la place disponible dans l'anneau r. Considérons l'algorithme suivant (dont une exécution est illustrée par la figure A.7) qui consiste à ajouter la 2-chaîne passant par p_i à l'anneau courant s'il peut la contenir, à un nouvel anneau sinon :

Algorithme A.4 Résolution de ADR sur $K_{2,p}$, C pair

1. $k = 1$; $R_k = \emptyset$
2. **pour** $i = 1$ à p **faire :**
   si $libre(R_k) < 2$ **alors :** $k = k + 1$; $R_k = \emptyset$
   $R_k = R_k \cup \{e_{n_1 p_i}, e_{n_2 p_i}\}$

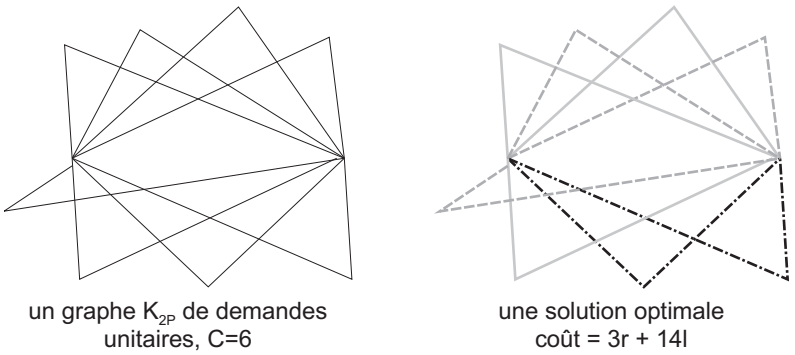

FIG. A.7 – Un exemple de résolution optimale de ADR sur un graphe $K_{2,p}$ de demandes unitaires, pour C pair.

Notons tout d'abord que cet algorithme construit une solution réalisable S de ADR sur G. Évaluons son coût. Puisque la capacité est paire, l'algorithme remplit complètement un anneau avant d'en créer un autre : le nombre d'anneaux à la fin est donc le minimum possible de $\left\lceil \frac{2p}{C} \right\rceil$ anneaux. De plus n_1 et n_2 sont raccordés à chaque anneau, mais un p_i n'est raccordé qu'à un seul. Le nombre de raccords total dans S est donc de $p + 2 \left\lceil \frac{2p}{C} \right\rceil$,

ce qui fait un coût total de $c(S) = \left\lceil \frac{2p}{C} \right\rceil r + \left(p + 2 \left\lceil \frac{2p}{C} \right\rceil \right) l$. Le coût de S est ainsi égal à la borne (A.1), ce qui prouve l'optimalité de S et permet de conclure :

Propriété A.5 *Pour un graphe de demandes unitaires $G = K_{2,p}$ et une capacité C paire, le problème ADR est soluble à l'optimum en temps linéaire et la valeur optimale est de :*

$$\mathcal{OPT} = \left\lceil \frac{2p}{C} \right\rceil r + \left(p + 2 \left\lceil \frac{2p}{C} \right\rceil \right) l$$

Annexe B

Partition d'un arbre en sous-arbres

Cette annexe présente deux algorithmes de partitionnement d'un arbre. Le premier effectue un découpage optimal en 3-arbres (*i.e.* qu'il maximise le nombre de 3-arbres) et le deuxième effectue un découpage exclusivement en 3-arbres et 4-arbres (à un arbre près).

B.1 Décomposition optimale d'un arbre en 3-arbres

Nous nous intéressons ici à la décomposition d'un arbre en sous-arbres de 3 arêtes, ou 3-arbres, *i.e.* le problème suivant :

3-TD[1]
Instance : un arbre T, une borne B (entier positif).
Question : existe-t-il une décomposition de T incluant au moins B 3-arbres arêtes-disjoints ?

En fait notre but, pour un arbre T donné, est de trouver la plus grande valeur de B pour laquelle 3-TD a une réponse positive, c'est-à-dire pour laquelle il existe une décomposition maximale (en cardinalité) de T en 3-arbres arêtes-disjoints. Une telle décomposition sera qualifiée de solution optimale pour 3-TD.

Remarquons que vérifier qu'une solution contient B 3-arbres arêtes-disjoints sous-arbres de T est polynomial, et ainsi 3-TD est dans $\mathcal{NP}$. Nous allons montrer qu'il est, en fait, soluble à l'optimum en temps polynomial et nous allons donner un algorithme qui non seulement calcule la borne B optimale mais aussi construit une solution optimale.

Pour commencer, nous avons le résultat suivant :

[1] 3-TD : *3-Tree Decomposition* (décomposition en 3-arbres).

Lemme B.1 *Soit T un arbre de racine r arbitraire. Soit T_k un 3-arbre de racine k, sous-arbre de T, tel que :*

1. *supprimer T_k ne déconnecte pas T ;*
2. *il n'y a, dans aucune branche de T passant par k, de 3-arbre descendant (strictement) plus profondément que T_k.*

Sous ces hypothèses : il existe une solution optimale de 3-TD qui contient T_k.

preuve :

Soit T un arbre et S une solution optimale de 3-TD (sur T) ne contenant pas T_k. Nous allons la transformer en une solution optimale de 3-TD qui inclut T_k.

Tout d'abord : il existe dans T au moins un 3-arbre ayant une ou deux arêtes communes avec T_k (sinon : $S \cup T_k$ serait une solution de 3-TD meilleure que S, ce qui contredirait que S est optimale). De plus il ne peut pas y avoir plus de 3 tels 3-arbres, puisque les éléments de S sont arêtes-disjoints. Notons aussi qu'un 3-arbre ayant plus de 2 arêtes communes avec T_k est exclu, puisque ce serait T_k lui-même.

Il s'en suit qu'il y a 1, 2 ou 3 3-arbres dans S, notés T_k^1, T_k^2, T_k^3, ayant au moins une arête commune avec T_k. Il y a ainsi 3 cas possibles, selon l'existence ou non de T_k^2 et T_k^3.

Cas 1 : seul T_k^1 existe.

Dans ce cas, en échangeant T_k et T_k^1, nous obtenons une nouvelle solution S' ayant la même cardinalité que S et dont tous les éléments sont arêtes-disjoints. Donc : S' est une solution optimale de 3-TD et elle contient T_k.

Cas 2 : seuls T_k^1 et T_k^2 existent

Dans ce cas l'un au moins, disons T_k^1, de ces sous-arbres a une unique arête commune avec T_k, tandis que l'autre, T_k^2, en a 1 ou 2. De plus, il est exclu qu'une arête d'un T_k^i soit adjacente (s'en lui appartenir) à T_k en un nœud autre que k : en effet supprimer T_k déconnecterait cette arête du reste de T, ce qui contredirait l'hypothèse 1. Une conséquence est que T_k a au moins 2 arêtes adjacentes à k et aussi que la figure B.1 présente toutes les configurations possibles des T_k^i par rapport à T_k. Nous constatons ainsi qu'il est toujours possible de construire un arbre T_k' ne comprenant que des arêtes de $\{T_k^1 \cup T_k^2\} \backslash T_k$. De ce fait, $S' = \{S \cup \{T_k, T_k'\}\} \backslash \{T_k^1, T_k^2\}$ est toujours une solution réalisable de 3-TD, et comme sa cardinalité est celle de S, elle est optimale. De ce fait, nous avons bien construit une solution optimale qui contient T_k.

Cas 3 : T_k^1, T_k^2 et T_k^3 existent

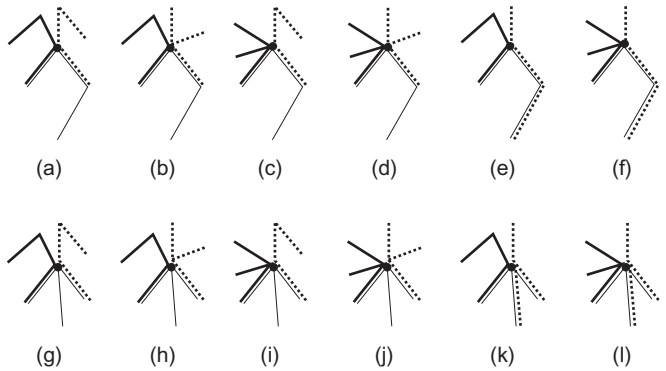

FIG. B.1 – toutes les configurations possibles du cas 2 de la démonstration du lemme B.1. T_k est en trait fin, T_k^1 en trait plein, T_k^2 en trait pointillé et un rond noir indique k.

Dans ce cas : chacun des T_k^i a exactement une arête en commun avec T_k. De ce fait T_k a forcément ses 3 arêtes adjacentes en k. En outre, les T_k^i ne peuvant pas aller plus profondément que T_k (hypothèse 2), deux d'entres eux au moins (ceux qui ne contiennent pas le prédécesseur de k — disons T_k^1 et T_k^2) ont cette même forme, comme illustré sur la figure B.2.

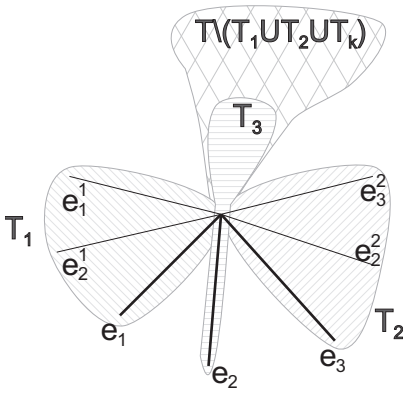

FIG. B.2 – configuration du cas 3 de la démonstration du lemme B.1.

Notons e_1, e_2, e_3 les arêtes de T_k et e_1^i, e_2^i, e_3^i celles de T_k^i, et posons, par exemple (et sans perte de généralité) : $e_1 = e_3^1, e_2 = e_3^3, e_3 = e_1^2$ (*cf* figure B.2). Soient $T_k^4 = \{e_1^3, e_2^3, e_1^1\}$ et $T_k^5 = \{e_1^2, e_2^2, e_2^1\}$. T_k^4 et T_k^5 sont bien des 3-arbres. De plus, soit : $S' = \{S \cup \{T_k, T_k^4, T_k^5\}\} \backslash \{T_k^1, T_k^2, T_k^3\}$. S' est bien une solution de 3-TD (les arbres sont arêtes-disjoints car ils l'étaient dans S et ceux rajoutés le sont entre eux ainsi qu'avec ceux initialement dans S) et S' et de même cardinalité que S. En d'autres termes : S' est une solution optimale de 3-TD qui contient T_k. •

Le lemme précédent nous permet de déduire celui qui suit, premier pas vers une construction récursive d'une solution optimale :

Lemme B.2 *Soit un arbre T et un 3-arbre T_k satisfaisant les hypothèses du lemme B.1. Soit S^* une solution optimale de 3-TD sur $T \backslash T_k$. On a : $S^* \cup T_k$ est une solution optimale sur T.*

preuve :

S^* est une solution de 3-TD sur $T \backslash T_k$ et par conséquent elle ne contient que des 3-arbres arêtes-disjoints ; comme T_k est un 3-arbre arêtes-disjoint avec tous les 3-arbres de S^*, $S^* \cup T_k$ est une solution réalisable de 3-TD sur T.

Soit S une solution de 3-TD sur T, optimale et contenant T_k (elle existe, en vertu du lemme B.1). Si on supprime T_k de S, on obtient une solution réalisable de 3-TD sur $T \backslash T_k$ et donc $|S| - 1 \leq |S^*|$, puisque S^* est optimale. De même, comme $S^* \cup T_k$ est solution de 3-TD sur T, on a : $|S^*| + 1 \leq |S|$. On en déduit que $|S^*| = |S| - 1$, et donc que $S^* \cup T_k$ est optimale pour T. •

Avant de passer à la construction récursive proprement dite d'une solution optimale de 3-TD, nous avons besoin du résultat suivant :

Lemme B.3 *Soit T un arbre de racine r fixée. Soit k un nœud n'ayant pour successeurs que des 2-chaînes, notées $c_1 \dots c_d$ ($d \geq 1$). On note e_k l'arête entre le nœud k et pred(k), son prédécesseur dans T ; on note T_k le sous-arbre de T formé par e_k et les $c_i, 1 \leq i \leq d$ (cf figure B.3). On note $OPT(X)$ une solution optimale de 3-TD sur un arbre X. On a :*

- *si d est pair : $OPT(T_k \backslash e_k) \cup OPT(T \backslash (T_k \backslash e_k))$ est une solution optimale de 3-TD sur T (cf figure B.4 (a)) ;*
- *si d est impair : $OPT(T_k) \cup OPT(T \backslash T_k)$ est une solution optimale de 3-TD sur T (cf figure B.4 (b)).*

De plus : en notant e_i^k l'arête de c_i adjacente à k, une solution optimale de 3-TD sur $\bigcup_{i=1}^{2p} c_i$ est (à numérotation près) : $\bigcup_{i=1}^{p} \{c_i \cup e_{i+p}^k\}$.

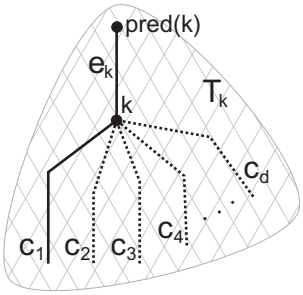

FIG. B.3 – la configuration du lemme B.3.

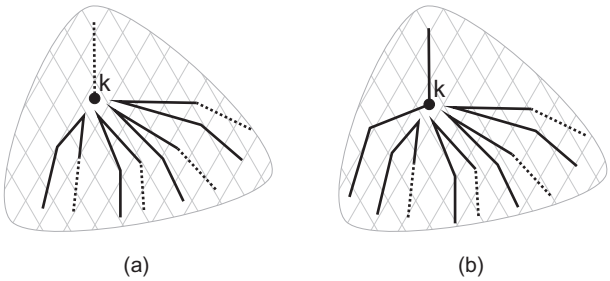

FIG. B.4 – solution optimale selon le lemme B.3. (a) d pair. (b) d impair.

preuve :

Nous disons qu'une solution "utilise" une 2-chaîne c_i si et seulement si au moins une arête de c_i appartient à un élément de S. Par extension, on dira que cet élément aussi "utilise" c_i.

Ceci définit, notons tout d'abord que l'arête d'un c_i qui n'est pas adjacente à k ne peut appartenir qu'à un 3-arbre contenant e_i^k, et cela pour une question de connexité. Aussi, supposons qu'il existe une solution S dont un élément t utilise 3 c_i (disons c_1, c_2, c_3). On a forcément : $t = \{e_1^k, e_2^k, e_3^k\}$ et les 3 autres arêtes de ces c_i ne peuvent appartenir à aucun autre 3-arbre. Donc, $S' = S\backslash\{t\} \cup \{c_1 \cup e_2^k\}$ est une solution réalisable de 3-TD sur T, de même cardinalité que S. Aussi, pour toute solution, il existe une solution de même valeur donc aucun élément n'utilise plus de 2 c_i. À partir de maintenant, nous nous plaçons dans ce cas. Aussi pour S solution réalisable, $t \in S$ utilisera 0, 1 ou 2 c_i. De plus, t n'utilise qu'un c_i si, et seulement si, il utilise e_k aussi.

Une conséquence est qu'il existe une solution de 3-TD sur $\bigcup_{i=1}^{2p} c_i$ dont les éléments utilisent tous 2 c_i. Or en regroupant ainsi les c_i par 2, on ne peut pas faire mieux que : $\bigcup_{i=1}^{p} \{c_i \cup e_{i+p}^k\}$, qui est donc optimale, ce qui prouve la dernière assertion du lemme (on peut même montrer que, à la numérotation des c_i près, cette solution est l'unique solution optimale parmi toutes celles possibles).

Traitons maintenant la première assertion, dans le cas où d est pair. Soit S une solution optimale de 3-TD sur T (dont aucun élément n'utilise plus de 2 c_i).

Supposons qu'il existe $t \in S$ tel que : $e_k \in t$ et $t \cap \{\bigcup c_i\} \neq \emptyset$. Le 3-arbre t, nécessairement unique dans S, utilise l'arête d'un ou deux c_i. Si t n'utilise qu'un c_i, disons c_1, alors comme il y a un nombre pair de c_i et qu'ils ne sont utilisés dans S que par 2, il reste au moins un c_i, disons c_2, qui n'est pas utilisé dans S. Dans ce cas, $S' = S\backslash\{t\} \cup \{c_1 \cup e_2^k\}$ est une solution réalisable de 3-TD sur T, de même cardinalité que S pour laquelle il n'existe pas $t \in S$

tel que : $e_k \in t$ et $t \cap \{\bigcup c_i\} \neq \emptyset$. Si t utilise 2 c_i, disons c_1, c_2, nous pouvons construire de même S', avec les mêmes propriétés.

Il existe donc une solution optimale de 3-TD sur T pour laquelle il n'existe pas $t \in S$ tel que : $e_k \in t$ et $t \cap \{\bigcup c_i\} \neq \emptyset$. Or, si un tel t n'existe pas, c'est que nous pouvons partitionner S en S_1 et S_2 avec S_1 solution réalisable sur $T \backslash (T_k \backslash e_k)$ et S_2 solution réalisable sur $T_k \backslash e_k$. De plus, S_1 et S_2 seront optimales sur leurs ensembles respectifs (sinon S ne serait pas optimale) et donc $OPT(T_k \backslash e_k) \cup OPT(T \backslash (T_k \backslash e_k))$ est optimale sur T.

Traitons maintenant le cas où d est impair. Soit S une solution optimale de 3-TD sur T (dont aucun élément n'utilise plus de 2 c_i).

Si e_k n'est pas utilisé dans S, alors tous les c_i utilisés le sont par paires. Comme il y en a un nombre impair, il en reste au moins un d'inutilisé (disons : c_1). Alors : $S' = S \cup \{e_k \cup c_1\}$ est une solution réalisable de 3-TD sur T, meilleure que S, ce qui est exclu. Donc, e_k est utilisé dans S par un 3-arbre t.

Si t n'utilise aucun c_i, alors il reste au moins un c_i inutilisé (disons : c_1). Alors, $S' = S \backslash \{t\} \cup \{e_k \cup c_1\}$ est une solution réalisable de 3-TD sur T de même cardinalité que S, donc optimale.

Si t utilise un unique c_i, disons c_1, il peut en utiliser une ou deux arêtes, mais dans les 2 cas, $S' = S \backslash \{t\} \cup \{e_k \cup c_1\}$ est une solution réalisable de 3-TD sur T de même cardinalité que S, donc optimale.

En conséquence, il existe une solution optimale sur T qui contient un arbre $t = \{e_k \cup c_1\}$ qui sépare T en $T_1 = T \backslash T_k$ et $T_2 = T_k \backslash t$. Donc, une solution $OPT(T_1) \cup OPT(T2) \cup \{t\}$ est optimale sur T, *i.e.* $OPT(T \backslash T_k) \cup OPT(T_k)$ est optimale sur T. •

Ce qui précède permet de construire récursivement une solution optimale de 3-TD sur un arbre. En effet, chaque lemme permet de décomposer un arbre T en 2 arbres T_1, T_2 en garantissant que l'union de solutions optimales, $OPT(T_1) \cup OPT(T_2)$, est une solution optimale pour l'arbre T. Ainsi nous obtenons le théorème suivant :

Théorème B.1 *Le problème 3-TD est soluble à l'optimum en temps linéaire ($\mathcal{O}(m)$ pour m : nombre d'arêtes de l'arbre).*

preuve :

Notre démonstration de ce théorème est constructive. Elle se base sur l'algorithme B.1, dans lequel nous notons T_i le sous-arbre de première arête $e_{pred(i),i}$ et $p(i)$ le poids (nombre d'arêtes) de cet arbre.

Justifions la validité de cet algorithme.

En début d'itération i de l'étape (2), nous avons l'invariant :

- $p(j)$ est calculé et correct pour tout $j > i$;
- $p(j) < 3$ pour tout $j > i$ (*i.e.* il n'y a pas de 3-arbres possibles avec seulement les arêtes restantes entre des sommets $j > i$) ;

Algorithme B.1 Résolution polynomiale de 3-TD

1. numéroter les nœuds de 1 à n en effectuant un parcours en largeur de T à partir d'une racine arbitraire.
   $S = \emptyset$
2. **pour** chaque nœud $i = n \dots 1$ **faire :**
   (a) calculer le poids de T_i : $p(i) = 1 + \sum_{j \in succ(i)} p(j)$
   (b) **si** $p(i) = 3$ **alors :**
   $S = S \cup \{T_i\}$
   $T = T \backslash T_i$
   (c) **sinon si** $p(i) > 3$ **alors :**
   i. trier les d fils de i par poids croissants (on assimile t_j et T_{t_j}) : $p(t_1) \leq p(t_2) \leq \ldots \leq p(t_d)$
   ii. $j = 1, k = d$
   iii. **tant que** $j < k$ **faire :**
   A. **si** $p(t_j) + p(t_k) = 3$ **alors :**
   $S = S \cup \{\{t_j \cup t_k\}\}$
   $T = T \backslash \{t_j \cup t_k\}$
   $p(i) = p(i) - 3$
   $j = j + 1, k = k - 1$
   B. **sinon si** $p(t_j) + p(t_k) = 2$ **alors :**
   les t_i sont tous des arêtes seules ;
   tant que $j < k - 1$ **faire**
   $S = S \cup \{\{t_j \cup t_{j+1} \cup t_{j+2}\}\}$
   $T = T \backslash \{t_j \cup t_{j+1} \cup t_{j+2}\}$
   $p(i) = p(i) - 3$
   $j = j + 3$
   sortir de la boucle (iii)
   C. **sinon :**
   les t_i sont tous des 2-chaînes ;
   on note e_i^1 l'arête de t_i adjacente à k.
   tant que $j < k$ **faire :**
   $S = S \cup \{\{t_j \cup e_{j+1}^1\}\}$
   $T = T \backslash \{t_j \cup t_{j+1}\}$
   $p(i) = p(i) - 4$
   $j = j + 2$
   sortir de la boucle (iii)
   iv. **si** $p(i) = 3$ **alors :**
   $S = S \cup \{\{T_i\}\}$
   $T = T \backslash T_i$

– S est inclue dans une solution optimale de 3-TD sur T.

Cet invariant est vrai lorsque $i = n$, puisqu'il n'y aucun sommet de numéro plus grand que n et que S est initialement vide. Supposons qu'il est vrai pour $i - 1$, et montrons qu'il est vrai pour i.

Tout d'abord, constatons que $p(i)$ peut être calculé en (2.a) et que sa valeur est juste (cela provient de l'hypothèse de récurrence). Ensuite constatons que la valeur $p(i)$ est corrigée au fur et à mesure que T est élagué et qu'il y a élagage tant que $p(i) > 3$. Pour cela étudions les différentes exécutions possibles de l'algorithme.

Si l'algorithme passe en (b) c'est que le poids de T_i est 3, et donc T_i est un 3-arbre. Or, comme nous avons effectué une numérotation en largeur, i est le plus profond des nœuds non encore exploré ; comme $\forall j > i : p(j) < 3$, par hypothèse, nous ne pouvons pas construire de 3-arbres allant plus profondément que T_i ; de plus, supprimer T_i ne déconnecte pas T. Donc, d'après le lemme B.2, il existe une solution optimale contenant T_i. Comme après l'exécution de (b) on passe au nœud suivant, l'invariant est respecté.

Si l'algorithme passe en (c) c'est que le sous-arbre T_i contient plus de 3 arêtes, et donc nous pouvons y inclure au moins un 3-arbre. En raison des initialisations (i) et (ii), à chaque pas de la boucle (iii) nous aurons que t_j est le plus petit sous-arbre n'incluant qu'un successeur direct de i, et que t_k est le plus grand.

Si la condition (A) est satisfaite alors $t_j \cup t_k$ forment un 3-arbre (ils sont adjacents en i) qui satisfait les hypothèses du lemme B.2 (car l'invariant est vrai pour l'étape précédente). Nous pouvons donc, comme pour (b), inclure cet arbre dans S et le supprimer de T, en corrigeant $p(i)$.

Si la condition (B) est satisfaite, c'est que tous les t_j sont de poids 1 (les poids sont à priori de 0, 1 ou 2 mais $p(t_j) = 0$ signifierait qu'il n'y a pas même d'arête de t_j vers son prédécesseur, donc t_j ne pourrait pas être successeur de i). Nous avons donc d arêtes adjacentes en k, que nous pouvons regrouper par 3 pour former des 3-arbres qui, là encore, respectent les hypothèses du lemme B.2 et donc pour lesquels nous pouvons procéder comme en (b). Lorsque s'achève l'exécution de (B), il reste au plus 2 arêtes pendantes à i et $p(i) \leq 3$. Le point (iv) détectera alors, le cas échéant, que T_i est un 3-arbre. Comme celui-ci respecte les hypothèses du lemme B.2, cela valide (iv) lorsqu'on vient directement de (B) et de ce fait, dans ce cas, l'invariant est respecté.

Si la condition (C) est satisfaite, c'est que les successeurs de i sont des 2-chaînes, c'est-à-dire que nous sommes dans la configuration du lemme B.3. La boucle **tant que** regroupe par 2 toutes ces 2-chaînes, sauf une s'il y en a un nombre impair. Cette dernière sera rassemblée avec l'arête $e_{pred(i),i}$ lors du point (iv). De ce fait, nous respectons la construction du lemme B.3, ce qui valide l'invariant quand on passe par (C).

Si nous passons par (iv) sans provenir ni de (C) ni de (B), c'est qu'il ne restait plus qu'un successeur à i et on a donc $p(i) \leq 3$. Dans le cas d'égalité, T_i respecte les hypothèses du lemme B.2 ce qui fini de justifier la correction de l'invariant.

Si l'algorithme ne passe ni en (b) ni en (c), c'est que $p(i) < 3$ et comme cette valeur est juste et qu'on ne touche pas à S ni à T, l'invariant est respecté.

En conséquent, l'invariant est toujours respecté. Comme après la dernière itération il ne reste dans T qu'au plus 2 arêtes, $S = \emptyset$ est alors optimale. En déroulant la récurrence dans l'autre sens, et en vertu des lemmes précédents, la solution S retournée est bien optimale pour l'arbre initial. Pour avoir la valeur de B optimal, il suffit de compter les éléments de S.

D'un point de vue complexité algorithmique, notons que le parcours en largeur se fait en $\mathcal{O}(m)$ pour l'étape 1. Pour l'étape 2, à l'itération i, on a au pire un tri et 2 parcours de d éléments, avec $d = deg(i) - 1$. Cependant, le tri peut en fait se résumer en une partition entre 2 ensembles : $\{t_j, p(t_j) = 1\}$ et $\{t_j, p(t_j) = 2\}$, ce qui se fait par un simple parcours des d fils. De ce fait, l'itération i se fait en $\mathcal{O}(deg(i))$ et donc l'étape 2 complète en $\sum_i deg(i) = \mathcal{O}(m)$. L'algorithme B.1 est donc bien linéaire. •

L'algorithme B.1 permet en fait de faire mieux qu'une décomposition optimale en 3-arbres. En effet des arêtes ou des 2-chaînes ne sont abandonnées que lorsqu'on est dans la configuration de la figure B.3 ou à la toute fin de l'algorithme, s'il reste (strictement) moins de 3 arêtes. Dans les deux cas, nous remarquons que la décomposition adoptée non seulement maximise le nombre de 3-arbres engendrés, mais aussi minimise le nombre de composantes connexes : dans le cas de la figure B.3, l'alternative a une décomposition en 3-arbres et arêtes serait d'utiliser des 2-chaînes, mais puisqu'il faut 2 2-chaînes pour remplacer 1 3-arbre et 1 arête, nous nous retrouverions avec le même nombre de composantes connexes. Ainsi, dans la mesure où nous ne déconnectons que lorsque nous sommes obligés et que nous déconnectons alors au minimum, nous déduisons le corollaire suivant :

Corollaire B.1.1 *Soit un arbre T ; soit S une décomposition de T en 3-arbres, donnée par l'algorithme B.1. Soit $\{e_1 \dots e_p\}$ et $\{c_1 \dots c_q\}$ les ensembles respectivement d'arêtes et de 2-chaînes non utilisées dans S. On a : $S \cup \{e_1 \dots e_p\} \cup \{c_1 \dots c_q\}$ est une partition de T en un nombre minimum de composantes connexes d'au plus 3 arêtes.*

Pour ce qui concerne l'extension d'une telle méthode à des graphes quelconques, Goldschmidt, Hochbaum et Olinick [12] donne la démonstration, qui leur vient de Tarsi [19], du résultat suivant :

Théorème B.2 *Couvrir un graphe avec un nombre minimum de 3-arbres est $\mathcal{NP}$-dur.*

B.2 Décomposition d'un arbre en 3/4-arbres

Lemme B.4 *Tout arbre est partitionnable en un ensemble d'arbres ayant tous 3 ou 4 arêtes, à l'exception d'un qui peut en avoir moins. De plus, un tel partitionnement est calculable en temps linéaire en le nombre d'arêtes de l'arbre.*

preuve :

Comme pour la la couverture d'un graphe par des 3-arbres, nous allons faire une démonstration constructive en donnant un algorithme (l'algorithme B.2) qui calcule la partition d'un graphe en 3-arbres et 4-arbres (plus une éventuelle 2-chaîne ou arête). Les deux algorithmes se ressemblent beaucoup mais n'ayant pas ici à garantir l'optimalité, cela permettra quelques simplifications par rapport à l'algorithme B.1. Prouvons maintenant la correction de l'algorithme B.2.

Nous notons T_i le sous-arbre de première arête $e_{pred(i),i}$ et $p(i)$ le poids (nombre d'arêtes) de cet arbre. En début d'itération i de l'étape (2), nous avons l'invariant :

- $p(j)$ est calculé et correct pour tout $j > i$;
- $p(j) < 3$ pour tout $j > i$ (*i.e.* il n'y a pas de 3-arbres possibles avec seulement les arêtes restantes entre des sommets $j > i$).

Cet invariant est vrai lorsque $i = n$, puisqu'il n'y aucun sommet de numéro plus grand que n et que S est initialement vide. Supposons qu'il est vrai pour $i - 1$, et montrons qu'il est vrai pour i.

Les successeurs de i ayant un numéro plus grand que i (en raison du parcours en largeur), $p(i)$ est calculable (et est correct) à l'étape (a). Si ensuite la condition (b) est vérifiée, c'est que T_i est un 3-arbre ou un 4-arbre et nous pouvons donc l'inclure dans S et le supprimer de l'arbre T. Le point (b) est ainsi justifié.

Si la condition (c) est satisfaite, T_i a plus de 4 arêtes qu'il faut alors répartir en plusieurs sous-arbres. Le tri (i) sépare les arêtes (en début de la liste des t_i) des 2-chaînes (en fin). Il ne peut pas y avoir autre chose car les successeurs de i sont des $j > i$ (parcours en largeur) et donc, par hypothèse de récurrence, leur poids $p(t_j)$ est de 1 ou 2. Nous pouvons faire des regroupements tant qu'il y a au moins 2 descendants (condition (ii)). Lorsque (A) est vérifié c'est que t_d (au moins) est une 2-chaîne tandis que t_{d-1} est une arête ou une 2-chaîne. Dans les 2 cas, $t_d \cup t_{d-1}$ est un arbre (de 3 ou 4 arêtes), puisque t_d et t_{d-1} sont adjacents en i. Nous pouvons donc les supprimer de T et les inclure dans S.

Lorsque la condition (B) est vérifiée, c'est qu'il ne reste plus, comme t_j, que des arêtes adjacentes en i. Nous pouvons donc, comme il est fait,

Algorithme B.2 Couverture d'un arbre T par des 3/4-arbres

1. numéroter les nœuds de 1 à n en effectuant un parcours en largeur de T à partir d'une racine arbitraire.
   $S = \emptyset$
2. **pour** chaque nœud $i = n \dots 1$ **faire :**
   (a) calculer le poids de T_i : $p(i) = 1 + \sum_{j \in succ(i)} p(j)$
   (b) **si** $p(i) = 3$ ou $p(i) = 4$ **alors :**
   $S = S \cup \{T_i\}$
   $T = T \backslash T_i$
   (c) **sinon si** $p(i) > 4$ **alors :**
   i. trier les d fils de i par poids croissants (on assimile t_j et T_{t_j}) : $p(t_1) \leq p(t_2) \leq \ldots \leq p(t_d)$
   ii. **tant que** $d > 1$ **faire :**
   A. **si** $p(t_d) + p(t_{d-1}) >= 3$ **alors :**
   $S = S \cup \{\{t_d \cup t_{d-1}\}\}$
   $T = T \backslash \{t_d \cup t_{d-1}\}$
   $p(i) = p(i) - p(t_d) - p(t_{d-1})$
   $d = d - 2$
   B. **sinon :**
   les t_i sont tous des arêtes seules ;
   tant que $d > 2$ **faire**
   $S = S \cup \{\{t_d \cup t_{d-1} \cup t_{d-2}\}\}$
   $T = T \backslash \{t_d \cup t_{d-1} \cup t_{d-2}\}$
   $p(i) = p(i) - 3$
   $d = d - 3$
   sortir de la boucle (ii)
   iii. **si** $p(i) = 3$ ou $p(i) = 4$ **alors :**
   $S = S \cup \{\{T_i\}\}$
   $T = T \backslash T_i$
3. **si** $T_i \neq \emptyset$ **alors :** $S = S \cup \{T_i\}$

les regrouper par 3 pour former des 3-arbres que nous mettons dans S et supprimons de T.

Lorsque la boucle (ii) s'achève (quelle que soit la manière) le poids de i est d'au plus 4, puisqu'il a soit au plus un successeur (sortie normale de la boucle) dont le poids est au plus 2, soit au plus 2 successeurs (sortie par (B)) mais qui sont alors des arêtes, donc ayant un poids total d'au plus 2. De ce fait le passage par (iii) mettra s'il y a lieu T_i dans S en le supprimant de T, ce qui fini de justifier la validité de l'invariant lorsqu'on passe par (c).

Si ni la condition (b) ni la condition (c) ne sont satisfaites, c'est que T_i est un arbre d'au plus 2 arêtes, donc sans rien retoucher l'invariant est respecté.

Lorsque l'étape (2) s'achève, T_i a un poids d'au plus 2. C'est l'éventuel 2-chaîne ou arête annoncée, puisque à toutes les autres étapes seuls des 3-arbres et des 4-arbres sont ajoutés à S.

Le temps de calcul de cette algorithme est le même que celui de l'algorithme B.1 : il est donc linéaire. •

Remarque : L'algorithme B.2, bien que suffisant pour notre propos, est très loin d'être optimal, au sens de la minimisation du nombre de 3/4-arbres. En effet il privilégie les 3-arbres aux 4-arbres, ce qui entraîne un sur-coût. Des améliorations seraient de grouper les arêtes d'abord par 4 puis par 3 à l'étape (B) ; d'essayer d'ajouter une arête seule lorsqu'on a $p(t_d) + p(t_{d-1}) = 3$ à l'étape (A), et plus généralement chaque fois qu'un 3-arbre est ajouté à S.

Annexe C

Ouverture des cycles d'un graphe

Plusieurs algorithmes étudiés dans ce rapport ont été conçus pour des graphes de demandes r-arborescents puis ont été étendus aux graphes connexes quelconques. Cette extension passe par une procédure d'ouverture des cycles d'un graphe afin de transformer un graphe connexe G en un arbre $T(G)$ de racine r.

Goldschmidt, Hochbaum et Olinick [12] ont présenté formellement une telle transformation et ont montré d'une part qu'elle était applicable en temps linéaire, et d'autre part qu'en refermant les cycles d'une solution de ADR sur $T(G)$ on obtient une solution de ADR sur G de coût au pire égal. Nous présentons ici les idées générales de cette procédure.

Pour qu'à une solution sur $T(G)$ corresponde une solution sur G, il suffit qu'il y ait une bijection entre les arêtes de $T(G)$ et celles de G. Pour qu'il n'y ait pas d'augmentation de coût lors de la transformation réciproque d'une solution de $T(G)$ en une solution de G, il suffit que les images dans G de deux arêtes adjacentes dans $T(G)$ soient elles aussi adjacentes (ainsi, le nombre de raccords ne peut pas augmenter).

Une construction possible d'un $T(G)$ qui respecte ces critères consiste à partir d'un arbre couvrant de G. Ensuite, pour chaque arête $e_{ij} \in G$ qui n'a pas encore été prise en compte, on crée un nœud fictif k et on ajoute à $T(G)$ e_{ik} (ou e_{jk}, les deux nœuds se trouvant dans $T(G)$, cela n'a pas d'importance). La figure C.1 illustre cette construction de $T(G)$, tandis que la figure C.2 montre le passage d'une solution sur $T(G)$ à une solution sur G.

Une telle manière de faire présente l'avantage d'être linéaire et aussi de pouvoir se combiner avec une numérotation en largeur des arêtes de $T(G)$ qui nous est utile pour la plupart des algorithmes travaillant sur des arbres.

FIG. C.1 – Un exemple d'ouverture des cycles d'un graphe.

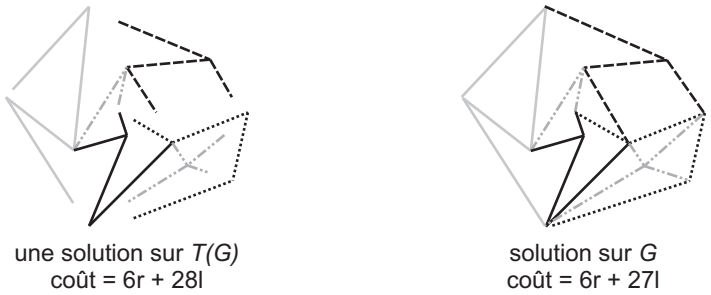

FIG. C.2 – Passage d'une solution sur $T(G)$ à une solution sur G (G et $T(G)$ sont ceux de la figure C.1).

Annexe D

Concaténation des anneaux d'une solution

Nous avons conçu un algorithme glouton de concaténation des anneaux d'une solution entre eux qui sert notamment dans la diversification des solutions lors de la recherche tabou. Cet algorithme modifie une solution réalisable en diminuant son nombre d'anneaux et son nombre de raccords ce qui permet un gain, surtout pour des solutions composées de nombreux anneaux peu pleins.

L'idée de base de l'algorithme est de concaténer des anneaux ayant de nombreux nœuds en commun. Cependant, nous ramenons ce nombre de nœuds communs à la demande satisfaite : puisqu'on peut s'attendre à ce qu'un anneau satisfaisant une grande demande contienne en moyenne plus de nœuds (et donc plus de nœuds en commun avec un autre anneau) qu'un anneau satisfaisant une faible demande, cela permet un re-équilibrage en faveur des petits anneaux dans l'espoir que plusieurs petits anneaux, satisfaisant globalement une même demande qu'un unique plus gros, pourront être regroupés et baisser ainsi plus significativement le nombre de raccords et le nombre d'anneaux de la solution.

Voici maintenant l'algorithme utilisé (nous notons $libre(r)$ la place libre dans l'anneau r, $satisf(r)$ la demande satisfaite par l'anneau r et $com(r, s)$ le nombre de nœuds communs entre les anneaux r et s) :

Algorithme D.1 Concaténation des anneaux d'une solution

1. `soit` $\mathcal{R}$ `l'ensemble des anneaux de la solution.`
   `soit` $\mathcal{R}' = \emptyset$
2. **tant que** $\mathcal{R} \neq \emptyset$ **faire :**
   (a) `prendre` $r_1 \in \mathcal{R}$
   (b) `soit` $\mathcal{Q} = \{r \in \mathcal{R} \backslash \{r_1\} | satisf(r) \leq libre(r_1)\}$
   (c) **si** $Q \neq \emptyset$ **alors :**

i. soit $r_2 = argmax \left\{ \frac{com(r_1,r)}{satisf(r)}, r \in \mathcal{Q} \right\}$

ii. $r_1 = r_1 \cup r_2$
$\mathcal{R} = \mathcal{R} \backslash \{r_2\}$

iii. retourner en (b).

(d) $\mathcal{R}' = \mathcal{R}' \cup \{r_1\}$
$\mathcal{R} = \mathcal{R} \backslash \{r_1\}$

3. $\mathcal{R}'$ est le nouvel ensemble d'anneaux de la solution

Remarque 1 : Aux étapes (a) et (i), il peut y avoir un choix arbitraire à faire (respectivement pour r_1 et r_2). Dans notre implantation, les anneaux sont gérés sous forme de liste aussi prenons nous simplement le premier élément de la liste qui convient.

Remarque 2 : Cet algorithme de concaténation des anneaux n'est pas optimal (il ne minimise pas le nombre d'anneaux). Si tel était le cas, nous aurions un algorithme polynomial pour résoudre les problèmes ADR et BIN-PACKING, qui sont $\mathcal{NP}$-complet !

Annexe E

Valeurs numériques de coefficients d'approximation

Cette annexe donne des valeurs numériques pour le coefficient α des approximations pour ADRL étudiées au chapitre 3, ceci afin de mieux apprécier les qualités relatives et les limites de ces algorithmes.

Dans ce qui suit nous avons une capacité C et le graphe possède m arêtes. De plus, nous envisageons des restrictions, respectivement :

1. graphe connexe
2. graphe connexe sans cycle de C (ou moins) arêtes
3. graphe eulérien

		restriction	
C	m	(1)	(2)
5	10	2,37	1,25
5	50	2,37	1,25
5	100	2,37	1,25
5	1000	2,37	1,25
20	10	4,74	1,43
20	50	4,74	1,43
20	100	4,74	1,43
20	1000	4,74	1,43
50	10	7,50	1,47
50	50	7,50	1,47
50	100	7,50	1,47
50	1000	7,50	1,47
100	10	10,61	1,49
100	50	10,61	1,49
100	100	10,61	1,49
100	1000	10,61	1,49

Couverture par des 2-chaînes

		restriction	
C	m	(1)	(2)
5	10	2,45	1,29
5	50	2,39	1,26
5	100	2,38	1,25
5	1000	2,37	1,25
20	10	4,90	1,48
20	50	4,78	1,44
20	100	4,76	1,43
20	1000	4,74	1,43
50	10	7,75	1,52
50	50	7,55	1,48
50	100	7,53	1,48
50	1000	7,50	1,47
100	10	10,96	1,53
100	50	10,68	1,50
100	100	10,64	1,49
100	1000	10,61	1,49

Couverture par des 3-arbres

C	m	restriction (1)	restriction (2)
5	10	2,21	1,33
5	50	2,13	1,16
5	100	2,12	1,13
5	1000	2,11	1,11
20	10	4,43	1,52
20	50	4,26	1,32
20	100	4,24	1,30
20	1000	4,22	1,27
50	10	7,00	1,57
50	50	6,73	1,36
50	100	6,70	1,33
50	1000	6,67	1,31
100	10	9,90	1,58
100	50	9,52	1,37
100	100	9,48	1,35
100	1000	9,43	1,32

Couverture par des 3-arbres et des 4-arbres

C	m	restriction (1)	restriction (2)
5	10	2,21	1,25
5	50	2,21	1,18
5	100	2,21	1,18
5	1000	2,21	1,17
20	10	3,48	1,14
20	50	3,48	1,07
20	100	3,48	1,06
20	1000	3,48	1,05
50	10	5,20	1,12
50	50	5,20	1,04
50	100	5,20	1,03
50	1000	5,20	1,02
100	10	7,21	1,11
100	50	7,21	1,03
100	100	7,21	1,02
100	1000	7,21	1,01

Couverture par des p-arbres, $\frac{C}{2} \leq p \leq C$

C	m	restriction (1)	restriction (2)
5	10	2,48	1,93
5	50	2,19	1,90
5	100	2,11	1,90
5	1000	1,97	1,90
20	10	4,43	3,34
20	50	3,89	3,32
20	100	3,74	3,32
20	1000	3,46	3,32
50	10	6,85	5,11
50	50	6,00	5,10
50	100	5,76	5,10
50	1000	5,32	5,10
100	10	9,60	7,15
100	50	8,42	7,14
100	100	8,07	7,14
100	1000	7,45	7,14

Heuristique eulérienne

Annexe F

Jeux de données

Toutes nos expérimentations ont nécessité des données et celles-ci ont pu joué un rôle dans les conclusions obtenues. Il est donc normal de présenter les données utilisées, ce qui est le but de cette annexe.

Il faut tout d'abord savoir que nous n'avons pas eu accès à des données réelles et nous avons donc du nous rabattre sur des données que nous avons nous même générées.

Le premier jeux de graphes a été généré aléatoirement en fonction d'un nombre de sommets n, d'une probabilité d'existence d'une arête p et d'une demande maximale $d_{\max}$: pour chaque couple de sommets $(i, j), i < j$, l'arête e_{ij} est créée avec la probabilité p et alors la demande correspondante d_{ij} est choisie uniformément sur $\{1 \dots d_{\max}\}$. Une autre approche consiste à donner, au lieu de p, une demande totale D et d'en déduire la probabilité à utiliser pour engendrer un graphe aléatoire pour lequel la somme des demandes est, en moyenne, de D (on prend $p = \frac{4D}{d_{\max}(n-1)n}$).

Afin d'avoir des données plus variées (voire plus réalistes), nous avons ensuite construit des graphes selon des principes dus à Goldschmidt, Laugier et Olinick [13] : n sommets sont choisis aléatoirement dans le plan et une arête existe si et seulement si la distance euclidienne entre ses deux extrémités est inférieure à une borne donnée. La demande le long d'une arête est de 3 avec la probabilité 4%, et de 2 avec la probabilité 32% et de 1 avec une probabilité de 64%.

Nous avons aussi travaillé sur des graphes particuliers. Ainsi des cas où nous connaissons une solution optimale ou une borne assez bonne (par exemple : un arbre, une grille, un graphe biparti-complet) nous ont permis de tester, au moins sur ces cas, les performances de notre recherche tabou et de nos heuristiques.

Il faut de plus constater que certains de ces graphes particuliers sont en fait très proches de situations réelles, comme il est décrit dans [5] : ainsi un réseau où tout le monde communique directement avec tout le monde est un graphe complet K_n ; en ne communiquant qu'avec ses 2 voisins on obtient

un cycle ; lorsqu'on a un serveur face à n client, cela forme une étoile $K_{1,n}$, voire un graphe $K_{2,n}$ si on double le serveur pour des questions de fiabilité ou de performances. Aussi avoir de bonnes performances de nos heuristiques sur ces cas particuliers est une chose souhaitable.

Annexe G

Programmation et temps de calcul

Toute la programmation a été faite en C++ avec le compilateur GNU gcc-2.95.2 pour Mingw et a été exécutée sur un Penium II avec 64Mo de mémoire tournant sous Windows 98. Avec une telle configuration, les temps de calcul obtenus sont tout à fait raisonnables.

Comme l'illustre les tableaux G.1(a) et (b), pour toutes les instances testées, l'exécution des algorithmes gloutons (*cf* chapitre 4) est quasiment instantanée, mis à part pour l'algorithme de recherche de la meilleure demande, bien qu'il reste très rapide, même sur de grosses instances.

Pour la recherche tabou (*cf* chapitre 5), si les paramètres sont choisis raisonnablement, sa durée est tout à fait acceptable, comme l'illustre les tableaux G.2 (a) et (b).

Vis-à-vis de ces temps d'exécution, il faut ajouter que nos programmes sont relativement génériques dans la mesure où une seule implantation a été faite pour toutes les variantes, notamment de la recherche tabou, cela afin de faciliter la comparaison des différentes versions, par exemple déterminer la gestion des tabous qui est préférable. Au vu de nos résultats, on peut re-écrire un logiciel n'intégrant que les éléments les plus prometteurs et pour lequel on peut ainsi adapter au mieux l'implantation et ainsi accélérer notablement les temps donnés ici.

Nous ne donnons pas les pages de code que constitue l'ensemble des programmes réalisés pour ce projet. En effet, cela représente peu d'intérêt, surtout dans notre cadre où les algorithmes et leurs performances doivent attirer l'attention beaucoup plus que des détails d'implantation.

Paramètres		**Temps d'exécution (sec.)**						
m	C	(1)	(2)	(3)	(4)	(5)	(6)	(7)
8517	75	0,11	1,43	0,17	1,48	0,49	0,28	31,42
8517	50	0,11	1,48	0,17	1,49	0,44	0,22	24,66
8517	20	0,11	1,48	0,27	1,60	0,61	0,22	14,23
5726	75	0,05	0,66	0,05	0,66	0,60	0,16	13,95
5726	50	0,05	0,66	0,11	0,65	0,28	0,17	11,10
5726	20	0,05	0,65	0,11	0,72	0,33	0,11	6,37
2797	75	0,00	0,16	0,06	0,16	0,27	0,05	3,13
2797	50	0,00	0,17	0,06	0,16	0,22	0,06	2,58
2797	20	0,06	0,16	0,06	0,17	0,11	0,05	1,54
1440	75	0,00	0,05	0,00	0,05	0,05	0,00	0,72
1440	50	0,00	0,05	0,00	0,05	0,06	0,00	0,60
1440	20	0,00	0,05	0,00	0,06	0,05	0,00	0,38

(a)

Paramètres		**Temps d'exécution (sec.)**						
m	C	(1)	(2)	(3)	(4)	(5)	(6)	(7)
8567	75	1,21	1,54	1,32	1,65	2,03	0,22	15,71
8567	50	1,21	1,54	1,43	1,76	2,20	0,16	12,20
8567	20	1,31	1,65	1,98	2,26	3,84	0,16	6,21
5754	75	0,55	0,71	0,61	0,71	0,99	0,11	7,09
5754	50	0,55	0,66	0,65	0,77	1,04	0,11	5,49
5754	20	0,60	0,71	0,87	0,99	1,71	0,11	2,86
2934	75	0,17	0,17	0,17	0,17	0,28	0,06	1,87
2934	50	0,16	0,16	0,16	0,22	0,27	0,05	1,43
2934	20	0,17	0,17	0,22	0,28	0,44	0,06	0,77
1440	75	0,00	0,05	0,06	0,05	0,05	0,00	0,44
1440	50	0,05	0,06	0,05	0,06	0,06	0,00	0,33
1440	20	0,05	0,06	0,05	0,06	0,11	0,00	0,22

(b)

TAB. G.1 – Temps d'exécution des algorithmes gloutons sur des graphes de m demandes et une capacité C. Les algorithmes sont : (1) `bin_packing1`, (2) `bin_packing2`, (3) `bin_packing3`, (4) `bin_packing4`, (5) Recherche du meilleur nœud, (6) Chemin pseudo-eulérien, (7) Recherche de la meilleure demande. Les demandes sont (a) unitaires, (b) dans $\{1, \ldots, 5\}$.

Paramètres		**Temps d'exécution (min.)**		
m	C	$n_t = 35$	$n_t = 25$	$n_t = 15$
1440	75	24,43	40,34	23,74
1440	50	26,97	35,94	37,33
1440	20	29,92	29,94	30,08
929	75	14,49	14,50	14,52
929	50	13,54	13,60	7,61
929	20	11,76	11,74	11,75
708	75	10,24	6,30	9,49
708	50	9,38	9,42	6,69
708	20	7,51	7,52	7,59
321	75	1,66	1,06	0,99
321	50	1,62	1,03	0,97
321	20	1,40	1,40	1,42

(a)

Paramètres		**Temps d'exécution (min.)**		
m	C	$n_t = 35$	$n_t = 25$	$n_t = 15$
1440	75	14,19	13,74	14,70
1440	50	8,34	8,30	8,29
1440	20	6,93	6,91	6,89
929	75	6,37	6,06	6,27
929	50	3,29	3,28	3,90
929	20	2,80	2,80	3,45
708	75	2,73	2,75	2,72
708	50	2,38	2,34	2,35
708	20	1,61	1,61	2,44
321	75	0,85	0,87	0,90
321	50	0,46	0,47	0,47
321	20	0,33	0,33	0,46

(b)

TAB. G.2 – Temps d'exécution pour 1000 itérations d'une recherche tabou sur un graphe de m demandes et une capacité C. Il y a n_t tabous et les demandes sont (a) unitaires, (b) dans $\{1, \ldots, 5\}$.

Annexe H

Modélisation par valuation des sommets

Dans tout ce rapport les modèles utilisés privilégiaient les arêtes par rapport aux sommets. Une alternative à cette approche serait donc de ne plus chercher un partitionnement des arêtes mais une valuation des sommets : nous associons à chaque sommet $i \in V$ d'un graphe $G = (V, E)$ une variable $x_i \in \mathbb{N}$ représentant le nombre d'anneaux auxquels cette arête est rattachée. Deux problèmes se posent alors : trouver des contraintes qui permettent d'assurer qu'une valuation correspond à (au moins) une solution réalisable de ADR et déterminer une telle solution.

Cette approche est beaucoup mieux adaptée pour le sous-cas ADRL (*i.e.* $l = 0$) que pour le cas général. En effet, notons que si une solution correspond à une valuation, son nombre de raccords sera obligatoirement $L = \sum_{i \in V} x_i$. Par contre son nombre d'anneaux, lui, n'est pas du tout évident et peut même prendre différentes valeurs pour une même valuation, comme l'illustre la figure H.1.

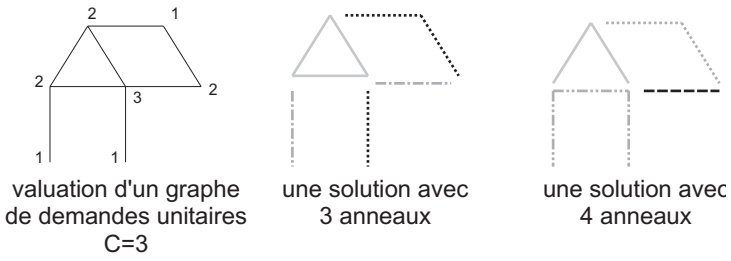

FIG. H.1 – Un exemple de valuation pour laquelle existent des solutions avec différents nombres d'anneaux.

En se limitant aux solutions arborescentes nous avons toutefois la propriété suivante :

Propriété H.1 *Soit un graphe $G = (V, E)$; on note $n = |V|$ et $m = |E|$. Soit $X = \{x_1, \dots x_n\}$ une valuation des sommets de G. Toute partition des arêtes de G en arbres qui respecte la valuation X aura exactement $(\sum_{i=1}^{n} x_i) - m$ composantes connexes.*

preuve :

Soit S une partition de G en arbres qui respecte la valuation X. Notons $y_i, 1 \leq i \leq m$ le nombre d'arbres de S ayant i arêtes. Le nombre de composantes connexes de S est $\sum_{i=1}^{m} y_i$. Or le nombre total d'arêtes de G est : $m = \sum_{i=1}^{m} i.y_i$ puisque S est une partition de G et que chaque i-arbre a i arêtes. De plus chaque i-arbre contient $i + 1$ sommets, ce qui donne : $\sum_{i=1}^{n} x_i = \sum_{i=1}^{m} (i + 1)y_i$. En soustrayant ces deux équations, on obtient : $\sum_{i=1}^{m} y_i = (\sum_{i=1}^{n} x_i) - m$. •

Nous pouvons même conclure lorsque $C = 2$. En effet dans ce cas une solution de ADR est arborescente et il y a y_1 arêtes solitaires et y_2 2-chaînes. Nous avons les relations :

$$\begin{cases} y_1 + 2y_2 = m \\ 2y_1 + 3y_2 = \sum_{i=1}^{n} x_i \end{cases}$$

Le déterminant de ce système est non nul, ce qui fait qu'il existe une solution unique. Cela se traduit par la propriété :

Propriété H.2 *Soit G un graphe et X une valuation de ses sommets. On a :*

1. *Toutes les solutions de ADR sur G pour $C = 2$ et qui respectent X sont composées du même nombre de 2-chaînes et d'arêtes solitaires.*
2. *Si les demandes sont unitaires et si les arêtes solitaires sont regroupées alors toutes les solutions de ADR qui respectent X ont le même nombre d'anneaux, et donc le même coût.*

Dans le cas général nous ne savons pas conclure. Toutefois, nous avons une liste de contraintes qu'une valuation doit respecter. Malheureusement cette liste est insuffisante pour garantir qu'il existe des solutions réalisables de ADR qui satisfont la valuation. Ces contraintes sont :

Propriété H.3 *Soit un graphe $G = (V, E)$ et $n = |V|$, $m = |E|$. Soit $X = \{x_1, \dots x_n\}$ une valuation des sommets de G. Le degré d'un sommet $i \in V$ est noté $deg(i)$; pour $S \subset V$, $\delta(S)$ est l'ensemble des arêtes ayant au moins une extrémité dans S.*

Pour qu'existe une solution de ADR réalisable pour une capacité C qui

respecte la valuation X il faut que les contraintes suivantes soient respectées :

$$x_i \leq deg(i) \qquad \forall i \in V \tag{H.1}$$

$$x_i \geq \left\lceil \frac{deg(i)}{C} \right\rceil \qquad \forall i \in V \tag{H.2}$$

$$\sum_{i \in S} x_i \geq \left\lceil \frac{|\delta(S)|}{C} \right\rceil \qquad \forall S \subset V \tag{H.3}$$

$$\sum_{i \in V} x_i \geq \left\lceil \frac{1 + \sqrt{1 + 8C}}{2} \right\rceil \frac{|E|}{C} \tag{H.4}$$

$$\sum_{i \in V} x_i \geq \sum_{i \in V} \left\lceil \frac{deg(i)}{C} \right\rceil \tag{H.5}$$

$$\sum_{j \in V, e_{ij} \in E} x_j \geq x_i \qquad \forall i \in V \tag{H.6}$$

preuve :

Nous nous plaçons dans le cas d'une solution réalisable de ADR pour une capacité C.

Une arête n'appartient qu'à un anneau. De ce fait, un sommet ne peut appartenir à plus d'anneaux que ce qu'il a d'arêtes incidentes. Cela justifie la contrainte (H.1).

La contrainte (H.2) est un cas particulier, quand $S = \{i\}$, de la contrainte (H.3). Cette dernière vient du fait qu'un anneau ne peut contenir plus de C arêtes, aussi s'il y a $|\delta(S)|$ arêtes adjacentes aux sommets de S, il faut que ces sommets appartiennent à au moins $\left\lceil \frac{|\delta(S)|}{C} \right\rceil$ anneaux.

La contrainte (H.4) traduit l'équation (2.2) de la propriété 2.3 page 16.

La contrainte (H.5) est une sommation des contraintes (H.2).

La contrainte (H.6) traduit que les demandes des voisins d'un sommet i sont répartis sur au moins autant d'anneaux que la demande de i, puisqu'elle l'inclut. •

Cette approche par valuation des sommets peut s'avérer complémentaire de celle que nous avons adoptée dans le reste de ce rapport, mais nous la laissons à peine ébauchée en ne répondant que partiellement à la question de savoir si une valuation est réalisable, et en ne répondant pas du tout à comment construire une solution qui respecte une valuation donnée.

Bibliographie

[1] The SONET Home Page. http ://www.sonet.com.

[2] Nadia Brauner, Yves Crama, and Christelle Wynants. Conception of a Telecommunication Network. private communication, August 2000.

[3] ETSI. ETSI - Telecom Standards. http ://www.etsi.org.

[4] ETSI. Transmission and Multiplexing (TM) ; Synchronous Digital Hierarchy (SDH) ; Network protection schemes ; Rings and other schemes. Technical specification, November 1997. ref : TS 101 010 v1.1.1.

[5] ETSI. Transmission and Multiplexing (TM) ; Synchronous Digital Hierarchy (SDH) ; Network protection schemes ; Types and characteristics. Technical specification, November 1997. ref : TS 101 009 v1.1.1.

[6] Bernard Fortz, Patrick Soriano, and Christelle Wynants. A Tabu Search Algorithm for Self-Healing Ring Network Design. Technical Report SMG 2000/15, Service des Mathématiques de la Gestion, Université Libre de Bruxelles, May 2000.

[7] Michael R. Garey and David S. Johnson. *Computer and Intractability (A Guide to the Theory of NP-Completeness).* W.H. Freeman And Company, 1979.

[8] Michael R. Garey and David S. Johnson. A 71/60 Theorem for Bin-Packing. *Journal of Complexity*, 1(1) :65–106, 1985.

[9] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operation Research*, (5) :533–549, 1986.

[10] Fred Glover and Manuel Laguna. *Modern Heuristic Techniques for Combinatorial Problems*, chapter 3 : Tabu Search. *in* Colin R. Reeves (editor), Blackwell Scientific Publications edition, 1993.

[11] Fred Glover and Manuel Laguna. *Tabu Search.* Kluwer Academic Publishers, London, 1997.

[12] Olivier Goldschmidt, Dorit S. Hochbaum, and Eli V. Olinick. The SONET Edge-Partition Problem, February 2001. (draft copy).

[13] Olivier Goldschmidt, Alexandre Laugier, and Eli V. Olinick. SONET/SDH Ring Assignment with Capacity Constraints, January 1998.

[14] P. Hansen. The Steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy, 1986.

[15] Ian Holyer. The NP-completeness of some edge-partition problems. *SIAM Journal of Computing*, 4(10) :713–717, November 1981.

[16] David S. Johnson, Alan Demers, Jeffrey D. Ullman, Michael R. Garey, and R.L. Graham. Worst-case performance bounds for simple one-dimensionnal packing algorithm. *SIAM journal of Computing*, 3(4) :299–325, December 1974.

[17] Shigeru Masuyama and Toshihide Ibaraki. Chain Packing in Graphs. *Algorithmica*, 6 :826–839, 1991.

[18] Demetris Mili. Self-Healing Ring Architectures for SONET Network Applications. http ://www.doc.ic.ac.uk/∼nd/surprise_96/journal/vol2/dm9/article2.html.

[19] M. Tarsi. private communication, 1998.