Thèse soutenue le 24 novembre 2006 pour obtenir le titre de

DOCTEUR de L'UNIVERSITÉ D'ÉVRY VAL D'ESSONNE

Spécialité Informatique

Algorithmes d'approximation pour l'optimisation en ligne d'ordonnancements et de structures de communications

Nicolas Thibault

Composition du jury

Rapporteurs :	M. Marc Demange	(Professeur à l'ESSEC)
	M. Vassilis Zissimopoulos	(Professeur à l'Université d'Athènes)
Examinateurs :	M. Evripidis Bampis	(Professeur à l'Université d'Évry)
	M. Ralf Klasing	(Chargé de recherche au LaBRI)
Président du jury :	M. Pierre Fraigniaud	(Directeur de recherche au LRI)
Directeur de thèse :	M. Christian Laforest	(Maître de conférences à l'Université d'Évry)

Thèse préparée au Laboratoire

Informatique, Biologie Intégrative et Systèmes Complexes
de l'Université d'Évry Val d'Essonne – FRE CNRS 2873

Table des matières

Introduction générale

La réservation de ressources dans un réseau de communication est un domaine d'application très vaste, qui implique des problèmes algorithmiques nombreux et variés. La modélisation d'un réseau sous la forme d'un *graphe* permet, indépendamment de sa nature physique (éventuellement hétérogène), de développer des algorithmes pour déterminer quels sont les noeuds et/ou les liens du réseau qui doivent être réservés pour répondre à un problème donné. Il s'agit alors de proposer des algorithmes gérant au mieux l'attribution de ces ressources pour les membres qui inter-agissent via le réseau.

Pour la résolution de problèmes liés à la réservation de ressources au niveau d'un lien particulier (indépendamment du reste du réseau), la modélisation sous forme d'*ordonnancement* est alors particulièrement adaptée. Il s'agit dans ce cas de traiter des demandes de réservation localisées sur un lien du réseau, en proposant des algorithmes pour gérer au mieux l'attribution des ressources sur ce lien. Dans cette thèse nous nous intéressons à la résolution d'un certain nombre de problèmes algorithmiques relatifs à la réservation de ressources dans un réseau, d'un point de vue global d'une part (problèmes de graphe), et d'un point de vue localisé, centré sur la gestion d'un lien d'autre part (problèmes d'ordonnancement).

Dans ce type de problèmes, il n'est pas toujours possible de connaître à l'avance toutes les données à traiter. En effet, les demandes des membres amenés à communiquer via le réseau ainsi que les demandes de réservation au niveau des liens arrivent en pratique au fils de l'eau. Pour prendre en compte cette difficulté, nous nous plaçons dans le contexte de l'algorithmique *on-line* (*en ligne*). Dans ce contexte, qui est à opposer à celui de l'algorithmique *off-line* (*hors ligne*), les données du problème à traiter ne sont pas connues dès le départ, mais *révélées au fur et à mesure*, sans *aucune connaissance du futur*. Un algorithme on-line peut donc prendre chaque nouvelle décision en fonction du passé, du présent, mais ne connaît rien du futur.

Pour chacun des problèmes traités, notre démarche est la suivante. Chaque solution construite par un des algorithmes on-line que nous proposons est évaluée *analytiquement* en fonction d'un (ou parfois plusieurs) critère(s) de performance. Pour cela, nous avons choisi de comparer la solution on-line courante à la *meilleure solution off-line* possible (démarche classique dans le cadre de l'algorithmique on-line). Cela permet de donner une garantie sur la performance d'un algorithme *au pire cas*, c'est-à-dire de borner pour toute exécution le rapport entre la performance de l'algorithme on-line évalué et la performance off-line optimale.

Nous allons maintenant préciser les problèmes on-line liés à la réservation de ressources que nous avons traités, sous la forme de problèmes de graphes d'une part (objet de la première partie de la thèse, intitulée *Groupes dynamiques dans un graphe*), puis sous la forme de problèmes d'ordonnancements d'autre part (objet de la seconde partie de la thèse, intitulée *Ordonnancements on-line*).

Groupes dynamiques dans un graphe. La construction d'une structure connectant dans un réseau un sous-ensemble *statique* (qui n'évolue pas dans le temps) de membres est un problème classique. On peut en effet évoquer toute forme de forum ou de réunion dans un réseau, où les membres veulent échanger des données via une structure dédiée. Mais, dans ce type de rassemblements virtuels, il n'est pas toujours possible de connaître toutes les données à l'avance : les membres peuvent arriver ou partir à tout moment (version *dynamique* du problème). Par exemple, dans les systèmes pair à pair (*peer-to-peer*), il est inconcevable de prédire qui va échanger des données avec qui. Les échanges se font "au fil de l'eau", en fonction de paramètres qu'il est difficile de quantifier à l'avance. Les membres à connecter et à déconnecter sont dévoilés au fur et à mesure.

Cette situation on-line peut être modélisée naturellement sous la forme du problème de graphe suivant. Il s'agit de construire au fur et à mesure une structure couvrante (un *arbre*, bâti sur le graphe sous-jacent) dont la qualité est satisfaisante. Ce dernier point est particulièrement important. En effet, lorsque les membres communiquent via leur structure dédiée, la *qualité de service* offerte (notamment en termes de *délais* d'acheminement des données point à point) doit être suffisante. Nous traduisons cette exigence de qualité sous la forme du problème d'optimisation suivant. Nous devons maintenir tout au long de la réunion un arbre qui minimise la distance maximum et moyenne entre les membres du groupe (correspondant au temps de communication maximum et moyen entre les membres).

Pour résoudre ce problème, nous proposons deux modèles d'interconnexion (tous deux de type on-line) pour prendre en compte l'ajout et le retrait des membres. Le premier modèle interdit les modifications profondes dans l'arbre courant : l'ajout d'un nouveau membre entraîne au plus l'ajout d'un chemin entre celui-ci et l'arbre courant (l'arbre est incrémenté) ; le retrait d'un membre entraîne au plus l'élagage des branches de l'arbre courant devenues inutiles (l'arbre est décrémenté). Nous évaluons alors la qualité de nos algorithmes en termes de *rapport de compétitivité* pour la distance maximum et moyenne entre les membres du groupe (le rapport de compétitivité mesure le rapport entre la performance de l'algorithme on-line évalué et la performance optimale en termes de diamètre ou de distance moyenne). Nous montrons également que dans certains cas, il n'existe pas d'algorithme on-line sans reconstruction dont le rapport de compétitivité (aussi bien pour la distance maximum que moyenne) est constant. Ces résultats négatifs nous ont amenés à relâcher les contraintes de notre modèle en autorisant les reconstructions. Ainsi, le second modèle autorise les modifications en profondeur dans l'arbre courant : l'ajout ou le retrait d'un membre peut impliquer une restructuration des liens internes de l'arbre. Nous proposons alors des algorithmes minimisant le nombre d'étapes où ont lieu ces restructurations, celles-ci étant très perturbantes puisqu'elles nécessitent (entre autres) la mise à jour des tables de routages des membres du groupe. Nous prouvons alors que nos algorithmes induisent (sous certaines conditions) un nombre au plus logarithmique d'étapes où ont lieu des reconstructions. Nous montrons également dans plusieurs cas l'optimalité (en ordre de grandeur) de ces résultats en exhibant une situation dans laquelle tout algorithme induit un nombre au moins logarithmique d'étapes où ont lieu des reconstructions.

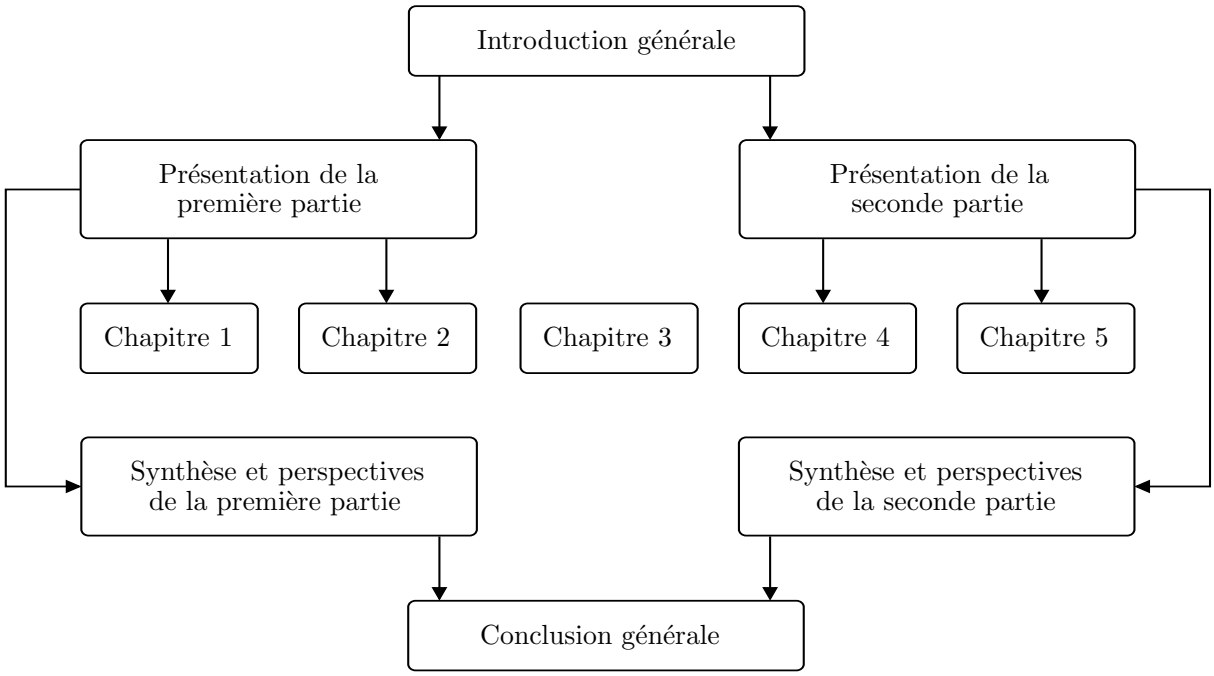
Ordonnancements on-line. La seconde partie de la thèse propose des algorithmes d'ordonnancement on-line pour résoudre le problème suivant. On se place maintenant au niveau du lien dans un réseau de communications. On considère que ce lien est composé de k sous-liens de capacité identique (par exemple une fibre optique dans laquelle k fréquences indépendantes peuvent être utilisées simultanément). Ce lien est géré par un opérateur qui le loue à des utilisateurs. Lorsqu'un utilisateur veut réserver le lien (pour transférer des données), l'opérateur doit choisir sur quel sous-canal l'ordonnancer. Lorsque trop de requêtes ont lieu simultanément, l'opérateur doit choisir lesquelles satisfaire ou rejeter. Nous nous concentrons sur la version du problème où l'opérateur ne connaît

pas à l'avance les requêtes des clients, révélées au fur et à mesure. Cette modélisation on-line est particulièrement adaptée à de nombreux cas réels où l'opérateur doit gérer les requêtes en temps réel, lorsque celles-ci sont révélées au fur et à mesure.

À nouveau, il s'agit d'un problème on-line, que nous modélisons de la manière suivante. On considère un système de $k \geq 1$ machines parallèles identiques (les k sous-canaux du lien). Une tâche Γ est définie par un triplet $\Gamma = (l, r, p)$, où l est son bord gauche, r son bord droit et $p \leq r - l$ sa longueur. Lorsqu'une nouvelle tâche est révélée, un algorithme on-line peut la rejeter ou l'ordonnancer sur une des k machines sur un intervalle de taille p compris entre l et r. Dans ce cas, toutes les tâches déjà ordonnancées sur cette machine et intersectant la nouvelle tâche sont interrompues.

Nous avons choisi d'étudier les deux paramètres suivants : le *poids* (la somme des longueurs des tâches ordonnancées) et la *taille* (le nombre de tâches ordonnancées). Pour évaluer la qualité d'un algorithme, nous utilisons à nouveau la notion de rapport de compétitivité. Aussi bien pour le problème de la maximisation de la taille (correspondant nombre maximum de requêtes d'utilisateur satisfaites) que pour celui de la maximisation du poids (correspondant à l'utilisation maximum des ressources et donc au profit maximum de l'opérateur), nous proposons des algorithmes dont le rapport de compétitivité est constant dans de nombreux cas non triviaux. Dans le cas particulier des intervalles (correspondant au cas où $\Gamma = (r, d, p)$ avec $p = d - r$), nous obtenons des garanties *simultanées* sur les critères poids et taille, aboutissant à un algorithme on-line *bicritère* dont les rapports de compétitivité pour la *taille* et pour le *poids* sont simultanément constants.

Plan de lecture du mémoire. L'ordre de lecture qu'il est conseillé de suivre est celui dans lequel le manuscrit est présenté. Toutefois, certains chapitres étant indépendants les uns des autres, il est possible de lire tout ou partie de ce mémoire dans n'importe quel ordre respectant les contraintes de précédences décrites dans le graphe ci-dessous.



Première partie

Groupes dynamiques dans un graphe

Présentation de la première partie

Définitions et notations

Dans toute la première partie de cette thèse, nous représenterons un réseau de communication par un graphe $G = (V, E, w)$ non orienté, connexe et valué. V est l'ensemble des *sommets* (représentant les nœuds du réseau), E l'ensemble des *arêtes* (représentant l'ensemble des liens physiques du réseau) et w une fonction qui, à chaque arête $e \in E$, associe un *poids* $w(e)$ strictement positif, (représentant le temps de transmission à travers le lien physique modélisé par l'arête e). Pour tous sommets u et v appartenant à V, on note $d_G(u, v)$ la *distance* entre u et v dans G, c'est-à-dire la somme des poids des arêtes d'un chemin de poids minimum entre u et v dans G.

Groupe initial. Le problème que nous étudions est le suivant. Le graphe G et un *groupe initial* M_0 de membres sont donnés en entrée. L'ensemble des membres M_0 est un sous-ensemble non vide des sommets de G (c'est-à-dire que l'on a $M_0 \subseteq V$ et $M_0 \neq \emptyset$). Par exemple, si nous considérons une réunion de membres sur un réseau (représenté par G), ce groupe initial M_0 (de taille $m_0 = |M_0|$) représente l'ensemble des participants présents dès le début de la réunion. Notre but, dans un premier temps, est de connecter ces sommets entre eux par une structure de connexion, bâtie sur le graphe G sous-jacent.

Partie dynamique. Ensuite arrive la partie dynamique du problème (le cœur de notre étude). Au fur et à mesure que la réunion progresse, de nouveaux membres (des sommets du graphe G) peuvent se joindre à la réunion. Dans ce cas, nous devons modifier notre structure de connexion pour intégrer chaque nouveau membre révélé. À l'inverse, des membres présents dans la réunion peuvent quitter la réunion en cours (des sommets appartenant au groupe courant). Dans ce cas, nous devons modifier notre structure de connexion chaque fois qu'un membre souhaite quitter la structure. Nous supposons que les membres à ajouter (resp. retirer) ne sont pas connus à l'avance. Nous nous plaçons donc dans le cadre d'une étude *on-line* (nous précisons cette notion dans la section suivante). Soulignons que le graphe G est connu dès le départ dans sa totalité et est statique (c'est-à-dire que ni les sommets, ni les arêtes de G ne peuvent apparaître ou disparaître) ; seul l'appartenance d'un sommet au groupe courant peut changer.

Séquences de requêtes. Nous allons maintenant introduire les notations qui vont nous permettre de modéliser cette situation. Toutes les définitions et notations que nous donnons dans ce chapitre sont valables pour toute la première partie de la thèse. Pour tout $i \geq 1$, nous notons $rq_i = (x_i, u_i)$ la $i^{\text{ème}}$ *requête* on-line, avec $x_i \in \{\text{ajout, retrait}\}$ et $u_i \in V$. Pour tout $i \geq 1$, nous notons M_i le $i^{\text{ème}}$ groupe ($M_i \subseteq V$), et $m_i = |M_i|$ sa taille.

Si $x_i =$ ajout, alors rq_i est une requête d'ajout. On a alors $u_i \in V \backslash M_{i-1}$ et $M_i = M_{i-1} \cup \{u_i\}$.

Si $x_i =$ retrait, alors rq_i est une requête de retrait. On a alors $u_i \in M_{i-1} \subseteq V$ et $M_i = M_{i-1} \backslash \{u_i\}$.

Pour tout $i \geq 1$, on note $rq_1, \ldots, rq_i, \ldots$ la séquence de requêtes on-line, où rq_i est la $i^{\text{ème}}$ requête. Par la suite, nous étudierons particulièrement les séquences de requêtes on-line composées uniquement d'ajouts ainsi celles composées uniquement de retraits. Afin de simplifier la lecture, une séquence de requêtes sera parfois représentée par la suite des groupes que la séquence de requêtes induit, c'est-à-dire que nous noterons :

$M_0 \subset \cdots \subset M_i$ une séquence composée uniquement d'ajouts,

$M_0 \supset \cdots \supset M_i$ une séquence composée uniquement de retraits (avec $0 \leq i \leq m_0 - 1$) et

$M_0, \ldots, M_i$ une séquence composée d'ajouts et de retraits mêlés.

Le modèle *sans reconstruction*

Pour définir précisément la nature de notre problème, nous précisons notre premier modèle d'étude, que nous appelons *sans reconstruction*, défini par l'ensemble des contraintes suivantes.

Contrainte *on-line*. La première contrainte que nous imposons (et qui constitue le cœur de notre étude) porte sur le mode de révélation de la séquence de requêtes (d'ajouts ou de retraits). En effet, nous nous plaçons ici dans le cas où cette séquence est révélée au fur et à mesure, élément par élément, sans aucune connaissance du futur. Ce modèle est connu sous le nom de on-line (voir [18, 25] pour des références sur les algorithmes on-line en général).

Contrainte *arbre*. La deuxième contrainte que nous imposons concerne la nature de la structure que nous devons construire à chaque étape pour connecter l'ensemble des sommets appartenant au groupe courant. Pour toute étape $i \geq 0$, nous imposons que la structure $T_i = (V_i, E_i)$ couvrant le groupe M_i ($M_i \subseteq V_i \subseteq V$) soit un *arbre* élagué, c'est-à-dire que chaque feuille de T_i est un sommet du groupe courant M_i (il n'y a donc pas de branches inutiles, dites *branches mortes*, pour la couverture de M_i dans T_i). Nous imposons que notre structure soit un arbre dans le but de simplifier les mécanismes de routage et de duplication des informations à faire circuler entre les membres du groupe. En effet, il n'y a qu'une seule route possible entre deux sommets d'un arbre, le routage est donc trivial. De plus, l'absence de cycle simplifie, entre autres, la diffusion d'un message d'un membre vers tous les autres : cette diffusion peut être faite de manière naturelle, par inondation, sans contrôle ni surplus de trafic.

Contrainte *emboîtement*. La troisième contrainte restreint ce que nous nous autorisons à changer dans l'arbre courant pour ajouter (resp. retirer) un sommet. À chaque étape i, nous imposons :

- Si i est une étape d'ajout, alors le nouvel arbre $T_i = (V_i, E_i)$ devra contenir l'arbre $T_{i-1} = (V_{i-1}, E_{i-1})$ de l'étape précédente (c'est-à-dire que l'on doit avoir $V_{i-1} \subseteq V_i$ et $E_{i-1} \subseteq E_i$). Cela signifie que nous interdisons les changements dans la partie déjà construite de l'arbre, car ces changements sont coûteux et perturbateurs pour les communications en cours entre membres du groupe courant. La connexion d'un nouveau sommet u devra donc se faire par l'ajout d'un chemin entre u et l'arbre T_{i-1}.
- Si i est une étape de retrait, alors le nouvel arbre $T_i = (V_i, E_i)$ devra être contenu dans l'arbre $T_{i-1} = (V_{i-1}, E_{i-1})$ de l'étape précédente (c'est-à-dire que l'on doit avoir $V_i \subseteq V_{i-1}$ et $E_i \subseteq E_{i-1}$). Cela signifie que nous interdisons les changements (coûteux et perturbateurs) dans

la partie de l'arbre couvrant le précédent groupe M_{i-1}. La déconnexion d'un sommet devra donc se faire par un simple élagage de l'arbre.

Nous soulignons que dans le cas particulier d'une séquence d'ajouts (resp. de retraits) $M_0 \subset \cdots \subset M_i$ (resp. $M_0 \supset \cdots \supset M_i$), la suite d'arbres $T_0 = (V_0, E_0), \ldots, T_i = (V_i, E_i)$ retournée doit satisfaire $V_0 \subseteq \cdots \subseteq V_i$ et $E_0 \subseteq \cdots \subseteq E_i$ (resp. $V_0 \supseteq \cdots \supseteq V_i$ et $E_0 \supseteq \cdots \supseteq E_i$), c'est-à-dire que nous ne sommes autoriser qu'à incrémenter (resp. décrémenter) l'arbre courant.

Si nous n'imposons que ces trois contraintes, résoudre le problème que nous nous posons est simple, mais peu intéressant. En effet, la difficulté (et l'intérêt) va venir du fait que nous ne voulons pas construire une suite d'arbres quelconques, mais une suite d'arbres offrant une certaine qualité de communication aux membres. Cela nécessite l'évaluation de la qualité d'un arbre en fonction de critères quantifiables.

Les critères d'évaluation. Nous allons nous intéresser à deux critères d'évaluation : le diamètre du groupe courant dans l'arbre et la distance moyenne entre les membres du groupe courant dans l'arbre. En effet, si le poids d'un lien (une arête du graphe G) représente le temps de transmission d'une information à travers ce lien, alors donner une garantie sur le diamètre (resp. la distance moyenne entre les sommets du groupe) permet d'assurer une certaine qualité de communication aux membres utilisateurs de la structure de communication en termes de latence maximum (resp. moyenne).

Diamètre. Nous définissons le diamètre d'un groupe de la manière suivante.

Définition 1 (Diamètre d'un groupe) *Soient $G = (V, E, w)$ un graphe et $M \subseteq V$ un groupe. Le* diamètre *de M dans G est :*

$$D_G(M) = \max\{d_G(u, v) : u, v \in M\}$$

Si on veut optimiser la latence maximum dans un groupe M, on doit minimiser le diamètre du groupe dans l'arbre couvrant M que l'on a construit. Étant donné que nous nous plaçons dans un contexte on-line, nous n'avons aucune connaissance du futur. Chaque nouvelle requête est donc potentiellement la dernière. Dans ces conditions, nous devons minimiser le diamètre de l'arbre courant *à chaque étape* de construction. Dans le but d'évaluer la qualité d'un algorithme retournant une séquence d'arbres, nous définissons maintenant le rapport de compétitivité d'un algorithme pour le diamètre. Le rapport de compétitivité est une notion standard de l'algorithmique on-line et consiste à comparer la qualité de la solution construite avec la qualité de la meilleure solution off-line possible (c'est-à-dire la meilleure solution que l'on aurait pu construire si toutes les données du problème nous avaient été révélées en une fois, et non au fur et à mesure).

Définition 2 (Rapport de compétitivité pour le diamètre) *Soient $G = (V, E, w)$ un graphe, M_0 un groupe initial et $rq_1, \ldots, rq_i, \ldots$ une séquence de requêtes on-line quelconques. Soit A un algorithme retournant une séquence d'arbres $T_0, \ldots, T_i, \ldots$ couvrant les groupes $M_0, \ldots, M_i, \ldots$. Soient $T_0^*, \ldots, T_i^*, \ldots$ les arbres couvrant les groupes $M_0, \ldots, M_i, \ldots$ tels que $D_{T_i^*}(M_i) = \min\{D_{T_i'}(M_i) : T_i'$ est un arbre couvrant $M_i\}$. L'algorithme A a un rapport de compétitivité c (ou est $c-$compétitif) pour le diamètre si on a :*

$$\forall i, \quad D_{T_i}(M_i) \leq c \cdot D_{T_i^*}(M_i)$$

L'interprétation du rapport de compétitivité est la suivante. Un algorithme A est $c-$compétitif si à chaque étape i, il construit un arbre T_i couvrant M_i induisant un diamètre au plus c fois plus grand que le diamètre de M_i dans le meilleur arbre possible (noté T_i^*).

Distance moyenne. Minimiser la distance moyenne entre les sommets d'un groupe dans un arbre revient à minimiser la somme des distances entre les sommets du groupe dans l'arbre, car il suffit de diviser la somme des distances par le nombre de couples possibles entre membres du groupe pour obtenir la distance moyenne. À partir de maintenant, nous allons donc nous intéresser à la *somme des distances* entre les sommets d'un groupe.

Définition 3 (Somme des distances) *Soient $G = (V, E, w)$ un graphe quelconque et $M \subseteq V$ un groupe. On définit la* somme des distances *entre les sommets de M dans G par :*

$$C_G(M) = \sum_{u,v \in M} d_G(u, v)$$

Si on veut optimiser la latence moyenne dans un groupe M, on doit minimiser la somme des distances du groupe dans l'arbre couvrant M que l'on a construit. Nous définissons maintenant le rapport de compétitivité d'un algorithme pour la somme des distances (de manière similaire à la définition 2).

Définition 4 (Rapport de compétitivité pour la somme des distances)
Soient $G = (V, E, w)$ un graphe, M_0 un groupe initial et $rq_1, \ldots, rq_i, \ldots$ une séquence de requêtes on-line quelconques. Soit A un algorithme retournant une séquence d'arbres $T_0, \ldots, T_i, \ldots$ couvrant les groupes $M_0, \ldots, M_i, \ldots$. Soient $T_0^*, \ldots, T_i^*, \ldots$ les arbres couvrant les groupes $M_0, \ldots, M_i, \ldots$ tels que $C_{T_i^*}(M_i) = \min\{C_{T_i'}(M_i) : T_i'$ est un arbre couvrant $M_i\}$. L'algorithme A a un rapport de compétitivité c (ou est $c-$compétitif) pour la somme des distances si on a :

$$\forall i, \quad C_{T_i}(M_i) \leq c \cdot C_{T_i^*}(M_i)$$

Le modèle *avec reconstructions*

Lorsque nous imposons à un algorithme de respecter les contraintes *on-line*, *arbre* et *emboîtement* (correspondant au modèle *sans reconstruction* décrit dans la section précédente), nous avons prouvé qu'il n'est pas toujours possible (aussi bien pour le diamètre que pour la somme des distances) d'obtenir une qualité de connexion satisfaisante, c'est-à-dire un rapport de compétitivité constant (voir les théorèmes 4 et 11 des sections 1.3.1 et 2.2.1). Si nous voulons obtenir des résultats satisfaisants en termes de qualité de connexion, nous devons donc définir un nouveau modèle d'étude plus souple, en relâchant au moins une des contraintes que nous nous sommes fixées. Nous avons choisi de relâcher la contrainte *emboîtement* pour les raisons suivantes. La contrainte *on-line* constitue l'intérêt principal de notre étude, la relâcher en priorité diminuerait donc énormément la portée de nos résultats, quant à la contrainte *arbre*, elle nous paraît totalement justifiée, puisqu'elle garantit une structure de communication simple et efficace. Pour que notre nouveau modèle soit intéressant, nous devons

- intégrer dans le modèle une contrainte garantissant un rapport de compétitivité constant pour le diamètre (resp. la somme des distances) et
- donner des garanties sur le nombre de reconstructions induites par le relâchement de la contrainte *emboîtement*. En effet, si nous n'imposons aucune restriction sur le nombre de

reconstructions, une solution triviale à notre problème consiste à détruire totalement, puis reconstruire à chaque étape un arbre couvrant le groupe courant, et ainsi obtenir un rapport de compétitivité constant pour le diamètre (resp. la somme des distances). Mais cette stratégie n'est pas intéressante puisque dans un modèle où les données sont révélées de manière on-line, l'intérêt est de trouver des algorithmes qui construisent une solution au fur et à mesure. Remettre en cause la totalité de la solution construite à chaque étape reviendrait à considérer une suite de problèmes off-line (méthode beaucoup trop coûteuse et perturbatrice pour les communications en cours entre membres du groupe courant).

Nous allons maintenant préciser les contraintes qui doivent être respectées et qui définissent le modèle *avec reconstructions*.

Contraintes *on-line* et *arbre*. Ces deux contraintes doivent à nouveau être respectées dans le modèle *avec reconstructions* et sont définies de la même façon que dans la section précédente.

Contrainte *qualité pour le diamètre* (resp. *la somme des distances*). L'idée est d'intégrer sous forme de *contrainte* la garantie de qualité de connexion que nous donnait le rapport de compétitivité pour le diamètre (resp. la somme des distances) défini pour le modèle *sans reconstruction*.

Définition 5 (Contrainte *qualité pour le diamètre* (resp. *la somme des distances*))
Soit $c \geq 1$ une constante quelconque, représentant le niveau de qualité désiré pour le critère diamètre (resp. somme des distances). Soient $G = (V, E, w)$ un graphe, M_0 un groupe initial et $rq_1, \ldots, rq_i, \ldots$ une séquence de requêtes on-line quelconques. Soit A *un algorithme retournant une séquence d'arbres $T_0, \ldots, T_i, \ldots$ couvrant les groupes $M_0, \ldots, M_i, \ldots$. Soient $T_0^*, \ldots, T_i^*, \ldots$ les arbres couvrant les groupes $M_0, \ldots, M_i, \ldots$ tels que $D_{T_i^*}(M_i) = \min\{D_{T_i'}(M_i) : T_i' \text{ est un arbre couvrant } M_i\}$ (resp. $C_{T_i^*}(M_i) = \min\{C_{T_i'}(M_i) : T_i' \text{ est un arbre couvrant } M_i\}$).*

L'algorithme A *satisfait la contrainte* qualité pour le diamètre *(resp.* la somme des distances*) avec un* niveau *c si on a :*

$$\forall i, \quad D_{T_i}(M_i) \leq c \cdot D_{T_i^*}(M_i) \qquad \textit{(resp. } C_{T_i}(M_i) \leq c \cdot C_{T_i^*}(M_i)\textit{)}$$

Remarques. L'interprétation de la contrainte *qualité* est la suivante. Un algorithme A respecte la contrainte *qualité pour le diamètre* (resp. *la somme des distances*) avec un niveau c (où c est une constante) si à chaque étape i, il construit un arbre T_i couvrant M_i induisant un diamètre (resp. une somme des distances) au plus c fois plus grand(e) que le diamètre (resp. la somme des distances entre les sommets) de M_i dans le meilleur arbre possible. Nous soulignons que nous n'imposons pas nécessairement que les contraintes *qualité pour le diamètre* et *qualité pour la somme des distances* soient vérifiées simultanément. En effet, à part pour certains sous cas (que nous préciserons lors de la synthèse des résultats dans le chapitre 5), nous nous sommes restreints dans cette thèse à l'étude d'un critère à la fois.

Si nous n'imposons que les trois contraintes *on-line*, *arbre* et *qualité pour le diamètre* (resp. *la somme des distances*), résoudre le problème que nous nous posons est simple, mais peu intéressant (il suffit en effet de reconstruire totalement l'arbre courant à chaque nouvelle étape). Dans le modèle *avec reconstructions*, la difficulté (et l'intérêt) va venir du fait que nous voulons minimiser les perturbations dans l'arbre courant. Cela implique qu'un algorithme va maintenant être évalué en fonction des perturbations qu'il va induire dans l'arbre.

Quantification et évaluation des perturbations dues aux reconstructions. Afin de quantifier précisément ces perturbations, nous avons besoin des définitions suivantes. Dans un arbre T couvrant un groupe $M \subseteq V$, nous pouvons distinguer deux types de sommets. D'une part, il y a les sommets qui ont à prendre des décisions dans le transfert des informations. Par exemple, les sommets de degré 3 ou plus dans l'arbre T, lorsqu'ils reçoivent une information d'un membre du groupe M, doivent router cette information sur le lien approprié. De plus, si l'information est transmise à plusieurs membres de M, elle doit être dupliquée. Nous soulignons qu'un tel sommet n'est pas nécessairement un sommet appartenant au groupe M. Nous appelons ces sommets des *sommets de connexion* (voir la définition 6). Nous incluons également dans cette catégorie les sommets appartenant au groupe M (même s'ils sont de degré inférieur à 3 dans T). La seconde catégorie de sommets est constituée des sommets de degré 2 dans l'arbre T n'appartenant pas au groupe M. En effet, lorsqu'un tel sommet reçoit une information sur un de ses liens, la seule opération qu'il doit faire est de la renvoyer sur son autre lien. Aucune décision n'a à être prise. Nous appelons ces sommets les *relais* (voir la définition 7).

Définition 6 (Sommet de connexion) *Soit $G = (V, E, w)$ un graphe et $M \subseteq V$ un groupe. Soit $T = (V_T, E_T, w)$ un arbre élagué couvrant M. $u \in V_T$ est un* sommet de connexion *de T si on a :*

- *$u \in M$*

ou

- *u est un sommet de degré au moins 3 dans l'arbre T.*

Définition 7 (Relais) *Soit $G = (V, E, w)$ un graphe et $M \subseteq V$ un groupe. Soit $T = (V_T, E_T, w)$ un arbre élagué couvrant M. $u \in V_T$ est un* relais *de T si on a :*

- *$u \notin M$*

et

- *u est un sommet de degré 2 dans l'arbre T.*

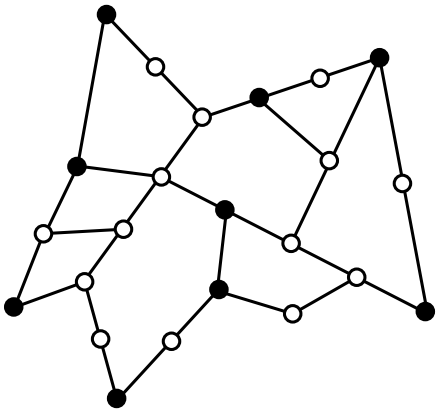
Remarques : nous soulignons que tous les sommets $u \in V_T$ de degré 1 dans l'arbre $T = (V_T, E_T, w)$ sont des membres du groupe M (car T est un arbre élagué), donc des sommets de connexion. Cela signifie que l'ensemble des sommets de connexion et l'ensemble des relais forment une partition de V_T. La figure 1(c) illustre les notions de sommets de connexion (les sommets entourés en pointillé) et de relais (les sommets non entourés).

Nous pouvons maintenant voir un arbre élagué comme un ensemble de sommets de connexion reliés par des chemins (tunnels) composés de relais. Cela nous amène à définir un *arbre de connexion*, qui est une simplification de l'arbre réel, obtenue en remplaçant chaque chemin de relais par une arête.

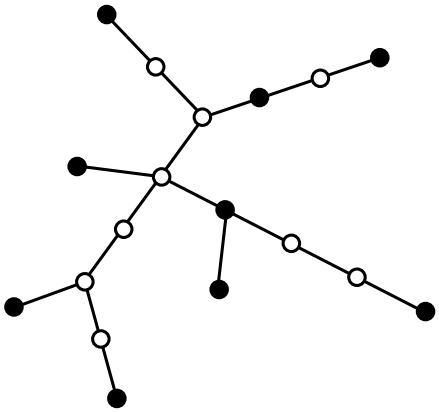
Définition 8 (Arbre de connexion) *Soit $G = (V, E, w)$ un graphe et $M \subseteq V$ un groupe. Soit $T = (V_T, E_T, w)$ un arbre élagué couvrant M. On note $T^c = (V_{T^c}, E_{T^c})$ l'*arbre de connexion associé *à T obtenu en remplaçant chaque chemin entre deux sommets de connexion composé uniquement de relais dans T par une arête non valuée.*

La figure 1 illustre la notion d'arbre de connexion. La figure 1(a) donne l'exemple d'un groupe M de membres dans un graphe devant être connectés (les membres sont les sommets noirs). La figure 1(b) donne un exemple d'arbre élagué T couvrant le même groupe M. Enfin, les figures 1(c) et 1(d) représentent respectivement les sommets de connexion de l'arbre T et l'arbre de connexion T^c associé à T.

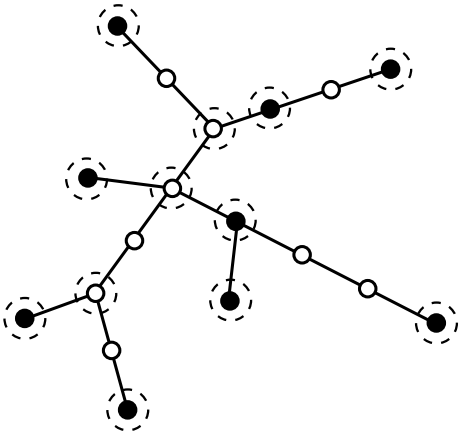
(a) Un graphe G



(b) Un arbre T couvrant M



(c) Les sommets de connexion de T



(d) L'arbre T^c associé à T

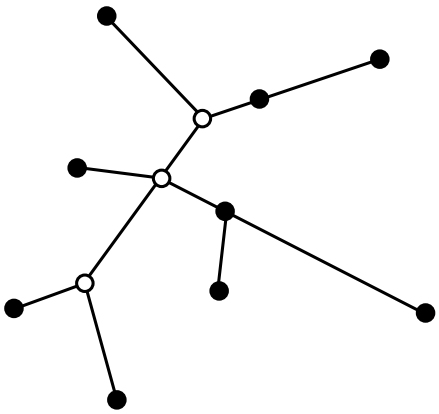


FIG. 1 – Illustration d'un arbre de connexion : un graphe (a), un arbre couvrant (b), les sommets de connexion (entourés en pointillé) (c) et l'arbre de connexion (d). Les sommets noirs sont les membres du groupe M.

La définition d'un *arbre de connexion* nous permet de définir maintenant ce que nous appelons un *changement élémentaire*.

Définition 9 (Changement élémentaire) *Soit $rq_0, \ldots, rq_i$ une séquence de requêtes on-line quelconques. Soit* A *un algorithme respectant les contraintes* on-line *et* arbre. *À l'étape k ($1 \leq k \leq i$), l'algorithme* A *construit un arbre T_k à partir de T_{k-1} (avec respectivement T_k^c et T_{k-1}^c leur arbre de connexion associé). On comptabilise un* changement élémentaire *pour :*

- *L'ajout d'une arête dans l'arbre de connexion T_{k-1}^c*

ou

- *Le retrait d'une arête dans l'arbre de connexion T_{k-1}^c.*

Par exemple, la figure 2 illustre la notion de *changement élémentaire* dans le scénario d'ajout suivant. D'abord, le nouveau membre à ajouter u_k est révélé (la figure 2(a) représente le graphe sous-jacent et la figure 2(b) l'arbre de connexion courant T_{k-1}^c). Ensuite, une arête est retirée de T_{k-1}^c (voir l'arête pointillée de la figure 2(c)). Enfin, trois nouvelles arêtes sont ajoutées à T_{k-1}^c pour connecter u_k (voir les arêtes pointillées de la figure 2(d)) et obtenir le nouvel arbre de connexion T_k^c (voir figure 2(e)). Au final, à l'étape d'ajout k, $1 + 3 = 4$ changements élémentaires ont eu lieu.

La figure 3 illustre la notion de *changement élémentaire* dans le scénario de retrait suivant. D'abord, le membre à retirer u_k est révélé (la figure 3(a) représente le graphe sous-jacent et la figure 3(b) l'arbre de connexion courant T_{k-1}^c). Ensuite, trois arêtes sont retirées de T_{k-1}^c (voir les arêtes pointillées de la figure 3(c)). Enfin, une nouvelle arête est ajoutée à T_{k-1}^c (voir les arêtes pointillées de la figure 3(d)) pour obtenir le nouvel arbre de connexion T_k^c (voir figure 3(e)). Au final, à l'étape de retrait k, $3 + 1 = 4$ changements élémentaires ont eu lieu.

Un changement élémentaire induit un coût (en temps, en coût réseau). Le nombre de changements élémentaires doit donc être minimisé. Nous reviendrons plus précisément dans la suite de cette section sur ce que nous entendons par "minimiser le nombre de changements élémentaires" (voir définition 11).

Remarque : nous avons défini le nombre de changements élémentaires dans l'arbre de connexion T_k^c plutôt que dans l'arbre T_k auquel il est associé pour la raison suivante. Si nous prenons l'exemple d'un groupe M_k composé de deux sommets u et v. Supposons que le chemin reliant u et v dans l'arbre T_k couvrant M_k soit composé d'un nombre d'arêtes arbitrairement grand. Pour déconnecter u de v, on doit casser la connexion de u à v. Si nous avions défini le nombre de changements élémentaires directement dans l'arbre T_k, celui-ci aurait été dans ce cas arbitrairement grand (et sans réel rapport avec le coût réseau associé). Grâce à la définition de l'arbre de connexion T_k^c, cette déconnexion sera comptabilisée comme un seul changement élémentaire (puisque dans T_k^c, le chemin reliant u et v est remplacé par une seule arête). Le coût réseau associé à cette déconnexion est ainsi plus justement représenté.

Afin de compléter l'évaluation des perturbations induites par le modèle *avec reconstructions*, nous définissons maintenant ce que nous appelons une *étape critique* (c'est-à-dire une étape induisant de fortes perturbations dans l'arbre courant).

Définition 10 (Étape critique) *Soit $rq_0, \ldots, rq_i$ une séquence de requêtes on-line quelconque. Soit* A *un algorithme respectant les contraintes* on-line *et* arbre *retournant une séquence d'arbres $T_0 = (V_0, E_0), \ldots, T_i = (V_i, E_i)$.*

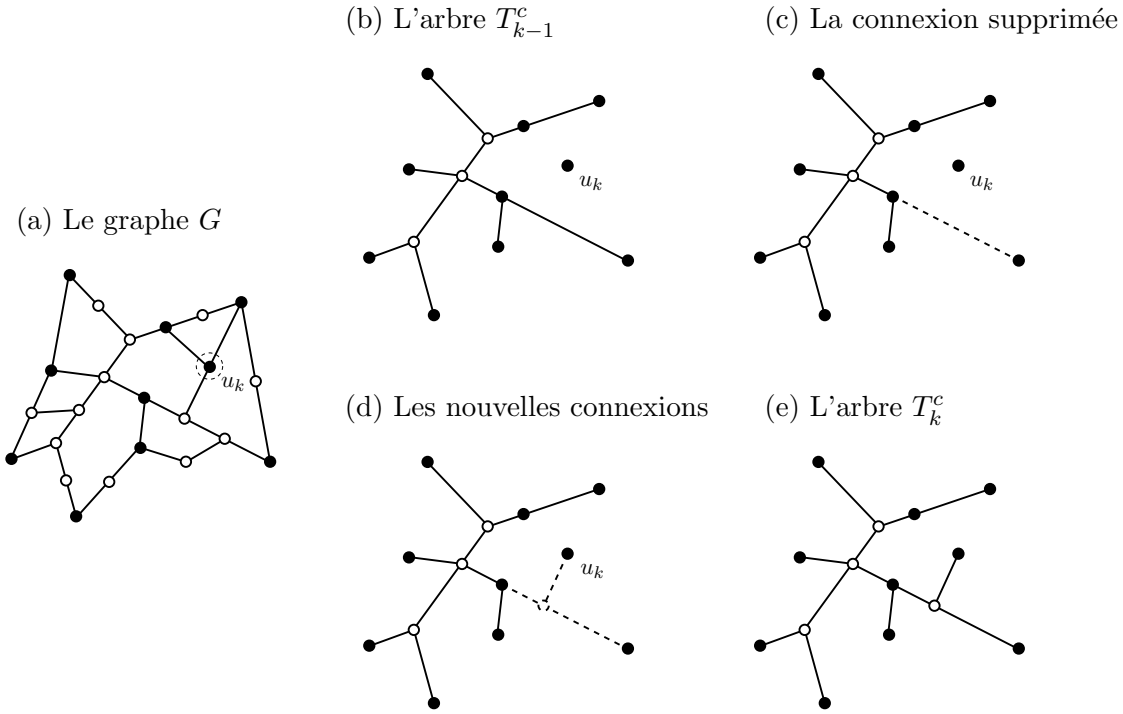


FIG. 2 – Illustration des changements élémentaires pour une étape d'ajout. Une arête en pointillée correspond à un changement élémentaire. Les sommets noirs sont les membres du groupe courant.

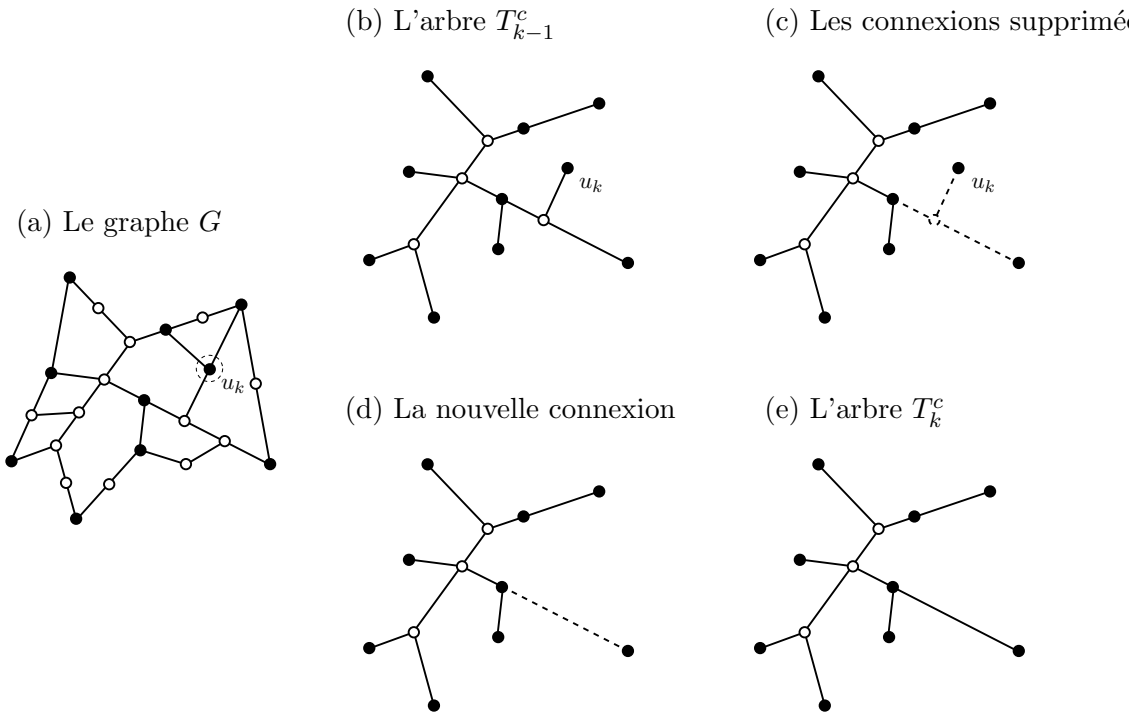


FIG. 3 – Illustration des changements élémentaires pour une étape de retrait. Une arête en pointillée correspond à un changement élémentaire. Les sommets noirs sont les membres du groupe courant.

– *Si rq_k ($1 \leq k \leq i$) est une requête d'ajout, alors l'étape k est une* étape critique *si on a :*

$$E_{k-1} \not\subseteq E_k$$

– *Si rq_k ($1 \leq k \leq i$) est une requête de retrait, alors l'étape k est une* étape critique *si on a :*

$$E_k \not\subseteq E_{k-1}$$

La figure 4(a) (resp. 5(a)) montre l'exemple d'une étape d'ajout (resp. de retrait) non critique, car l'arbre T_k est obtenu par l'ajout (resp. le retrait) d'un chemin entre le sommet u_k et l'arbre T_{k-1} ; on a donc bien $E_{k-1} \subseteq E_k$ (resp. $E_k \subseteq E_{k-1}$). La figure 4(b) (resp. 5(b)) montre une autre possibilité d'ajout (resp. de retrait) : u_k est ajouté (resp. retiré) et une partie de l'arbre T_{k-1} est modifiée (par exemple pour respecter la contrainte *qualité*, ou pour n'importe quelle autre raison). Dans ce cas, on a $E_{k-1} \not\subseteq E_k$ (resp. $E_k \not\subseteq E_{k-1}$) : il s'agit donc d'une étape critique.

Notons qu'une étape est critique lorsqu'elle induit le non-respect de la contrainte *emboîtement*. Nous distinguons les étapes critiques des autres car elles génèrent d'importants changements dans l'arbre courant. En effet, les routes entre membres déjà présents dans le groupe courant doivent être changées après une étape critique. Toutes les tables de routage des sommets de connexion peuvent potentiellement être modifiées. Non seulement cette mise à jour génère un surplus de trafic important, mais perturbe également les communications entre les membres du groupe courant déjà en cours. Le nombre d'étape critique doit donc être minimisé. Nous reviendrons plus précisément dans la suite de cette section sur ce que nous entendons par "minimiser le nombre d'étapes critiques" (voir définition 11).

D'autre part, un ajout (resp. retrait) simple (c'est-à-dire non critique) nécessite uniquement l'ajout (resp. le retrait) d'un chemin dans l'arbre courant. Il s'agit donc de changements locaux. La mise à jour des tables de routage ne nécessite que la diffusion de l'identité du nouveau membre (resp. l'information de disparition du membre retiré) dans l'arbre. Il n'y a donc pas de re-routage nécessaire entre les membres du groupe courant.

Nous définissons maintenant précisément les deux quantités à minimiser : le nombre de changements élémentaires moyen par étape, et le nombre d'étapes critiques.

Définition 11 (Nombre de changements élémentaires (resp. d'étapes critiques))
Soit $rq_0, \ldots, rq_i$ une séquence de requêtes on-line quelconques. Soit A *un algorithme respectant les contraintes* on-line *et* arbre *retournant une séquence d'arbres $T_0, \ldots, T_i$. Soient $T_0^c, \ldots, T_i^c$ les arbres de connexion respectivement associés à $T_0, \ldots, T_i$.*

– *Pour tout k ($1 \leq k \leq i$), soit $\sharp CE(T_k^c)$ le nombre de changements élémentaires effectués par l'algorithme* A *à l'étape k. Le* nombre de changements élémentaires moyen par étape *$\sharp CEM(T_0^c, \ldots, T_i^c)$ induit par l'algorithme* A *après i étapes est défini par :*

$$\sharp CEM(T_0^c, \ldots, T_i^c) = \frac{1}{i} \sum_{k=1}^{i} \sharp CE(T_k^c)$$

– *On note le* nombre d'étapes critiques *induit par l'algorithme* A *après i étapes par :*

$$\sharp EC(T_0, \ldots, T_i)$$

(a) L'étape k n'est pas une étape critique (b) L'étape k est une étape critique

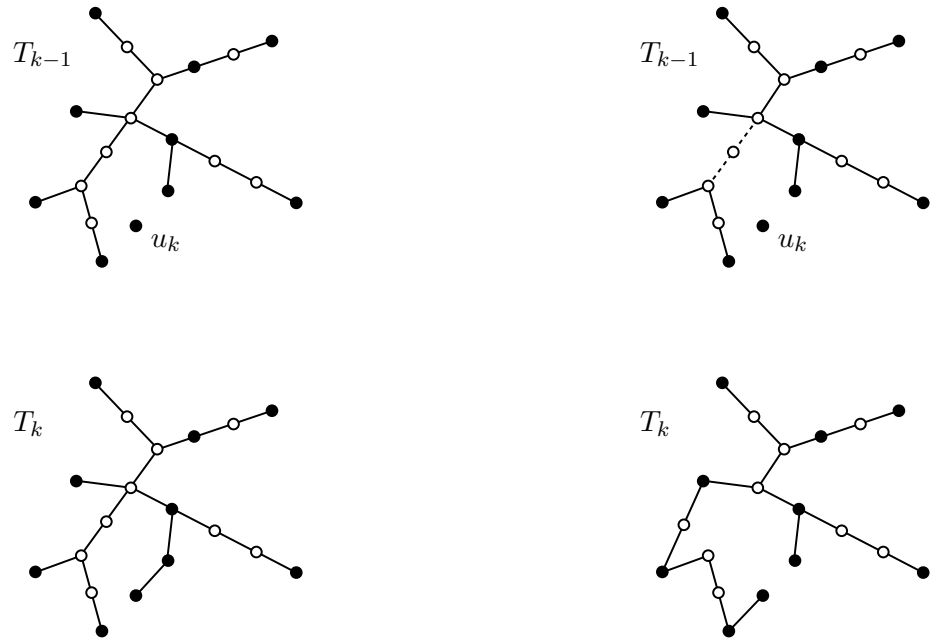


FIG. 4 – Ajout non critique versus ajout critique. Les lignes pointillées sont les arêtes retirées. Les sommets noirs sont les membres du groupe courant.

(a) L'étape k n'est pas une étape critique (b) L'étape k est une étape critique

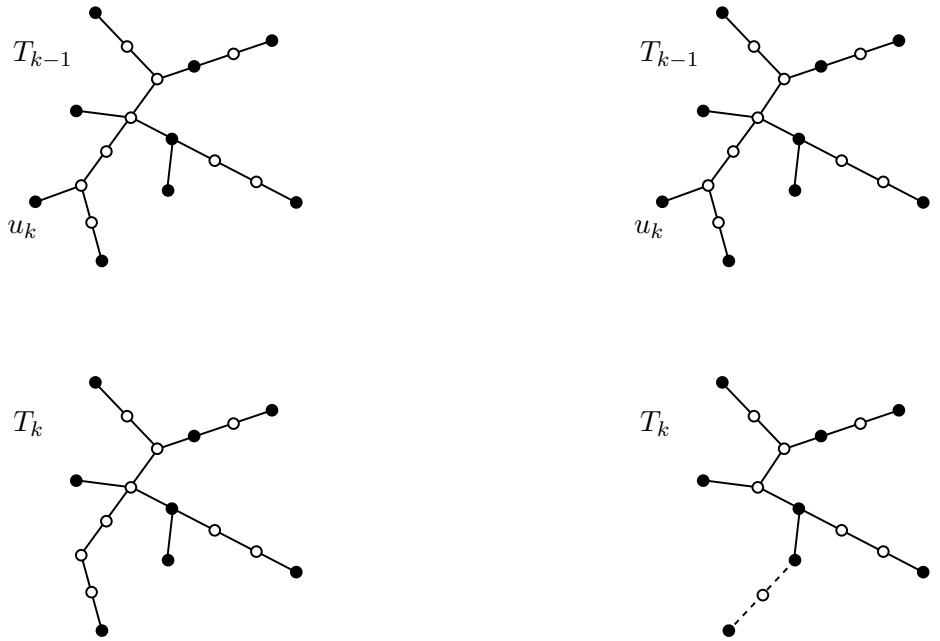


FIG. 5 – Retrait non critique versus retrait critique. Les lignes pointillées sont les arêtes ajoutées. Les sommets noirs sont les membres du groupe courant.

Deux critères d'évaluation. Dans le modèle *avec reconstructions*, les deux quantités à minimiser *simultanément* sont le *nombre de changements élémentaires moyen par étape* et le *nombre d'étapes critiques*. Nous soulignons que minimiser simultanément l'une et l'autre des quantités nous donne des garanties complémentaires sur la qualité d'un algorithme. Un algorithme qui induit un *nombre de changements élémentaires moyen par étape* faible sera économique en termes de coût réseau associé et un algorithme qui induit un *nombre d'étapes critiques* faible sera économique en termes de nombre d'étapes où des mises à jour internes importantes sont à effectuer. La difficulté du modèle *avec reconstructions* va donc venir du fait que les algorithmes que nous allons proposer vont devoir respecter les contraintes *on-line*, *qualité pour le diamètre* (ou *qualité pour la somme des distances*) tout en minimisant simultanément le *nombre de changements élémentaires moyen par étape* et le *nombre d'étapes critiques*. Ces deux dernières quantités seront donc utilisées pour l'évaluation d'un algorithme dans le cadre du modèle *avec reconstructions*.

État de l'art

Le problème que nous étudions a été exploré essentiellement dans sa version cherchant à minimiser le *poids* de l'arbre courant (c'est-à-dire la somme des poids des arêtes de l'arbre). La version off-line de ce problème (c'est-à-dire lorsque les données sont intégralement connues dès le départ) est connue sous le nom du problème de l'arbre de Steiner. Étant donné qu'il s'agit d'un problème NP-complet (voir [29] pour des précisions sur la notion de NP-complétude), des algorithmes d'approximations ont été proposés pour résoudre ce problème (par exemple dans [8, 36]).

Dans [65], B. Waxman a été le premier à présenter la version on-line de l'arbre de Steiner. Dans cet article, il a divisé le problème en deux catégories : le modèle dans lequel les changements dans l'arbre courant ne sont pas autorisés d'une part (correspondant au modèle *sans reconstruction*, défini en section), le modèle dans lequel les changements dans l'arbre courant sont autorisés d'autre part (correspondant au modèle *avec reconstructions*, défini en section). Puis, dans [39], M. Imaze et B. Waxman ont proposé un algorithme pour chaque modèle. Pour le modèle *sans reconstruction*, les auteurs se sont limités à l'étude des séquences d'ajouts on-line (de nouveaux membres peuvent intégrer le groupe courant au fur et à mesure, mais ne peuvent pas le quitter). Ils prouvent que l'algorithme qu'ils proposent a un rapport de compétitivité en $O(\log i)$ pour le critère poids (où i est le nombre de sommets ajoutés). Ils montrent également qu'il n'existe pas d'algorithme sans reconstruction dont le rapport de compétitivité est meilleur qu'une fonction en $\Omega(\log i)$. Pour le modèle *avec reconstructions*, les auteurs proposent un algorithme traitant le cas des ajouts et des retraits on-line mêlés. Cette stratégie a un rapport de compétitivité constant pour le critère poids et induit un *nombre de changements élémentaires moyen par étape* (voir définition 9) en $0(\sqrt{i})$. En revanche, les auteurs ne donnent pas de garantie sur le nombre d'*étapes critiques* (voir définition 10). Nous pouvons maintenant diviser les travaux existants en deux parties (comme B. Waxman dans [65]).

D'une part, dans [3, 9, 66] (par exemple), le modèle *sans reconstruction* est considéré. Dans [3], les auteurs donnent une borne inférieure en $\Omega(\frac{\log i}{\log \log i})$ sur le rapport de compétitivité de tout algorithme résolvant le problème de l'arbre de Steiner on-line (donc pour le critère poids) dans un plan Euclidien. Dans [9], les auteurs considèrent le problème de l'arbre de Steiner généralisé : étant donné un ensemble de paires de sommets, il s'agit de trouver un sous-graphe de poids minimum tel que chaque paire de sommet est connectée par le sous-graphe construit. Un algorithme dont le rapport de compétitivité est en $O(\log^2 i)$ est proposé pour la version traitant les ajouts on-line de ce problème. Dans [66], des bornes supérieures et inférieures linéaires en nombre de paires ajoutées sont obtenues pour le problème de l'arbre de Steiner on-line généralisé sur les graphes orientés.

D'autre part, dans [30, 54] (par exemple), le modèle *avec reconstructions* est envisagé. Dans [30], le but est de minimiser simultanément le poids de l'arbre courant et la distance à partir d'un sommet racine vers tous les autres sommets du groupe courant. Seule la version on-line avec ajouts est traitée. Les auteurs proposent alors une stratégie induisant au plus un changement élémentaire par étape et dont le rapport de compétitivité est simultanément en $O(\log i)$ pour le critère poids et constant pour le critère distance à la racine. Dans [1, 47, 54], des méthodes traitant simultanément les ajouts et les retraits on-line pour l'optimisation de plusieurs critères (dont le poids et le diamètre) sont proposées. Ces méthodes sont ensuite évaluées par des simulations. Aucune borne supérieure analytique n'est proposée, ni pour les différents rapports de compétitivité, ni pour le nombre de changements induits. Nous soulignons que dans [30, 39, 54], seuls le nombre de changements élémentaires est pris en compte pour mesurer le niveau de perturbation due aux changements dans l'arbre courant. À notre connaissance, il n'existe pas de travaux antérieurs aux nôtres qui considèrent le nombre d'*étapes critiques* comme critère d'évaluation d'un algorithme.

Quant au critère somme des distances entre membres du groupe, à notre connaissance, seule la version off-line du problème (introduite dans [38]) a déjà été traitée : le problème étant NP-complet (voir [29, 41]), des algorithmes d'approximations ont été proposés (voir [46, 68, 69]).

Plan de la première partie

Dans la première partie de la thèse (les chapitres 1, 2 et 3), nous traitons le problème décrit dans ce chapitre pour les critères diamètre et somme des distances. Pour chacun des critères, nous avons étudié :

- Le cas général des séquences on-line composées d'ajouts et de retraits mêlés.
- Le cas particulier des séquences on-line d'ajouts seuls.
- Le cas particulier des séquences on-line de retraits seuls.

Pour chacun de ces cas, nous avons étudié les modèles *sans* et (lorsque cela était nécessaire pour des raisons de qualité de connexion) *avec reconstruction(s)*. Pour chaque modèle d'étude, nous avons proposé :

- Un algorithme pour le modèle *sans reconstruction* et l'analyse de son rapport de compétitivité pour le diamètre (resp. la somme des distances), ce qui correspond à une garantie sur la qualité de l'algorithme dans le *pire cas*. Chaque fois qu'un algorithme *avec reconstructions* est proposé, nous analysons le nombre de changements qu'il induit dans le *pire cas*.
- Une borne inférieure universelle (c'est-à-dire valable *pour tout* algorithme respectant les contraintes du modèle) sur le rapport de compétitivité dans le *pire cas* pour le diamètre (resp. la somme des distances) pour le modèle *sans reconstruction*, et une borne inférieure universelle sur le nombre de changements nécessaires dans le *pire cas* pour le modèle *avec reconstructions*.

Nous décrivons maintenant de manière plus précise l'organisation des chapitres 1, 2, 3 et 3.4. Les chapitres 1 et 2 sont organisés de façon similaire, la différence étant que le critère d'étude est le diamètre pour le chapitre 1, et la somme des distances pour le chapitre 2. Dans le chapitre 3, nous nous intéressons au relâchement des contraintes *on-line* et *arbre* imposées par notre modèle, et enfin, le chapitre 3.4 synthétise les résultats de cette première partie.

Chapitre 1. Dans la section 1.1, nous étudions le cas général des séquences de requêtes on-line mêlant ajouts et retraits de sommets. Nous montrons que dans ce cas, pour obtenir une qualité satisfaisante en termes de diamètre (c'est-à-dire un rapport de compétitivité constant pour le diamètre),

tout algorithme doit reconstruire l'arbre courant un nombre de fois linéaire en nombre de requêtes on-line.

Ces résultats très négatifs nous amènent à regarder, en section 1.2, le cas des séquences de requêtes on-line composées uniquement d'ajouts. Nous proposons dans cette section un algorithme *sans reconstruction*, permettant d'obtenir simplement un rapport de compétitivité constant pour le diamètre.

Nous envisageons ensuite dans la section 1.3 la situation symétrique des séquences de requêtes on-line composées uniquement de retraits. Nous proposons d'abord en section 1.3.1 un algorithme *sans reconstruction* dont le rapport de compétitivité n'est pas constant. Dans cette même section, nous montrons qu'il n'existe pas d'algorithme *sans reconstruction* dont le rapport de compétitivité est constant. Ce résultat nous amène à étudier en section 1.3.2 le modèle *avec reconstructions*, où nous proposons un algorithme induisant un nombre d'étapes critiques logarithmique en nombre de sommets retirés et un nombre constant de changements élémentaires moyen par étape. Nous montrons également que tout algorithme induit dans le pire cas un nombre d'étapes critiques au moins logarithmique en nombre de sommets retirés, prouvant ainsi que notre algorithme est optimal en ordre de grandeur.

Chapitre 2. Dans la section 2.1, nous étudions le cas général des séquences de requêtes on-line mêlant ajouts et retraits de sommets. Comme pour le diamètre (en section 1.1), nous montrons que pour obtenir une qualité satisfaisante en termes de somme des distances, tout algorithme doit reconstruire l'arbre courant un nombre de fois linéaire en nombre de requêtes on-line.

Ces résultats très négatifs nous amènent à regarder, en section 2.2, le cas des séquences de requêtes on-line composées uniquement d'ajouts. Nous proposons d'abord en section 2.2.1 un algorithme *sans reconstruction* dont le rapport de compétitivité n'est pas constant. Dans cette même section, nous montrons qu'il n'existe pas d'algorithme *sans reconstruction* dont le rapport de compétitivité est constant. Ce résultat nous amène à étudier en section 2.2.2 le modèle *avec reconstructions*, où nous proposons un algorithme induisant un nombre d'étapes critiques logarithmique en nombre de sommets ajoutés et un nombre constant de changements élémentaires moyen par étape. Nous montrons également que tout algorithme induit un nombre d'étapes critiques au moins logarithmique en nombre de sommets ajoutés, prouvant ainsi que notre algorithme est optimal en ordre de grandeur.

En section 2.3, nous donnons le seul résultat que nous avons pour l'instant pour la somme des distances dans le cas des séquences on-line composées uniquement de retraits dans le modèle *avec reconstructions* : tout algorithme induit un nombre d'étapes critiques au moins logarithmique en nombre de sommets retirés.

Chapitre 3. Dans ce chapitre, nous proposons de relâcher au maximum les contraintes de notre problème de départ, pour ne garder que la contrainte d'*incrémentalité* d'un groupe (nous définissons en section 3.1 une quantité associée à cette contrainte, que nous appelons le *coût d'une séquence incrémentale*). Puis, en section 3.2, nous donnons des bornes supérieures et inférieures pour tout graphe sur la valeur du coût d'une séquence incrémentale. En section 3.3, nous montrons que le problème de la construction d'une séquence incrémentale de coût minimum ne peut pas être approché avec un rapport d'approximation meilleur que 2, sauf si $P = NP$. Enfin, en section 3.4, nous proposons pour ce problème un algorithme $4-$approché.

Chapitre 1

Garanties sur le diamètre du groupe

Ce chapitre est dédié à la minimisation du diamètre des groupes dans les arbres couvrants successifs. La section 1.1 traite le cas général des séquences de requêtes on-line mêlant ajouts et retraits, la section 1.2 celui des séquences composées uniquement d'ajouts et la section 1.3 celui des séquences composées uniquement de retraits. Cette dernière section est divisée en deux sous-sections : la section 1.3.1 traite le problème dans le modèle *sans reconstruction* et la section 1.3.2 le traite dans le modèle *avec reconstructions* (voir [62]).

1.1 Cas des ajouts et retraits mêlés

Dans cette section, nous prouvons que pour tout algorithme respectant les contraintes *on-line*, *arbre* et *qualité pour le diamètre*, pour tout i suffisamment grand, il existe un graphe et une séquence qui induit un nombre d'étapes critiques linéaire en i (où i est le nombre de requêtes on-line).

La notion d'*adversaire adaptatif*. Pour montrer ce résultat, nous allons utiliser un *adversaire adaptatif*. Il s'agit d'une notion classique de l'algorithmique on-line (voir [18]). L'idée est la suivante : pour montrer un résultat d'impossibilité, un *adversaire* va révéler au fur et à mesure les données du problème de manière à piéger n'importe quel algorithme on-line. L'adversaire va s'*adapter* en révélant chaque nouvelle donnée à traiter en fonction du comportement de l'algorithme aux étapes précédentes. Ainsi, l'usage d'un tel adversaire permet d'obtenir un résultat universel d'impossibilité sur le niveau de qualité d'un algorithme on-line dans le *pire cas*.

Notons qu'il existe dans le contexte on-line un autre type d'adversaire fréquemment utilisé, appelé adversaire *oblivious*. L'idée est de piéger tout algorithme on-line en proposant une séquence *a priori*. En effet, même si cette séquence est *présentée* au fur et à mesure à l'algorithme, elle n'est pas *construite* au fur et à mesure en fonction du comportement effectif de celui-ci. Un tel adversaire est donc plus faible que l'adversaire adaptatif. L'adversaire oblivious est utilisé pour piégé les algorithmes on-line *randomisés*, c'est pourquoi nous n'aurons à aucun moment recours à ce type d'adversaire (les algorithmes que nous proposons étant tous évalués de manière déterministe dans le pire cas).

Nous définissons maintenant la séquence particulière suivante (à l'aide d'un adversaire adaptatif).

Définition de la séquence $M_0, \ldots, M_i$. Soit $c \geq 1$ une constante quelconque (représentant le niveau de qualité désiré pour le diamètre). Nous posons $p = \lceil c + 2 \rceil$. Soit $C = (V_C, E_C)$ le cycle de longueur p où toutes les arêtes ont un poids égal à 1. Soit A un algorithme on-line respectant

les contraintes *arbre* et *qualité pour le diamètre* avec un niveau c. Nous utilisons un adversaire adaptatif pour définir la séquence considérée dans le cycle C.

Nous soulignons le fait que nous ne spécifions pas ici toutes les étapes de la séquence, mais uniquement les étapes "principales", intéressantes pour notre analyse. Pour tout $k \geq 0$ et $b \in \{0,1\}$, nous notons chacune de ces étapes $i = \alpha(k,b)$, définie par $\alpha(k,0) = 2k(p-2)$ et $\alpha(k,1) = 2k(p-2)+p-2$. Remarquons que pour tout $k \geq 0$, on a $\alpha(k,0) < \alpha(k,1) < \alpha(k+1,0) < \alpha(k+1,1)$. Nous numérotons de cette manière les étapes successives pour mettre en évidence le caractère répétitif de la stratégie d'ajouts que décrivons maintenant.

Pour tout $i = \alpha(k,b)$, soit T_i l'arbre construit par l'algorithme A à l'étape i. Pour tout $k \geq 0$, nous définissons la séquence de requêtes on-line (ajouts ou retraits) de la manière suivante.

- À l'étape $\alpha(k,0) = 2k(p-2)$, on a :

  $$M_{\alpha(k,0)} = V_C$$

  Étant donné que $T_{\alpha(k,0)}$ est un arbre couvrant $M_{\alpha(k,0)}$, il est nécessairement constitué de toutes les arêtes du cycle C sauf une, que l'on note e_k. Soient v_k^1 et v_k^2 les sommets connectés par e_k. L'adversaire adaptatif choisit alors de retirer (un par un) tous les sommets de $M_{\alpha(k,0)}$ sauf v_k^1 et v_k^2 pour obtenir $M_{\alpha(k,1)}$.
- À l'étape $\alpha(k,1) = 2k(p-2)+p-2$, on a :

  $$M_{\alpha(k,1)} = \{v_k^1, v_k^2\}$$

  L'adversaire adaptatif choisit alors d'ajouter (un par un) à $M_{\alpha(k,1)}$ tous les sommets du cycle C pour obtenir $M_{\alpha(k+1,0)} = V_C$.

Nous avons spécifié avec $\alpha(k,b)$ uniquement les étapes "principales", correspondant aux étapes où l'adversaire adaptatif a à faire un choix. En effet, entre deux étapes "principales" successives $\alpha(k,0)$ et $\alpha(k,1)$ (resp. $\alpha(k,1)$ et $\alpha(k+1,0)$), les sommets sont retirés (resp. ajoutés) dans un ordre quelconque. Nous soulignons le fait que notre séquence s'arrête après la dernière étape "principale", lorsque exactement i requêtes ont été exécutées.

Tout algorithme induit un nombre d'étapes critiques au moins linéaire en i dans le pire cas. Le lemme suivant est préliminaire au Théorème 1. Il décrit les sous-séquences où *au moins* une étape critique a lieu.

Lemme 1 *Soit $c \geq 1$ une constante quelconque (représentant le niveau de qualité désiré pour le critère diamètre). Si, pour tout i, $\alpha(k,0) \leq i \leq \alpha(k,1)$, on a $D_{T_i}(M_i) \leq c \cdot D_{T_i^*}(M_i)$, alors :*

$$\sharp EC(T_{\alpha(k,0)}, T_{\alpha(k,0)+1}, \ldots, T_{\alpha(k,1)}) \geq 1$$

Preuve. Nous prouvons le lemme 1 par l'absurde. Supposons qu'il existe $k \geq 0$ tel que pour tout i, $\alpha(k,0) \leq i \leq \alpha(k,1)$, la contrainte *qualité pour le diamètre* est satisfaite avec un niveau c *et* il n'y a pas d'étape critique.

Pour tout i, $\alpha(k,0) \leq i \leq \alpha(k,1)$, les arbres $T_{\alpha(k,0)}, T_{\alpha(k,0)+1}, \ldots, T_{\alpha(k,1)}$ contiennent toutes les arêtes du cycle sauf une, notée e_k. Nous soulignons que l'absence d'étape critique implique que cette arête manquante est la même dans tous les arbres $T_{\alpha(k,0)}, T_{\alpha(k,0)+1}, \ldots, T_{\alpha(k,1)}$.

Nous nous intéressons maintenant à l'étape $\alpha(k,1) = 2k(p-2)+p-2$, où $M_{\alpha(k,1)} = \{v_k^1, v_k^2\}$. On a :

$$D_{T_{\alpha(k,1)}}\left(M_{\alpha(k,1)}\right) = p-1$$

De plus, $T^*_{\alpha(k,1)}$ est l'arbre composé uniquement de l'arête e_k et de ses deux sommets extrémités. On a alors :

$$D_{T^*_{\alpha(k,1)}}\left(M_{\alpha(k,1)}\right) = 1$$

Comme $p = \lceil c+2 \rceil$, on en déduit :

$$\frac{D_{T_{\alpha(k,1)}}\left(M_{\alpha(k,1)}\right)}{D_{T^*_{\alpha(k,1)}}\left(M_{\alpha(k,1)}\right)} = p - 1 = \lceil c+1 \rceil > c$$

Ce résultat contredit l'hypothèse assurant que la contrainte *qualité pour le diamètre* est respectée avec un niveau c. □

Le théorème suivant montre que si les contraintes *on-line*, *arbre* et *qualité pour le diamètre* sont satisfaites, tout algorithme induit un nombre d'étapes critiques dans le pire cas en $\Omega(i)$, où i est le nombre de requêtes on-line.

Théorème 1 *Soit $c \geq 1$ une constante quelconque (représentant le niveau de qualité désiré pour le critère diamètre). Pour tout algorithme* A*, pour tout i suffisamment grand, il existe un graphe G_0, il existe $M_0, \ldots, M_i$ tel que si* A *retourne une séquence d'arbres $T_0, \ldots, T_i$ couvrant respectivement $M_0, \ldots, M_i$ et respectant la contrainte* qualité pour le diamètre *de niveau c, alors :*

$$\sharp EC(T_0, \ldots, T_i) \in \Omega(i)$$

Preuve. Soit $c \geq 1$ une constante quelconque. Nous posons $p = \lceil c+2 \rceil$. Soit i le nombre de requêtes on-line. Il existe k et G_0 (où G_0 est ici le cycle non valué de longueur p), il existe $M_0, \ldots, M_i$ (la séquence définie dans le paragraphe précédent) tels que :

$$\alpha(k,1) \;\leq\; i \;\leq\; \alpha(k+1,1) \;=\; 2(k+1)(p-2) + p - 2$$

Comme $p = \lceil c+2 \rceil$ est une constante, on a $k \in \Omega(i)$. De plus, d'après le lemme 1, on a :

$$\left.\begin{array}{l} \sharp EC(T_{\alpha(0,0)}, T_{\alpha(0,0)+1}, \ldots, T_{\alpha(0,1)}) \;\geq\; 1 \\ \sharp EC(T_{\alpha(1,0)}, T_{\alpha(1,0)+1}, \ldots, T_{\alpha(1,1)}) \;\geq\; 1 \\ \qquad\qquad \vdots \\ \sharp EC(T_{\alpha(k,0)}, T_{\alpha(k,0)+1}, \ldots, T_{\alpha(k,1)}) \;\geq\; 1 \end{array}\right\} \Rightarrow\; \sharp EC(T_{\alpha(0,0)}, T_{\alpha(k,0)+1}, \ldots, T_{\alpha(k,1)}) \;\geq\; k+1$$

$$\begin{array}{lll} \Rightarrow & \sharp EC(T_0, \ldots, T_i) \;\geq\; k+1 & (\text{car } i \geq \alpha(k,1)) \\ \Rightarrow & \sharp EC(T_0, \ldots, T_i) \;\in\; \Omega(i) & (\text{car } k \in \Omega(i)) \end{array}$$

□

Le théorème 1 montre que le maintient d'une qualité constante pour le diamètre induit un nombre d'étapes critiques linéaire en nombre de requêtes on-line. Ce résultat est particulièrement négatif, puisqu'il montre que tout algorithme on-line doit reconstruire un nombre de fois comparable en ordre de grandeur à celui de la solution triviale consistant à casser l'arbre à chaque étape pour le reconstruire totalement.

Les résultats de la section 1.1 montrent que si nous voulons obtenir de meilleurs résultats en terme de rapport de compétitivité ou en terme de nombre d'étapes critiques, nous devons restreindre notre modèle d'étude. C'est ce que nous allons faire maintenant avec le cas des séquences de requêtes on-line composées uniquement d'ajouts d'une part (voir section 1.2), et celles composées uniquement de retraits d'autre part (voir section 1.3).

1.2 Cas des ajouts seuls

Borne supérieure

Lorsque nous nous restreignons aux ajouts on-line, le problème associé au diamètre devient très simple. En effet, nous proposons un algorithme $2-$compétitif très simple (appelé AD pour Ajouts pour le Diamètre), qui consiste à construire un arbre de plus courts chemins enracinés en un sommet quelconque du groupe initial puis à ajouter un plus court chemin entre cette racine et chaque nouveau sommet révélé. Nous définissons plus formellement AD de la manière suivante.

Algorithme Ajouts pour le Diamètre – AD

1 Soit $G = (V, E, w)$ un graphe.
2 Soit $M_0 \subseteq V$ le groupe initial.
3 Construire un arbre T_0 de plus courts chemins enraciné en r_0
4 ($r_0 \in M_0$ est quelconque)
5 Soit T_i l'arbre couvrant M_i à l'étape i.
6 Soit u_{i+1} le $i+1^{\text{ème}}$ sommet révélé de la séquence d'ajouts
7 Construire T_{i+1} couvrant $M_{i+1} = M_i \cup \{u_{i+1}\}$ en ajoutant à T_i
8 un plus court chemin de u_{i+1} à r_0 sans créer de cycle

Remarques : l'algorithme AD respecte bien les contraintes *arbres* et *emboîtement*, puisqu'à chaque étape d'ajout, on incrémente l'arbre courant d'au plus un chemin sans créer de cycle (voir lignes 7 et 8 de l'algorithme AD).

Construire T_0 un arbre de plus courts chemins enraciné en r_0 (voir lignes 3 et 4 de l'algorithme AR) peut être fait en temps polynomial, en utilisant l'algorithme de Dijkstra (voir [20]). De plus, incrémenter l'arbre ne nécessite que l'ajout d'un plus court chemin. Cela peut être fait en temps polynomial, à nouveau en utilisant l'algorithme de Dijkstra.

Analyse de l'algorithme AD. Nous prouvons maintenant que l'algorithme AD est $2-$compétitif.

Théorème 2 *L'algorithme* AD *est* $2-$*compétitif, c'est-à-dire que pour toute séquence d'ajouts* $M_0 \subset \cdots \subset M_i$*, on a :*

$$D_{T_i}(M_i) \leq 2D_{T_i^*}(M_i)$$

Preuve. Soient $u, v \in M_i$ tels que :

$$\begin{aligned} D_{T_i}(M_i) &= d_{T_i}(u,v) \leq d_{T_i}(u,r_0) + d_{T_i}(r_0,v) \\ &\quad \text{(d'après l'inégalité triangulaire)} \\ &\leq d_G(u,r_0) + d_G(r_0,v) \\ &\quad \text{(car } T_i \text{ est un arbre de plus courts chemins enraciné en } r_0) \\ &\leq 2D_G(M_i) \\ &\quad \text{(car } u,v,r_0 \in M_i) \\ &\leq 2D_{T_i^*}(M_i) \\ &\quad \text{(car pour tout arbre } T \text{ couvrant un} \\ &\quad \text{groupe } M \subseteq V, \text{ on a } D_G(M) \leq D_T(M)) \end{aligned}$$

□

1.3 Cas des retraits seuls

Lorsque nous nous restreignons aux retraits on-line, le problème devient plus complexe que pour les ajouts seuls. En effet, nous allons montrer que l'algorithme RD sans reconstruction (définis dans la section 1.3.1) est $(m_0 - 1)$ − compétitif pour le diamètre (avec $m_0 = |M_0|$ la taille du groupe initial), et que ce résultat est optimal dans le pire cas (c'est à dire qu'il n'existe pas d'algorithme dans le modèle *sans reconstruction* dont le rapport de compétitivité est meilleur). Cela va nous amener à examiner le modèle *avec reconstructions*. Le relâchement de la contrainte *emboîtement* va en effet nous permettre d'obtenir des résultats plus intéressants, c'est-à-dire un algorithme dont le nombre de reconstructions est optimal en ordre de grandeur et qui maintient un niveau de qualité constant pour le diamètre (voir section 1.3.2).

1.3.1 Modèle *sans reconstruction*

Borne supérieure

Nous proposons un algorithme traitant le problème du retrait on-line de sommets, $(m_0 - 1)$ − compétitif pour le diamètre et n'induisant aucune reconstruction. Nous définissons l'algorithme RD (pour Retraits pour le Diamètre) de la manière suivante.

Algorithme Retraits pour le Diamètre − RD

1 Soit $G = (V, E, w)$ un graphe.
2 Soit $M_0 \subseteq V$ le groupe initial.
3 Construire le graphe complet des distances $G_0 = (V_0, E_0, w_0)$ tel que $V_0 = M_0$,
4 E_0 contient toutes les arêtes possibles entre 2 sommets distincts $u, v \in V_0$,
5 pour toute arête uv dans E_0, on pose $w_0(e) = d_G(u, v)$.
6 Construire $T_{\min}$ un arbre couvrant de poids minimum de G_0.
7 Construire un arbre T_0 couvrant M_0 en associant à chaque arête uv de $T_{\min}$
8 un chemin de longueur $d_G(u, v) = w_0(uv)$ dans G.
9 Soit T_i l'arbre couvrant M_i à l'étape i.
10 Soit $u_{i+1} \in M_0$ le $i + 1^{\text{ième}}$ sommet révélé de la séquence de retraits.
11 Construire T_{i+1} en élaguant (si nécessaire) T_i pour obtenir
12 l'arbre élagué couvrant M_{i+1}.

Remarques : l'algorithme RD respecte bien les contraintes *arbre* et *emboîtement*, puisqu'à chaque étape de retrait, on élague (si nécessaire) l'arbre courant pour obtenir un nouvel arbre entièrement contenu dans le précédent.

Construire le graphe complet des distances G_0 (voir lignes 3, 4 et 5 de l'algorithme RD) peut être fait en temps polynomial, en utilisant l'algorithme de Dijkstra. Construire $T_{\min}$ un arbre couvrant de poids minimum de G_0 (voir ligne 6 de l'algorithme RD) peut être fait en temps polynomial, en utilisant l'algorithme de Prim (voir [20]). Construire T_0 à partir de $T_{\min}$ (voir lignes 7 et 8 de l'algorithme RD) peut être fait en temps polynomial, en utilisant l'algorithme de Dijkstra. Enfin, chaque étape de retrait nécessite l'élagage des branches mortes de l'arbre courant, ce qui peut-être à nouveau fait en temps polynomial par un simple parcours de celui-ci.

Nous soulignons que l'arbre $T_{\min}$ de poids minimum couvrant M_0 dans G_0 doit être construit

de sorte que son redéploiement en l'arbre T_0 (obtenu en associant à chaque arête uv de $T_{\min}$ un chemin de longueur $d_G(u,v) = w_0(uv)$ dans G) soit un arbre (d'après la contrainte *arbre*). On peut montrer qu'un tel arbre $T_{\min}$ existe et peut être construit en temps polynomial (à l'aide d'une version dirigée de l'algorithme de Prim). Cette propriété étant annexe au problème que nous nous posons, dans un souci de concision, nous ne développons pas sa preuve ici.

Analyse de l'algorithme RD**.** Nous prouvons maintenant que l'algorithme RD est $(m_0 - 1)-$compétitif. Pour cela, nous avons besoin du lemme préliminaire suivant.

Lemme 2 *Soit T_0 l'arbre couvrant M_0 construit par l'algorithme* RD*. Pour tous sommets $u, v \in M_0$, on a :*

$$d_{T_0}(u,v) \leq (m_0 - 1) d_G(u,v)$$

Preuve. Soit T_0 l'arbre couvrant M_0 construit par l'algorithme RD à partir de $T_{\min} = (V_{\min}, E_{\min}, w_{\min})$. Soient u et v deux sommets quelconques de M_0. Deux cas peuvent se produire :

- Si $uv \in E_{\min}$, on a $d_{T_{\min}}(u,v) = d_{G_0}(u,v)$. De plus, par construction de T_0 et $T_{\min}$, on a $d_{T_0}(u,v) = d_{T_{\min}}(u,v)$ et $d_{G_0}(u,v) = d_G(u,v)$. On en déduit :

$$d_{T_0}(u,v) = d_G(u,v) \leq (m_0 - 1) d_G(u,v)$$

- Sinon, on a $uv \notin E_{\min}$. Soient $u_1 u_2 \dots u_k$ le chemin reliant les sommets u et v dans l'arbre $T_{\min}$ (avec $u = u_1$ et $v = u_k$). On a alors :

$$d_{T_{\min}}(u,v) = d_{G_0}(u_1,u_2) + d_{G_0}(u_2,u_3) + \dots + d_{G_0}(u_{k-1},u_k) \tag{1.1}$$

Montrons maintenant par l'absurde que $\forall j,\ 1 \leq j \leq k-1,\ d_{G_0}(u_i,u_{i+1}) \leq d_{G_0}(u,v)$. Pour cela, on suppose qu'il existe u_i et u_{i+1} tels que $d_{G_0}(u_i,u_{i+1}) > d_{G_0}(u,v)$. Considérons alors l'arbre T' obtenu à partir de $T_{\min}$ en lui enlevant l'arête $u_i u_{i+1}$ et en lui ajoutant l'arête uv. T' est bien un arbre puisque l'ajout de uv relie entre elles les deux composantes connexes disjointes obtenues après le retrait de l'arête $u_i u_{i+1}$. Examinons maintenant le poids de l'arbre T' :

$$\begin{aligned} W(T') &= W(T_{\min}) - w_0(u_i,u_{i+1}) + w_0(u,v) \ < \ W(T_{\min}) \\ &\quad (\text{car } w_0(u_i,u_{i+1}) = d_{G_0}(u_i,u_{i+1}) > d_{G_0}(u,v) = w_0(u,v)) \end{aligned}$$

Ce résultat contredit le fait que $T_{\min}$ est un arbre couvrant de poids minimum de G_0. On en déduit que $\forall j,\ 1 \leq j \leq k-1$, on a :

$$d_{G_0}(u_i,u_{i+1}) \leq d_{G_0}(u,v) \tag{1.2}$$

D'après (1.1), (1.2) et comme $T_{\min}$ a $m_0 - 1$ arêtes (car $T_{\min}$ est un arbre couvrant de G_0), on a :

$$d_{T_{\min}}(u,v) \leq (m_0 - 1) d_{G_0}(u,v)$$

De plus, par construction de T_0 et G_0, on a $d_{T_0}(u,v) = d_{T_{\min}}(u,v)$ et $d_{G_0}(u,v) = d_G(u,v)$. On en déduit :

$$d_{T_0}(u,v) \leq (m_0 - 1) d_G(u,v)$$

□

Théorème 3 *L'algorithme* RD *est* $(m_0 - 1)$*-compétitif, c'est-à-dire que pour toute séquence de retraits* $M_0 \supset \cdots \supset M_i$*, on a :*

$$D_{T_i}(M_i) \leq (m_0 - 1) D_{T_i^*}(M_i)$$

Preuve. Soient $u_0, v_0 \in M_i$ deux sommets tels que $d_{T_i}(u_0, v_0) = D_{T_i}(M_i)$. Par définition de l'algorithme RD, l'arbre T_0 contient l'arbre T_i, on a donc $d_{T_i}(u_0, v_0) = d_{T_0}(u_0, v_0)$. On en déduit :

$$\begin{aligned} D_{T_i}(M_i) &= d_{T_0}(u_0, v_0) \\ &\leq (m_0 - 1) d_G(u_0, v_0) \\ &\quad \text{(d'après le lemme (2))} \\ &\leq (m_0 - 1) D_G(M_i) \\ &\quad \text{(car } u_0, v_0 \in M_i\text{)} \\ &\leq (m_0 - 1) D_{T_i^*}(M_i) \\ &\quad \text{(car pour tout arbre } T \text{ couvrant courant un} \\ &\quad \text{groupe } M \subseteq V\text{, on a } D_G(M) \leq D_T(M)\text{)} \end{aligned}$$

□

Le théorème 3 montre que l'algorithme RD est $(m_0 - 1)-$compétitif. Ce résultat est peu satisfaisant. En effet, si on veut un rapport de compétitivité constant pour le diamètre, cet algorithme n'est exploitable que pour le cas très restreint d'un groupe initial M_0 de taille constante.

Néanmoins, dans le pire cas, l'algorithme RD est optimal. En effet, nous allons maintenant prouver que si nous n'autorisons aucune reconstruction (c'est-à-dire si la contrainte *emboîtement* est respectée), il n'existe pas d'algorithme dont le rapport de compétitivité pour le diamètre est meilleur que celui de RD.

Borne inférieure

Le théorème 4 montre que pour tout algorithme A est au moins $(m_0 - 1) -$ compétitif pour le diamètre.

Théorème 4 *Pour tout algorithme* A *respectant les contraintes* arbre *et* emboîtement*, pour tout* i *suffisamment grand, il existe un graphe* G_0 *et une séquence de* i *retraits tels que* A *est au moins* $(m_0 - 1) -$ *compétitif pour le diamètre, c'est à dire que l'on a :*

$$D_{T_i}(M_i) \geq (m_0 - 1) D_{T_i^*}(M_i)$$

Preuve. Soit $G_0 = (V_0, E_0, w_0)$ le cycle à $i+2$ sommets tel que $\forall e \in E_0$, $w_0(e) = 1$. Soit $M_0 = V_0$ le groupe initial. Soit $T_0 = (V_{T_0}, E_{T_0}, w_{T_0})$ l'arbre couvrant M_0 construit à l'étape initiale par un algorithme quelconque A.

Comme T_0 est un arbre (car A respecte la contrainte *arbre*), il contient toutes les arêtes de G_0, sauf une, notée e_0. Soient u_0 et v_0 les deux sommets extrémités de l'arête e_0. Considérons maintenant n'importe quelle séquence de i retraits telle que $M_i = \{u_0, v_0\}$ (une telle séquence existe car $m_0 = i+2$). D'après la contrainte *emboîtement*, T_0 doit contenir T_i, donc T_i est également l'arbre contenant toutes les arêtes de G_0, sauf e_0. À l'étape i, on a donc :

$$D_{T_i}(M_i) = i + 1 = m_0 - 1$$

De plus, T_i^* est l'arbre composé uniquement de l'arête e_0 et de ses deux sommets extrémités. On a alors :

$$D_{T_i^*}(M_i) = 1$$

On en déduit :

$$D_{T_i}(M_i) \geq (m_0 - 1) D_{T_i^*}(M_i)$$

□

Bilan de la section 1.3.1 Les résultats de la section 1.3.1 montrent que si nous voulons garantir un rapport de compétitivité constant pour le diamètre, nous devons relâcher les contraintes que nous imposons. En effet, d'après le théorème 4, il n'existe pas d'algorithme respectant les contraintes *arbre* et *emboîtement* dont le rapport de compétitivité est constant. Nous allons donc maintenant relâcher la contrainte *emboîtement* : dans le modèle *avec reconstructions*, nous allons garantir une qualité constante pour le diamètre, au prix d'un certain nombre de reconstructions que nous devrons maîtriser.

1.3.2 Modèle *avec reconstructions*

Borne supérieure

Pour définir l'algorithme RDR (pour Retraits pour le Diamètre avec Reconstructions), nous avons besoin de l'algorithme suivant, que nous appelons DM (pour Distance Minimum). Nous notons DM(M) l'algorithme DM appliqué au groupe M de taille m qui renvoie un groupe $M(r^*)$ de taille $\lfloor \frac{m}{2} \rfloor + 1$ ainsi que r^*, que nous appelons sa *racine associée*.

Algorithme Distance Minimum – DM

1 Soit $G = (V, E, w)$ un graphe.
2 Soit $M \subseteq V$.
3 Pour chaque $r \in M$, trier les m sommets de M par valeur croissante de leur
4 distance à r : $r, u_1^r, \ldots, u_{m-1}^r$ $(d_G(r, u_1^r) \leq d_G(r, u_2^r) \leq \cdots \leq d_G(r, u_{m-1}^r))$.
5 Soit $M(r) = \left\{ r, u_1^r, \ldots, u_{\lfloor \frac{m}{2} \rfloor}^r \right\}$.
6 Retourner $M(r^*)$ et sa racine associée r^* tel que :
7 $$d_G\left(r^*, u_{\lfloor \frac{m}{2} \rfloor}^{r^*}\right) = \min\left\{ d_G\left(r, u_{\lfloor \frac{m}{2} \rfloor}^r\right) : r \in M \right\}$$

Remarques : pour tout $r \in M$, les sommets $u_1^r, \ldots, u_{m-1}^r$ peuvent être triés par valeur croissante de $d_G(r, u_k^r)$ (voir lignes 3 et 4 de l'algorithme DM) et leur groupe associé $M(r)$ construit en temps polynomial en utilisant l'algorithme de Dijkstra. L'algorithme DM(M) renvoie donc $M(r^*)$ et sa racine associée en temps polynomial.

L'algorithme RDR. L'idée principale de l'algorithme RDR est de définir des étapes particulières de retraits, appelées les *étapes de reconstruction* durant lesquelles l'arbre courant est (totalement) reconstruit (dans le but de respecter la contrainte *qualité pour le diamètre*). Entre deux étapes

successives de reconstruction, lorsqu'un membre est retiré, l'arbre courant est simplement élagué de ses éventuelles branches mortes.

La séquence (a_k) suivante définit les étapes de reconstruction de notre algorithme : $m_{a_0} = m_0$ est la taille du groupe initial M_0 et pour tout a_k ($k \geq 1$), $m_{a_k} = \left\lfloor \frac{m_{a_{k-1}}}{2} \right\rfloor$ est la taille du groupe M_{a_k}.

Algorithme Retraits pour le Diamètre avec Reconstructions – RDR

1 Soit $G = (V, E, w)$ un graphe.
2 Soit $M_0 \subseteq V$ le groupe initial.
3 À l'étape $a_0 = 0$:
4 Construire un arbre de plus courts chemins T_0 couvrant M_0,
5 enraciné en r_0, avec r_0 la racine associée retournée par $\mathsf{DM}(M_0)$.
6 Après la dernière étape de reconstruction a_k :
7 Soit M_{a_k+j} le groupe courant.
8 Soit u_{a_k+j} le $j^{\text{ème}}$ sommet à retirer depuis la dernière étape de reconstruction a_k.
9 Si $m_{a_k+j} > \left\lfloor \frac{m_{a_k}}{2} \right\rfloor$ (ce qui correspond à $j < m_{a_k} - \left\lfloor \frac{m_{a_k}}{2} \right\rfloor$)
10 Alors mettre à jour l'arbre $T_{a_k+j-1} = (V_{a_k+j-1}, E_{a_k+j-1})$ en l'élaguant
11 (si nécessaire). On obtient l'arbre élagué $T_{a_k+j} = (V_{a_k+j}, E_{a_k+j})$ couvrant
12 M_{a_k+j} et satisfaisant $E_{a_k+j} \subseteq E_{a_k+j-1}$.
13 Sinon, on a $m_{a_k+j} = \left\lfloor \frac{m_{a_k}}{2} \right\rfloor$ (ce qui correspond à $j = m_{a_k} - \left\lfloor \frac{m_{a_k}}{2} \right\rfloor$).
14 Il s'agit d'une étape de reconstruction et on a $m_{a_k+j} = \left\lfloor \frac{m_{a_k}}{2} \right\rfloor = m_{a_{k+1}}$.
15 Casser l'arbre courant et construire $T_{a_{k+1}}$, un arbre de plus courts chemins
16 couvrant $M_{a_{k+1}}$, enraciné en $r_{a_{k+1}}$ (avec $r_{a_{k+1}} \in M_{a_{k+1}}$ la racine associée
17 retournée par $\mathsf{DM}(M_{a_{k+1}})$). a_{k+1} est alors la nouvelle dernière étape de
18 reconstruction.

Remarques : les étapes de reconstruction de l'algorithme RDR sont potentiellement des étapes critiques, puisque l'arbre courant est totalement cassé puis reconstruit. Les autres étapes ne sont pas des étapes critiques puisque l'arbre courant est juste élagué (voir lignes 10 et 11 de l'algorithme RDR) pour obtenir un nouvel arbre entièrement contenu dans l'arbre précédent. Nous soulignons également que la contrainte *arbre* est respectée.

Pour toute étape de reconstruction (et pour l'étape initiale), construire T_i un arbre de plus courts chemins enraciné en r_0 (voir lignes 4, 5, 15, 16 et 17 de l'algorithme RDR) peut être fait en temps polynomial, en utilisant l'algorithme de Dijkstra. De plus, chaque étape non critique nécessite l'élagage des branches mortes de l'arbre courant, ce qui peut être à nouveau fait en temps polynomial, par un simple parcours de celui-ci. Enfin, l'algorithme DM s'exécutant en un temps polynomial, on en déduit que RDR s'exécute en temps polynomial.

L'algorithme RDR respecte la contrainte *qualité pour le diamètre*. Le théorème 5 montre que RDR respecte la contrainte *qualité pour le diamètre* avec un niveau $c = 4$.

Théorème 5 *Pour toute séquence de retraits $M_0 \supset \cdots \supset M_i$, l'algorithme RDR respecte la contrainte* qualité pour le diamètre *avec un niveau $c = 4$, c'est-à-dire que l'on a :*

$$D_{T_i}(M_i) \leq 4 D_{T_i^*}(M_i)$$

Preuve.

– Si i est une étape de reconstruction. Dans ce cas, $i = a_k$. Soient $u_0, v_0 \in M_{a_k}$ tels que $d_{T_{a_k}}(u_0, v_0) = D_{T_{a_k}}(M_{a_k})$ (avec T_{a_k} l'arbre couvrant M_{a_k} enraciné en r^*, construit par RDR à l'étape a_k). On a alors :

$$\begin{aligned} D_{T_{a_k}}(M_{a_k}) &= d_{T_{a_k}}(u_0, v_0) \leq d_{T_{a_k}}(u_0, r^*) + d_{T_{a_k}}(r^*, v_0) \\ &\quad \text{(d'après l'inégalité triangulaire)} \\ &= d_G(u_0, r^*) + d_G(r^*, v_0) \\ &\quad \text{(car } T_{a_k} \text{ est un arbre de plus courts chemin enraciné en } r^*) \\ &\leq 2D_G(M_{a_k}) \leq 4D_G(M_{a_k}) \\ &\quad \text{(car } u_0, v_0, r^* \in M_{a_k}) \\ &\leq 4D_{T^*_{a_k}}(M_{a_k}) \\ &\quad \text{(car pour tout arbre } T \text{ couvrant courant un} \\ &\quad \text{groupe } M \subseteq V, \text{ on a } D_G(M) \leq D_T(M)) \end{aligned}$$

– Sinon, i n'est pas une étape de reconstruction. Soit j, $1 \leq j < m_{a_k} - \left\lfloor \frac{m_{a_k}}{2} \right\rfloor$ le nombre de sommets retirés après la dernière étape de reconstruction a_k (c'est-à-dire que j est tel que $m_{a_k+j} \geq \left\lfloor \frac{m_{a_k}}{2} \right\rfloor + 1$). Soit $M(r^*) = \left\{ r^*, u_1^{r^*}, \ldots, u_{\left\lfloor \frac{m_{a_k}}{2} \right\rfloor}^{r^*} \right\}$ le groupe retourné par DM(M_{a_k}) lors de la dernière étape de reconstruction a_k. Comme $M_{a_k+j} \subset M_{a_k}$ (par définition de la séquence de retrait) et $M(r^*) \subseteq M_{a_k}$ avec $m_{a_k+j} \geq \left\lfloor \frac{m_{a_k}}{2} \right\rfloor + 1$ et $|M(r^*)| = \left\lfloor \frac{m_{a_k}}{2} \right\rfloor + 1$, on a $M_{a_k+j} \cap M(r^*) \neq \emptyset$. Il existe donc $v \in M_{a_k+j} \cap M(r^*)$. Comme $v \in M(r^*)$, $v = r^*$ ou bien $v = u_l^{r^*}$, avec $l \leq \left\lfloor \frac{m_{a_k}}{2} \right\rfloor$. Comme les sommets $r^*, \ldots, u_{\left\lfloor \frac{m_{a_k}}{2} \right\rfloor}^{r^*}$ sont triés par valeurs croissantes de leur distance à r^* (voir l'algorithme RDR), on a :

$$d_G(r^*, v) \leq d_G\left(r^*, u_{\left\lfloor \frac{m_{a_k}}{2} \right\rfloor}^{r^*}\right) \tag{1.3}$$

De plus, comme l'algorithme DM(M_{a_k}) trouve r^* et $M(r^*)$ tels que $d_G\left(r^*, u_{\left\lfloor \frac{m_{a_k}}{2} \right\rfloor}^{r^*}\right) = \min\left\{ d_G\left(r, u_{\left\lfloor \frac{m_{a_k}}{2} \right\rfloor}^{r}\right) : r \in M_{a_k} \right\}$, pour tout $r^0 \in M_{a_k+j} \subset M_{a_k}$, on a :

$$d_G\left(r^*, u_{\left\lfloor \frac{m_{a_k}}{2} \right\rfloor}^{r^*}\right) \leq d_G\left(r^0, u_{\left\lfloor \frac{m_{a_k}}{2} \right\rfloor}^{r^0}\right) \tag{1.4}$$

Comme $m_{a_k+j} \geq \left\lfloor \frac{m_{a_k}}{2} \right\rfloor + 1$ et comme $r^0, u_1^{r^0}, \ldots, u_{m_{a_k}-1}^{r^0}$ sont triés par valeurs croissantes de leur distance à r^0, il existe $u_l^{r^0} \in M_{a_k+j}$ avec $\left\lfloor \frac{m_{a_k}}{2} \right\rfloor \leq l \leq m_{a_k+j} - 1$ tel que :

$$d_G\left(r^0, u_{\left\lfloor \frac{m_{a_k}}{2} \right\rfloor}^{r^0}\right) \leq d_G\left(r^0, u_l^{r^0}\right) \tag{1.5}$$

D'après (1.3), (1.4), (1.5) et comme r^0 et $u_l^{r^0}$ appartiennent à M_{a_k+j}, par définition du diamètre, on obtient :

$$\exists v \in M_{a_k+j} \cap M(r^*) : d_G(r^*, v) \leq D_G(M_{a_k+j}) \tag{1.6}$$

Soient $u^0 \in M_{a_k+j}$ et $v^0 \in M_{a_k+j}$ tels que $d_{T_{a_k+j}}(u^0, v^0) = D_{T_{a_k+j}}(M_{a_k+j})$ (où T_{a_k+j} est l'arbre couvrant M_{a_k+j} construit par RDR à l'étape $a_k + j$). On a :

$$\begin{aligned}
D_{T_{a_k+j}}(M_{a_k+j}) &= d_{T_{a_k+j}}(u^0, v^0) \;=\; d_{T_{a_k}}(u^0, v^0) \\
&\quad \text{(car, par définition de l'algorithme RDR, on a } T_{a_k+j} \subseteq T_{a_k}) \\
&\leq d_{T_{a_k}}(u^0, r^*) + d_{T_{a_k}}(r^*, v^0) \\
&\quad \text{(car } r^* \in T_{a_k} \text{ et d'après l'inégalité triangulaire)} \\
&= d_G(u^0, r^*) + d_G(r^*, v^0) \\
&\quad \text{(car } T_{a_k} \text{ est un arbre de plus courts chemins enraciné en } r^*) \\
&\leq d_G(u^0, v) + d_G(v, r^*) + d_G(r^*, v) + d_G(v, v^0) \\
&\quad \text{(d'après l'inégalité triangulaire, en utilisant le sommet } v \text{ de (1.6))} \\
&\leq 4D_G(M_{a_k+j}) \\
&\quad \text{(car } v \in M_{a_k+j},\ u^0 \in M_{a_k+j},\ v^0 \in M_{a_k+j} \text{ et d'après (1.6))} \\
&\leq 4D_{T^*_{a_{k+j}}}(M_{a_{k+j}}) \\
&\quad \text{(car pour tout arbre } T \text{ couvrant courant un} \\
&\quad \text{groupe } M \subseteq V, \text{ on a } D_G(M) \leq D_T(M))
\end{aligned}$$

En conclusion, pour tout i, $0 \leq i \leq m_0 - 1$, on obtient $D_{T_i}(M_i) \leq 4D_{T_i^*}(M_i)$. □

Nous évaluons maintenant la qualité de l'algorithme RDR. Nous rappelons que l'évaluation de la qualité d'un algorithme dans le modèle *avec reconstructions* ne se mesure plus en termes de rapport de compétitivité (celui-ci étant désormais intégré dans les contraintes du modèle), mais suivant les deux critères suivants : d'une part le nombre d'étapes critiques induites et d'autre part le nombre de changements élémentaires moyen par étape effectués (voir définition 11). Plus ces deux quantités sont faibles, meilleur est l'algorithme.

L'algorithme RDR induit un nombre d'étapes critiques logarithmique en i. Le théorème 6 montre que dans le pire cas, l'algorithme RDR induit un nombre d'étapes critiques en $O(\log i)$, où i est le nombre de sommets retirés.

Théorème 6 *Pour toute séquence de retraits $M_0 \supset \cdots \supset u_i$, soit $T_0, \ldots, T_i$ $(0 \leq i \leq m_0 - 1)$ la séquence d'arbres retournée par l'algorithme RDR. On a :*

$$\sharp EC(T_0, \ldots, T_i) \;\leq\; \lfloor \log_2(2i) \rfloor \;\in\; O(\log i)$$

Preuve. Deux cas peuvent se produire :

- Si $i < m_0 - \left\lfloor \frac{m_0}{2} \right\rfloor$, par définition de l'algorithme RDR, il n'y a pas d'étape de reconstruction. Donc, $EC(T_0, \ldots, T_i) = 0$.
- Sinon, $i \geq m_0 - \left\lfloor \frac{m_0}{2} \right\rfloor \geq \frac{m_0}{2}$ et on a :

$$m_0 \leq 2i \tag{1.7}$$

De plus, par définition de la séquence (a_k), s'il y a p reconstructions (c'est-à-dire p étapes critiques), alors p est tel que :

$$
\begin{array}{rcll}
m_{a_{p+1}} < m_0 - i \leq m_{a_p} & \Rightarrow & m_0 - i \leq \frac{m_0}{2^p} & \text{(par définiton de la séquence } (a_k), \\
 & & & \text{on a } \forall k,\, m_{a_k} \leq \frac{m_0}{2^k}) \\
 & \Rightarrow & m_0 - i \leq \frac{2i}{2^p} & \text{(d'après (1.7))} \\
 & \Rightarrow & 1 \leq \frac{2i}{2^p} & \text{(par définition, } i \leq m_0 - 1) \\
 & \Rightarrow & p \leq \lfloor \log_2(2i) \rfloor & \text{(car } p \text{ est un entier)} \\
 & \Rightarrow & p \in O(\log i) &
\end{array}
$$

□

L'algorithme RDR effectue un nombre constant de changements élémentaires moyen par étape. Dans ce paragraphe, nous montrons que même si l'algorithme RDR induit un nombre logarithmique d'étapes critiques, il n'effectue qu'un nombre constant de changements élémentaires moyen par étape. Pour montrer cela, nous avons d'abord besoin des lemmes 3 et 4, ainsi que du corollaire 1, résultats préliminaires au théorème 7.

Lemme 3 *Soit T un arbre élagué couvrant un groupe M quelconque. Soit $T^c = (V_{T^c}, E_{T^c})$ l'arbre de connexion associé à T. On a :*

$$|E_{T^c}| \leq 2|M| - 3$$

Preuve. Pour tout sommet $u \in V_{T^c}$, on note $\delta(u)$ le degré de u dans T^c. Pour simplifier les notations, on note $\delta_{\leq 2} = \{u \in V_{T^c} : \delta(u) \leq 2\}$ et $\delta_{\geq 3} = \{u \in V_{T^c} : \delta(u) \geq 3\}$. Comme T^c est un arbre, on a :

$$|E_{T^c}| \;=\; |V_{T^c}| - 1 \;=\; |\delta_{\leq 2}| + |\delta_{\geq 3}| - 1 \tag{1.8}$$

De plus, on a :

$$2|E_{T^c}| \;=\; \sum_{u \in V_{T^c}} \delta(u) \;=\; \sum_{\substack{u \in V_{T^c} \\ \delta(u) \leq 2}} \delta(u) + \sum_{\substack{u \in V_{T^c} \\ \delta(u) \geq 3}} \delta(u) \;\geq\; |\delta_{\leq 2}| + 3|\delta_{\geq 3}| \tag{1.9}$$

D'après (1.8) et (1.9), on a :

$$2|\delta_{\leq 2}| + 2|\delta_{\geq 3}| - 2 \;\geq\; |\delta_{\leq 2}| + 3|\delta_{\geq 3}| \quad \Rightarrow \quad |\delta_{\leq 2}| - 2 \;\geq\; |\delta_{\geq 3}| \tag{1.10}$$

D'après (1.8) et (1.10), et comme tout sommet de T^c de degré 1 ou 2 appartient à M (par définition d'un arbre connexion), on obtient :

$$|E_{T^c}| \;\leq\; 2|\delta_{\leq 2}| - 3 \;\leq\; 2|M| - 3$$

□

Corollaire 1 *Soit i ($1 \leq i \leq m_0 - 1$) la $i^{ème}$ étape d'une séquence de retraits quelconque. Soient T_i l'arbre élagué couvrant le $i^{ème}$ groupe M_i construit par un algorithme quelconque et T_i^c l'arbre de connexion associé à T_i. On rappelle que le nombre de changements élémentaires effectués à l'étape i est noté $\sharp CE(T_i^c)$. On a :*

$$\sharp CE(T_i^c) \leq 4|M_i|$$

Preuve. D'après le lemme 3, on a $|E_{T^c_{i-1}}| \leq 2|M_{i-1}| - 3 \leq 2|M_i|$. Pour chaque étape i, chaque connexion (c'est-à-dire arête de l'arbre T^c_i) peut être cassée au plus une fois, et il y a au plus $|E_{T^c_i}| \leq 2|M_i| - 2 \leq 2|M_i|$ nouvelles connexions (arêtes) à construire. On en déduit qu'à l'étape i, n'importe quel algorithme fait au plus $|E_{T^c_{i-1}}| + |E_{T^c_i}| \leq 4|M_i|$ changements élémentaires. □

Le corollaire 1 montre entre autres que même si i est une étape critique, l'algorithme RDR effectue au plus $4|M_i|$ changements élémentaires à l'étape i.

Lemme 4 *Soit i ($1 \leq i \leq m_0 - 1$) la $i^{ème}$ étape d'une séquence de retraits quelconque tel que i n'est pas une étape critique. Soient T_i l'arbre retourné par l'algorithme* RDR *à la $i^{ème}$ étape et T^c_i l'arbre de connexion associé à T_i. On a :*

$$\sharp CE(T^c_i) \leq 4$$

Preuve. Par définition de RDR, à chaque étape i non critique (c'est-à-dire à chaque étape où l'arbre n'est pas cassé, puis reconstruit), le nouvel arbre T_i est obtenu par élagage des branches inutiles de T_{i-1}. Comme T_{i-1} est un arbre élagué couvrant M_{i-1} et comme il n'y a qu'un seul sommet u retiré de T_{i-1} pour obtenir T_i, nous évaluons le nombre de changements élémentaires dans T^c_i de la manière suivante :

- Si u est un sommet de degré supérieur ou égal à 3 dans T_{i-1}, alors $T^c_i = T^c_{i-1}$ (car u est toujours un sommet de connexion dans T^c_i, car son degré est toujours supérieur ou égal à 3 dans T^c_i). On a donc :
$$\sharp CE(T^c_i) = 0 \leq 4$$
- Si u est un sommet de degré 2 dans T_{i-1}. Soient v et w les deux voisins de u dans T^c_{i-1}. T^c_i est obtenu à partir de T^c_{i-1} par suppression des arêtes vu et uw de T^c_{i-1} et l'ajout de l'arête vw. On a donc :
$$\sharp CE(T^c_i) = 2 + 1 = 3 \leq 4$$
- Si u est un sommet de degré 1 dans T_{i-1}. Soit v l'unique voisin de u dans T^c_{i-1}. Deux cas peuvent se produire :
  - Si $v \in M_{i-1}$, alors T^c_i est obtenu à partir de T^c_{i-1} par suppression de l'arête uv dans T^c_{i-1}. On a donc :
  $$\sharp CE(T^c_i) = 1 \leq 4$$
  - Sinon, $v \notin M_{i-1}$. Comme v n'appartient pas à M_{i-1} et est un sommet de connexion (car v est un sommet de l'arbre T^c_{i-1}), son degré dans T_{i-1} est nécessairement supérieur ou égal à 3. Deux sous-cas peuvent alors se produire :
    - Si v est un sommet de degré supérieur ou égal à 4 dans T_{i-1}, alors T^c_i est obtenu à partir de T^c_{i-1} par suppression de l'arête uv dans T^c_{i-1} (et v est toujours un sommet de connexion dans T^c_i car son degré est alors supérieur ou égal à $4 - 1 = 3$). On a donc :
    $$\sharp CE(T^c_i) = 1 \leq 4$$
    - Sinon, v est un sommet de degré 3. Soient w et t les sommets tels que u, w et t sont les trois voisins du sommet v dans T^c_{i-1}. T^c_i est alors obtenu à partir de T^c_{i-1} par suppression des trois arêtes uv, uw et ut dans T^c_{i-1} et par l'ajout de l'arête wt. On a donc :
    $$\sharp CE(T^c_i) = 3 + 1 = 4$$

□

Théorème 7 *Pour toute séquence de retraits $M_0 \supset \cdots \supset u_i$, soient $T_0, \ldots, T_i$ $(0 \leq i \leq m_0 - 1)$, la séquence d'arbre retournée par l'algorithme* RDR *et $T_0^c, \ldots, T_i^c$ la séquence des arbres de connexion associée. On a :*

$$\sharp CEM(T_0^c, \ldots, T_i^c) \ \leq \ 12$$

Preuve. Deux cas peuvent se produire :

– Si $i < m_0 - \left\lfloor \frac{m_0}{2} \right\rfloor$, alors, par définition de l'algorithme RDR, il n'y a pas d'étape critique (car pas d'étape de reconstruction). Donc, d'après le lemme 4, on a :

$$\sharp CEM(T_0^c, \ldots, T_i^c) = \frac{1}{i} \sum_{k=1}^{i} \sharp CE(T_k^c) \leq \frac{4i}{i} = 4$$

– Sinon, $i \geq m_0 - \left\lfloor \frac{m_0}{2} \right\rfloor \geq \frac{m_0}{2}$ et on obtient :

$$m_0 \leq 2i \tag{1.11}$$

Par définition de RDR, on a alors au moins une étape critique. Soit $\sharp CE_{cr}(T_0^c, \ldots, T_i^c)$ le nombre total de changements élémentaires effectués pendant les différentes étapes critiques et soit $\sharp CE_{\text{ncr}}(T_0^c, \ldots, T_i^c)$ le nombre total de changements élémentaires effectués pendant les étapes non critiques. Nous majorons maintenant les quantités $\sharp CE_{\text{cr}}(T_0^c, \ldots, T_i^c)$ et $\sharp CE_{\text{ncr}}(T_0^c, \ldots, T_i^c)$:

1. Majoration de $\sharp CE_{\text{cr}}(T_0^c, \ldots, T_i^c)$:
   D'après le corollaire 1, à chaque étape critique a_k, on a $CE(T_{a_k}^c) \leq 4|M_{a_k}|$. De plus, par définition de RDR, $|M_{a_k}| = m_{a_k} \leq \frac{m_0}{2^k}$. Enfin, d'après le théorème 6, RDR induit au plus $\lfloor \log_2(2i) \rfloor$ étapes critiques. On en déduit :

$$\begin{aligned}
\sharp CE_{\text{cr}}(T_0^c, \ldots, T_i^c) \ &\leq \ \sum_{p=1}^{\lfloor \log_2(2i) \rfloor} 4\frac{m_0}{2^p} \ = \ 4m_0 \left(\frac{\left(\frac{1}{2}\right)^{\lfloor \log_2(2i) \rfloor + 1} - \frac{1}{2}}{\frac{1}{2} - 1} \right) \\
&= \ 8m_0 \left(\frac{1}{2} - \left(\frac{1}{2}\right)^{\lfloor \log_2(2i) \rfloor + 1} \right) \\
&\leq \ 4m_0 \ \leq \ 8i \\
&\quad \text{(d'après (1.11))}
\end{aligned}$$

2. Majoration de $\sharp CE_{\text{ncr}}(T_0^c, \ldots, T_i^c)$:
   D'après le lemme 4, on a :

$$\sharp CE_{\text{ncr}}(T_0^c, \ldots, T_i^c) \leq 4i$$

Donc, comme $\sum_{k=1}^{i} \sharp CE(T_k^c) = \sharp CE_{\text{cr}}(T_0^c, \ldots, T_i^c) + \sharp CE_{\text{ncr}}(T_0^c, \ldots, T_i^c)$, on obtient :

$$\sharp CEM(T_0^c, \ldots, T_i^c) = \frac{1}{i} \sum_{k=1}^{i} \sharp CE(T_k^c) \leq \frac{8i + 4i}{i} = 12$$

□

Borne inférieure

Dans cette section, nous prouvons que pour tout algorithme respectant les contraintes *arbre* et *qualité pour le diamètre* avec un niveau constant c, pour tout i suffisamment grand, il existe une séquence particulière de retraits qui induit un nombre d'étapes critiques logarithmique en i (où i est le nombre de sommets retirés). Pour prouver cela, nous décrivons d'abord le graphe G et la séquence particulière de retraits.

Description du graphe G. Soient k, d, $0 \leq k \leq d$, et $p \geq 3$ trois entiers. On définit le graphe $G_k^p = (V_k^p, E_k^p, w_k^p)$ récursivement de la manière suivante :

- $G_0^p = (V_0^p, E_0^p, w_0^p)$ est le cycle de longueur p tel que $\forall e \in E_0^p$, $w_0^p(e) = 2^d$.
- $\forall k$, $1 \leq k \leq d$, on définit $G_k^p = (V_k^p, E_k^p, w_k^p)$ de la manière suivante. $\forall v \in V_{k-1}^p$, soit $C_v = (V_v^C, E_v^C, w_v^C)$ le cycle de longueur p tel que $v \in V_v^C$, $\left(V_v^C \backslash \{v\}\right) \cap V_{k-1}^p = \emptyset$ et $w_v^C(e) = \frac{2^{d-k}}{p^k}$. $G_k^p = (V_k^p, E_k^p, w_k^p)$ est le graphe tel que :
  - $V_k^p = V_{k-1}^p \cup \bigcup_{v \in V_{k-1}^p} V_v^C$
  - $E_k^p = E_{k-1}^p \cup \bigcup_{v \in V_{k-1}^p} E_v^C$
  - $\forall e \in E_{k-1}^p$, $w_k^p(e) = w_{k-1}^p(e)$ et $\forall e \in \bigcup_{v \in V_{k-1}^p} E_v^C$, $w_k^p(e) = w_v^C(e)$.

Voir la figure 1.1 pour une illustration du graphe G_k^p (avec $k = 2$ et $p = 4$). Nous pouvons maintenant

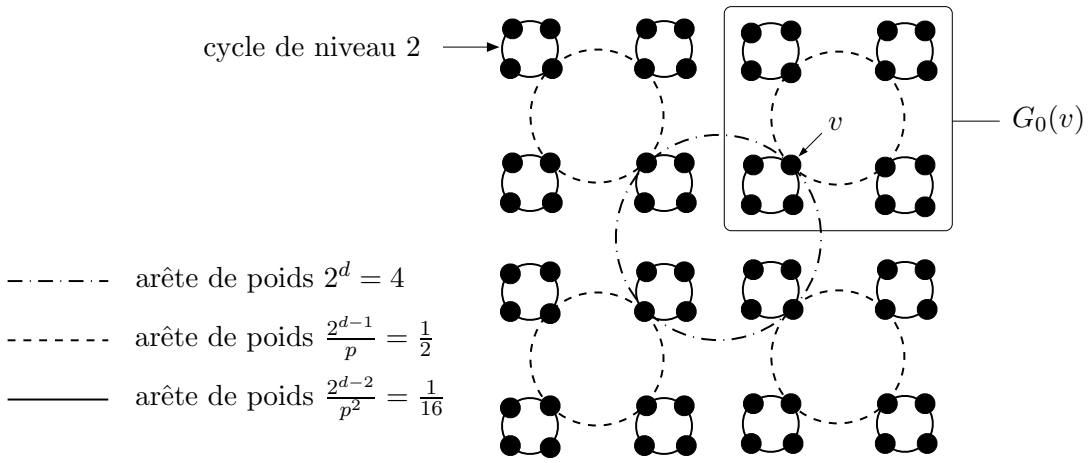


FIG. 1.1 – Le graphe G_2^4

définir le graphe $G = (V, E, w)$. Soit $c \geq 1$ la constante correspondant au niveau de qualité requis pour le diamètre et d un entier suffisamment grand tel que $i \leq \left|V_d^{\lceil 6c+2 \rceil}\right| - 1$ (où i est le nombre de sommets retirés et $V_d^{\lceil 6c+2 \rceil} = M_0$ est le groupe initial). Nous posons $G = G_d^{\lceil 6c+2 \rceil}$.

Définition d'un cycle de niveau k. Un cycle $C = (V^C, E^C, w)$ (sous-graphe de G) est dit de niveau k ($0 \leq k \leq d$) si chaque arête $e \in E^C$ a un poids $w(e) = \frac{2^{d-k}}{p^k}$. voir figure 1.1 pour une illustration avec un cycle de niveau 2 (Nous soulignons que G_2^4 est trop petit pour être un graphe

de la forme $G_d^{\lceil 6c+2 \rceil}$, mais il ne s'agit que d'une illustration).

Définition des sous-graphes $G_k(v)$. Soit v un sommet quelconque du graphe G ($v \in V$). Soit k le plus petit entier tel que $C_k = (V_k^C, E_k^C, w)$ est le cycle de niveau k contenant le sommet v ($v \in V_k^C$). On définit $G_k(v) = (V_k(v), E_k(v), w)$ le sous-graphe induit par tous les sommets et arêtes atteignables à partir du sommet v en ne traversant que des arêtes de poids strictement inférieur à $\frac{2^{d-k}}{p^k}$ (c'est-à-dire en parcourant les arêtes des cycles de niveau strictement supérieur à k à partir du sommet v). Voir la figure 1.1 pour l'illustration d'un tel sous-graphe.

Définition de la séquence de retraits $M_0 \supset \cdots \supset M_i$. Soit A un algorithme *quelconque* respectant les contraintes *arbre* et *qualité* (pour le diamètre) avec un niveau c. Nous utilisons un *adversaire adaptatif* pour définir la séquence de retraits dans le graphe $G = (V, E, w)$ défini ci-dessus. Nous définissons d'abord une séquence générique de retraits de sommets. Nous soulignons que nous ne spécifions pas chaque étape élémentaire de retrait, mais seulement les étapes "importantes", intéressantes pour notre analyse (que nous notons $i = \alpha(k, b)$). Pour tout $k \geq 0$, pour tout $b \in \{0, 1\}$, pour tout $i = \alpha(k, b)$ ($0 \leq \alpha(k, b) \leq m_0 - 1$), soit T_i l'arbre couvrant M_i construit pour l'algorithme A à l'étape i. Nous soulignons qu'à chaque étape $\alpha(k, b)$ ($1 \leq \alpha(k, b) \leq m_0 - 1$), on a $\alpha(k, b) = |M_0| - |M_{\alpha(k,b)}|$. La séquence de retraits est alors définie de la manière suivante. Nous posons $p = \lceil 6c + 2 \rceil$.

Cas de base :

- À l'étape $\alpha(0, 0) = 0$, on a :
$$M_{\alpha(0,0)} = V$$
Comme $T_{\alpha(0,0)}$ est un arbre couvrant $M_{\alpha(0,0)}$, $T_{\alpha(0,0)}$ est nécessairement composé (entre autres) de toutes les arêtes du cycle $C_0 = (V_0^C, E_0^C, w)$, sauf une arête e_0. Soient v_0^1 et v_0^2 les deux sommets connectés par l'arête e_0. L'adversaire adaptatif enlève maintenant de $M_{\alpha(0,0)}$ (un par un et dans n'importe quel ordre) tous les sommets de $\bigcup_{v \in V_0^C \setminus \{v_0^1, v_0^2\}} V_1(v)$ pour obtenir $M_{\alpha(0,1)}$.

- À l'étape $\alpha(0, 1)$, on a :
$$M_{\alpha(0,1)} = V_1(v_0^1) \cup V_1(v_0^2)$$
L'adversaire adaptatif enlève maintenant de $M_{\alpha(0,1)}$ (un par un et dans n'importe quel ordre) tous les sommets de $V_1(v_0^1)$ pour obtenir $M_{\alpha(1,0)}$ (nous soulignons que l'adversaire choisit d'enlever tous les sommets de $V_1(v_0^1)$ plutôt que de $V_1(v_0^2)$ arbitrairement).

Cas général :

- À l'étape $\alpha(k, 0)$. Soit $C_k = (V_k^C, E_k^C, w)$ le cycle de niveau k tel que $V_k^C \subset M_{\alpha(k-1,1)}$. On a :
$$M_{\alpha(k,0)} = \bigcup_{v \in V_k^C} V_{k+1}(v)$$
Comme $T_{\alpha(k,0)}$ est un arbre couvrant $M_{\alpha(k,0)}$, $T_{\alpha(k,0)}$ est nécessairement composé (entre autres), de toutes les arêtes du cycle C_k, sauf une arête e_k. Soient v_k^1 et v_k^2 les deux sommets connectés par l'arête e_k. L'adversaire adaptatif enlève maintenant de $M_{\alpha(k,0)}$ (un par un et dans n'importe quel ordre) tous les sommets de $\bigcup_{v \in V_k^C \setminus \{v_k^1, v_k^2\}} V_{k+1}(v)$ pour obtenir $M_{\alpha(k,1)}$.

– À l'étape $\alpha(k,1)$, on a :

$$M_{\alpha(k,1)} = V_{k+1}(v_k^1) \cup V_{k+1}(v_k^2)$$

L'adversaire adaptatif enlève maintenant de $M_{\alpha(k,1)}$ (un par un et dans n'importe quel ordre) tous les sommets de $V_{k+1}(v_k^1)$ pour obtenir $M_{\alpha(k+1,0)}$ (nous soulignons que l'adversaire choisit d'enlever tous les sommets de $V_{k+1}(v_k^1)$ plutôt que de $V_{k+1}(v_k^2)$ arbitrairement).

Remarques : nous avons spécifié avec $\alpha(k,b)$ seulement les étapes "importantes" de la séquence de retraits, correspondant aux étapes où l'adversaire adaptatif a à faire un choix. En effet, entre deux étapes "importantes" successives $\alpha(k,0)$ et $\alpha(k,1)$ (resp. $\alpha(k,1)$ et $\alpha(k+1,0)$), les sommets sont retirés un par un dans n'importe quel ordre. Nous arrêtons d'enlever des sommets après la dernière étape "importante", lorsque exactement i sommets ont été retirés. La figure 1.2 illustre les six premières étapes importantes $\alpha(0,0)$, $\alpha(0,1)$, $\alpha(1,0)$, $\alpha(1,1)$, $\alpha(2,0)$ et $\alpha(2,1)$ de la séquence de retraits. Nous soulignons que dans cet exemple, les arbres successifs sont construits par un algorithme arbitraire. Nous rappelons que G_2^4 est trop petit pour être un graphe de la forme $G_d^{\lceil 6c+2 \rceil}$, mais il ne s'agit que d'illustrer les notations et le mécanisme de l'adversaire.

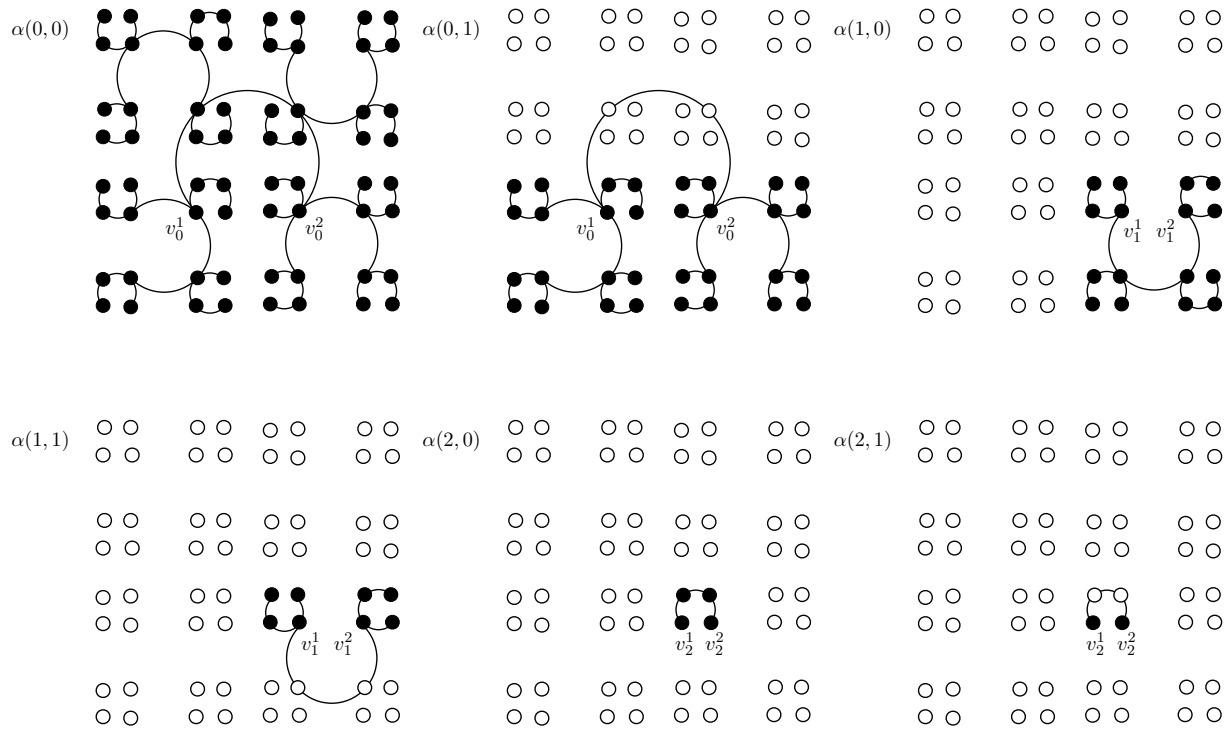


FIG. 1.2 – Illustration de la séquence de retraits sur le graphe G_2^4

Tout algorithme induit un nombre d'étapes critiques au moins logarithmique en i dans le pire cas. Le lemme préliminaire suivant est central dans notre analyse. Il décrit les sous-séquences de retraits où *au moins* une étape critique a lieu.

Lemme 5 *Soit $c \geq 1$ une constante quelconque (représentant le niveau de qualité requis pour le diamètre) et soit G le graphe décrit dans cette section. Pour tout $k \geq 0$, soient $T_{\alpha(k,0)}, T_{\alpha(k,0)+1}, \ldots, T_{\alpha(k,1)}$ n'importe quels arbres couvrant respectivement $M_{\alpha(k,0)}, M_{\alpha(k,0)+1}, \ldots, M_{\alpha(k,1)}$. Si, pour tout i, $\alpha(k,0) \leq i \leq \alpha(k,1)$, on a $D_{T_i}(M_i) \leq c \cdot D_{T_i^*}(M_i)$, alors :*

$$\sharp EC(T_{\alpha(k,0)}, T_{\alpha(k,0)+1}, \ldots, T_{\alpha(k,1)}) \geq 1$$

Preuve. Nous prouvons le lemme 5 par l'absurde. Supposons qu'il existe $k \geq 0$ tel que pour tout i, $\alpha(k,0) \leq i \leq \alpha(k,1)$, la contrainte *qualité pour le diamètre* est satisfaite et qu'il n'existe aucune étape critique, c'est-à-dire qu'il existe $k \geq 0$ tel que pour tout i, $\alpha(k,0) \leq i \leq \alpha(k,1)$, on a $D_{T_i}(M_i) \leq c \cdot D_{T_i^*}(M_i)$ et chaque arbre de la séquence $T_{\alpha(k,0)}, T_{\alpha(k,0)+1}, \ldots, T_{\alpha(k,1)}$ est contenu dans le précédent. Ces arbres sont composés (entre autres) de toutes les arêtes du cycle $C_k \subset G$, sauf une arête, notée e_k. Nous soulignons le fait que, comme il n'y a pas d'étape critique, l'arête manquante e_k est toujours la même dans tous les arbres $T_{\alpha(k,0)}, T_{\alpha(k,0)+1}, \ldots, T_{\alpha(k,1)}$.

Nous nous concentrons sur l'étape $\alpha(k,1)$, où nous avons $M_{\alpha(k,1)} = V_{k+1}(v_k^1) \cup V_{k+1}(v_k^2)$. Nous allons maintenant minorer $D_{T_{\alpha(k,1)}}\left(M_{\alpha(k,1)}\right)$ et majorer $D_{T^*_{\alpha(k,1)}}\left(M_{\alpha(k,1)}\right)$ pour montrer qu'à cette étape particulière, la contrainte *qualité pour le diamètre* n'est pas satisfaite. Cela va nous mener à une contradiction et donc nous permettre de prouver le lemme.

– Minoration de $D_{T_{\alpha(k,1)}}\left(M_{\alpha(k,1)}\right)$. Comme les deux sommets v_k^1 et v_k^2 extrémités de l'arête e_k sont reliés par un chemin fait de $p-1 = \lceil 6c+1 \rceil$ arêtes de poids $\frac{2^{d-k}}{p^k}$ dans $T_{\alpha(k,1)}$ (alors qu'ils sont reliés par l'arête e_k de poids $\frac{2^{d-k}}{p^k}$ dans le graphe G), on a :

$$D_{T_{\alpha(k,1)}}\left(M_{\alpha(k,1)}\right) \;\geq\; (p-1)\frac{2^{d-k}}{p^k} \;\geq\; (6c+1)\frac{2^{d-k}}{p^k} \tag{1.12}$$

– Majoration de $D_{T^*_{\alpha(k,1)}}\left(M_{\alpha(k,1)}\right)$. Par construction du graphe G, deux cas peuvent se produire :

1. Si $k = d$, il n'y a pas de cycle de niveau $k+1$ dans G. On a donc :

$$D_G\left(M_{\alpha(k,1)}\right) \;=\; w(e_k) \;=\; \frac{2^{d-k}}{p^k} \;\leq\; 3\frac{2^{d-k}}{p^k}$$

2. Si $k \leq d-1$, on a :

$$\begin{aligned}
D_G\left(M_{\alpha(k,1)}\right) &\leq D_G\left(V_{k+1}(v_k^1)\right) + d_G(v_k^1, v_k^2) + D_G\left(V_{k+1}(v_k^2)\right) \\
&\leq \sum_{e \in E_{k+1}(v_k^1)} w(e) + w(e_k) + \sum_{e \in E_{k+1}(v_k^2)} w(e) \\
&\quad \text{(car pour tout graphe (ou sous-graphe)} \\
&\quad G = (V, E, w),\ D_G(V) \leq \sum_{e \in E} w(e)) \\
&\leq \sum_{l=k+1}^{d} \frac{2^{d-l}}{p^l} p^{l-k} + \frac{2^{d-k}}{p^k} + \sum_{l=k+1}^{d} \frac{2^{d-l}}{p^l} p^{l-k} \\
&\leq \frac{2}{p^k} \sum_{l=k+1}^{d} 2^{d-l} + \frac{2^{d-k}}{p^k} \;\leq\; 2\frac{2^{d-k}}{p^k} + \frac{2^{d-k}}{p^k} \;=\; 3\frac{2^{d-k}}{p^k}
\end{aligned}$$

De plus, d'après [46], il existe un arbre $T_{\alpha(k,1)^{\text{off}}}$ couvrant $M_{\alpha(k,1)}$ tel que $D_{T^{\text{off}}_{\alpha(k,1)}}(M_{\alpha(k,1)}) \leq 2D_G(M_{\alpha(k,1)})$. Comme $T^*_{\alpha(k,1)}$ est un arbre couvrant $M_{\alpha(k,1)}$ optimal pour le diamètre, on en déduit :

$$D_{T^*_{\alpha(k,1)}}(M_{\alpha(k,1)}) \;\leq\; D_{T^{\text{off}}_{\alpha(k,1)}}(M_{\alpha(k,1)}) \;\leq\; 2D_G(M_{\alpha(k,1)}) \;\leq\; 6\frac{2^{d-k}}{p^k} \tag{1.13}$$

d'après (1.12) et (1.13), on obtient :

$$\frac{D_{T_{\alpha(k,1)}}\left(M_{\alpha(k,1)}\right)}{D_{T^*_{\alpha(k,1)}}\left(M_{\alpha(k,1)}\right)} \;\geq\; \frac{(6c+1)\frac{2^{d-k}}{p^k}}{6\frac{2^{d-k}}{p^k}} \;\geq\; c+\frac{1}{6} \;>\; c$$

Ce résultat contredit l'hypothèse de départ qui nous dit que la contrainte *qualité pour le diamètre* est respectée avec un niveau c. □

Le théorème suivant montre que si les contraintes *on-line*, *arbre* et *qualité pour le diamètre* sont satisfaites, tout algorithme induit un nombre d'étapes critiques en $\Omega(\log i)$ dans le pire cas (où i est le nombre de sommets retirés).

Théorème 8 *Soit $c \geq 1$ une constante quelconque (représentant le niveau de qualité requis). Pour tout algorithme* A*, pour tout i suffisamment grand, il existe un graphe, il existe une séquence $M_0 \supset \cdots \supset M_i$ tels que si l'algorithme* A *retourne une séquence d'arbres $T_0, \ldots, T_i$ couvrant respectivement $M_0 \supset \cdots \supset M_i$ et respectant la contrainte* qualité pour le diamètre *avec un niveau c, alors on a :*

$$\sharp EC(T_0, \ldots, T_i) \in \Omega(\log i)$$

Preuve. Soit $c \geq 1$ une constante quelconque. Nous posons $p = \lceil 6c+2 \rceil$. Soit i le nombre de sommets enlevés. Il existe d et G (G est le graphe défini dans cette section), il existe $M_0 \supset \cdots \supset M_i$ (la séquence définie dans cette section) tels que :

$$\alpha(d-1,1) \;\leq\; i \;\leq\; \alpha(d,1) \;\leq\; |V| \;=\; p^{d+1}$$

On a donc : $i \leq p^{d+1} \quad \Rightarrow \quad \log_p(i) - 1 \leq d$. Comme $p = \lceil 6c+2 \rceil$ est une constante, on a $d \in \Omega(\log i)$. De plus, d'après le lemme 5, on obtient :

$$\left.\begin{array}{l} \sharp EC(T_{\alpha(0,0)}, T_{\alpha(0,0)+1}, \ldots, T_{\alpha(0,1)}) \;\geq\; 1 \\ \sharp EC(T_{\alpha(1,0)}, T_{\alpha(1,0)+1}, \ldots, T_{\alpha(1,1)}) \;\geq\; 1 \\ \qquad\qquad \vdots \\ \sharp EC(T_{\alpha(d-1,0)}, T_{\alpha(d-1,0)+1}, \ldots, T_{\alpha(d-1,1)}) \;\geq\; 1 \end{array}\right\} \Rightarrow\; \sharp EC(T_{\alpha(0,0)}, T_{\alpha(d-1,0)+1}, \ldots, T_{\alpha(d-1,1)}) \;\geq\; d$$

$$\begin{array}{lll} \Rightarrow & \sharp EC(T_0, \ldots, T_i) \;\geq\; d & (\text{car } i \geq \alpha(d-1,1)) \\ \Rightarrow & \sharp EC(T_0, \ldots, T_i) \;\in\; \Omega(\log i) & (\text{car } d \in \Omega(\log i)) \end{array}$$

□

Bilan de la section 1.3.2. Les théorèmes 6 et 8 montrent que l'algorithme avec reconstructions que nous proposons (RDR) permet d'obtenir un nombre optimal (en ordre de grandeur) d'étapes critiques (en $O(\log i)$, avec i le nombre de retraits). De plus, le théorème 7 nous garantit que le nombre de changements élémentaires moyen par étape effectué par l'algorithme RDR est borné par la constante 12.

Nous allons maintenant, dans le chapitre suivant, nous intéresser à une métrique plus fine que le diamètre pour mesurer la qualité d'un algorithme (toujours en termes de rapport de compétitivité ou de contrainte *qualité*) : la somme des distances entre les sommets d'un groupe.

Chapitre 2

Garanties sur la somme des distances entre les membres du groupe

Ce chapitre est dédié à l'étude de la somme des distances entre les membres du groupe courant dans les arbres couvrants successifs. La section 2.1 traite le cas général des séquences de requêtes on-line mêlant ajouts et retraits. La section 2.2 quant à elle traite le cas des séquences composées uniquement d'ajouts. Cette section est divisée en deux sous-sections : la section 2.2.1 traite le problème dans le modèle *sans reconstruction* (voir [56, 58]) et la section 2.2.2 le traite dans le modèle *avec reconstructions* (voir [63]). Enfin, la section 2.3 traite le cas des séquences composées uniquement de retraits.

2.1 Cas des ajouts et retraits mêlés

Le théorème 9 montre que pour tout algorithme respectant les contraintes *on-line*, *arbre* et *qualité pour la somme des distances* avec un niveau c, pour tout i suffisamment grand, il existe un graphe et une séquence qui induit un nombre d'étapes critiques au moins linéaire en i, où i est le nombre de requêtes on-line.

Théorème 9 *Soit $c \geq 1$ une constante quelconque (représentant le niveau de qualité désiré pour la somme des distances). Pour tout algorithme* A*, pour tout i suffisamment grand, il existe un graphe G_0, il existe $M_0, \ldots, M_i$ tel que si* A *retourne une séquence d'arbres $T_0, \ldots, T_i$ couvrant respectivement $M_0, \ldots, M_i$ et respectant la contrainte* qualité pour la somme des distances *de niveau c, alors :*

$$\sharp EC(T_0, \ldots, T_i) \in \Omega(i)$$

Preuve. Le théorème 9 se prouve exactement de la même façon que le théorème 1, en remplaçant dans le lemme 1 $D_{T_{\alpha(k,1)}}\left(M_{\alpha(k,1)}\right) = p-1$ par $C_{T_{\alpha(k,1)}}\left(M_{\alpha(k,1)}\right) = 2(p-1)$ et $D_{T^*_{\alpha(k,1)}}\left(M_{\alpha(k,1)}\right) = 1$ par $C_{T^*_{\alpha(k,1)}}\left(M_{\alpha(k,1)}\right) = 2$. □

Le théorème 9 montre qu'obtenir une qualité constante pour la somme des distances induit un nombre d'étapes critiques linéaire en i (où i est le nombre de requêtes on-line). Ce résultat est particulièrement négatif, puisqu'il montre que tout algorithme on-line doit reconstruire l'arbre courant le même nombre de fois, en ordre de grandeur, que la solution triviale consistant à casser l'arbre à chaque étape pour le reconstruire totalement.

Les résultats de la section 2.1 montrent que si nous voulons obtenir de meilleurs résultats en termes de rapport de compétitivité ou en termes de nombre de reconstructions à effectuer, de la

même façon que pour le diamètre (voir chapitre 1), nous devons restreindre notre modèle d'étude. C'est ce que nous allons faire maintenant avec le cas des séquences de requêtes on-line composées uniquement d'ajouts d'une part (voir section 2.2), et celles composées uniquement de retraits d'autre part (voir section 2.3).

2.2 Cas des ajouts seuls

Nous nous intéressons dans cette section au problème des séquences composées uniquement de requêtes d'ajouts. Nous allons montrer que dans le modèle *sans reconstruction*, l'algorithme AS définis dans la section 2.2.1 (pour Ajouts pour la Somme des distances) est $\max(2i, 12)$ − compétitif pour la somme des distances. Nous montrons ensuite que ce résultat est optimal en ordre de grandeur (c'est-à-dire que tout algorithme dans le modèle *sans reconstruction* a un rapport de compétitivité en $\Omega(i)$). Cela va nous amener à examiner le modèle *avec reconstructions*. Le relâchement de la contrainte *emboîtement* va en effet nous permettre d'obtenir des résultats plus intéressants, c'est-à-dire un algorithme dont le nombre d'étape critiques induites est optimal en ordre de grandeur et qui maintient un niveau de qualité constant pour la somme des distances (voir section 2.2.2).

2.2.1 Modèle *sans reconstruction*

Borne supérieure

Nous proposons un algorithme traitant le problème de l'ajout on-line de sommets, $\max(2i, 12)$ − compétitif pour la somme des distances. Pour définir l'algorithme AS (pour Ajouts pour la Somme des distances), nous avons besoin de la définition suivante d'un *médian* d'un groupe.

Définition 12 (Médian d'un groupe) *Soient $G = (V, E, w)$ un graphe et $M \subseteq V$ un groupe. Le sommet $r \in M$ est un* médian *de M si on a :*

$$\sum_{u \in M} d_G(u, r) = \min \left\{ \sum_{u \in M} d_G(u, v) : v \in M \right\}$$

L'algorithme AS consiste à construire un arbre de plus courts chemins enracinés en un médian du groupe initial puis à ajouter un plus court chemin entre cette racine et chaque nouveau sommet révélé. Nous définissons plus formellement l'algorithme AS de la manière suivante.

Algorithme Ajouts pour la Somme des distances − AS

Soit $G = (V, E, w)$ un graphe.
Soit $M_0 \subseteq V$ le groupe initial.
    Construire un arbre de plus courts chemins T_0 couvrant M_0,
    enraciné en r_0, médian de M_0.
Soit T_i l'arbre couvrant M_i à l'étape i.
Soit $u_{i+1} \in M_0$ le $i + 1^{\text{ème}}$ sommet révélé de la séquence d'ajouts.
    Construire l'arbre T_{i+1} couvrant $M_{i+1} = M_i \cup \{u_{i+1}\}$ en ajoutant à
    T_i un plus court chemin de u_{i+1} à r_0 sans créer de cycle.

Remarques : l'algorithme AS respecte bien la contrainte *emboîtement*, puisqu'à chaque étape d'ajout, on incrémente l'arbre courant d'au plus un chemin (voir lignes 7 et 8 de l'algorithme AS). Nous soulignons également que la contrainte *arbre* est respectée, puisqu'à chaque étape d'ajout, l'algorithme maintient l'absence de cycle et la connexité de la structure (voir lignes 7 et 8 de l'algorithme AS).

Calculer r_0 un médian de M_0 et construire T_0 un arbre de plus courts chemins enraciné en r_0 (voir lignes 3 et 4 de l'algorithme AS) peut être fait en temps polynomial, en utilisant l'algorithme de Dijkstra. De plus, incrémenter l'arbre ne nécessite que l'ajout d'un plus court chemin, ce qui peut être fait en temps polynomial, à nouveau en utilisant l'algorithme de Dijkstra.

Analyse de l'algorithme AS. Nous prouvons maintenant que l'algorithme AS est $\max(2i, 12)-$ compétitif. Pour cela, nous avons besoin des lemmes préliminaires 6, 7 et 8 suivants.

Lemme 6 *Soient $M \subseteq V$ un groupe et $r \in M$ un médian de M. On a alors :*

$$C_G(M) \geq |M| \sum_{u \in M} d_G(u, r)$$

Preuve. Par définition de $r \in M$, médian de M, on a :

$$C_G(M) \;=\; \sum_{v \in M} \sum_{u \in M} d_G(u, v) \;\geq\; \sum_{v \in M} \sum_{u \in M} d_G(u, r) \;=\; |M| \sum_{u \in M} d_G(u, r)$$

□

Lemme 7 *L'algorithme AS est m_i-compétitif, c'est-à-dire que pour toute séquence d'ajouts $M_0 \subset \cdots \subset M_i$, on a :*

$$C_{T_i}(M_i) \;\leq\; m_i \cdot C_{T_i^*}(M_i)$$

Preuve. D'après l'inégalité triangulaire, on a :

$$\begin{aligned}
C_{T_i}(M_i) \;&\leq\; \sum_{u \in M_i} \sum_{v \in M_i} (d_{T_i}(u, r_0) + d_{T_i}(r_0, v)) \;=\; \sum_{u \in M_i} \sum_{v \in M_i} (d_G(u, r_0) + d_G(r_0, v)) \\
&\quad \text{(car } T_i \text{ est un arbre de plus courts chemins enraciné en } r_0\text{)} \\
&=\; m_i \sum_{u \in M_i} d_G(u, r_0) + m_i \sum_{v \in M_i} d_G(r_0, v) \;=\; 2m_i \sum_{u \in M_i} d_G(u, r_0) \\
&\leq\; m_i \cdot C_G(M_i) \\
&\quad \text{(car } r_0 \in M_i \text{ avec } C_G(M_i) = \sum_{u \in M_i \setminus \{r_0\}} \sum_{v \in M_i \setminus \{r_0\}} d_G(u, v) + 2 \sum_{u \in M_i} d_G(u, r_0)\text{)} \\
&\leq\; m_i \cdot C_{T_i^*}(M_i) \\
&\quad \text{(car pour tout arbre } T \text{ couvrant courant un} \\
&\quad \text{groupe } M \subseteq V\text{, on a } C_G(M) \leq C_T(M)\text{)}
\end{aligned}$$

□

Notons que le résultat suivant (le lemme 8) est donné sous forme de lemme afin d'être réutilisé dans la preuve du théorème 12.

Lemme 8 *L'algorithme* AS *est* $\left(4 + \frac{6i}{m_0} + \frac{2i^2}{m_0^2}\right)$ *– compétitif, c'est-à-dire que pour toute séquence d'ajouts* $M_0 \subset \cdots \subset M_i$*, on a :*

$$C_{T_i}(M_i) \;\leq\; \left(4 + \frac{6i}{m_0} + \frac{2i^2}{m_0^2}\right) C_{T_i^*}(M_i)$$

Preuve. D'après l'inégalité triangulaire, on a :

$$\begin{aligned}
C_{T_i}(M_i) &\leq \sum_{u\in M_i}\sum_{v\in M_i}(d_{T_i}(u,r_0)+d_{T_i}(r_0,v)) \;=\; \sum_{u\in M_i}\sum_{v\in M_i}(d_G(u,r_0)+d_G(r_0,v)) \\
&\quad \text{(car } T_i \text{ est un arbre de plus courts chemins enraciné en } r_0\text{)} \\
&= m_i \sum_{u\in M_i} d_G(u,r_0) + m_i \sum_{v\in M_i} d_G(r_0,v) \;=\; 2m_i \sum_{u\in M_i} d_G(u,r_0) \\
&= 2m_i \sum_{u\in M_0} d_G(u,r_0) + 2m_i \sum_{u\in M_i\setminus M_0} d_G(u,r_0) \\
&\leq \frac{2m_i}{m_0} C_G(M_0) + 2m_i \sum_{u\in M_i\setminus M_0} d_G(u,r_i) + 2m_i \sum_{u\in M_i\setminus M_0} d_G(r_i,r_0) \\
&\quad \text{(d'après le lemme 6 avec } r_0 \text{ médian de } M_0 \text{ et l'inégalité triangulaire)} \\
&\leq \frac{2m_i}{m_0} C_G(M_i) + 2m_i \sum_{u\in M_i} d_G(u,r_i) + 2im_i d_G(r_i,r_0) \\
&\quad \text{(car } M_0 \subseteq M_i,\ M_i\setminus M_0 \subseteq M_i \text{ et } |M_i\setminus M_0| = i\text{)} \\
&\leq \left(\frac{2m_i}{m_0} + 2\right) C_G(M_i) + \frac{2im_i}{m_0} \sum_{u\in M_0} d_G(r_i,u) + \frac{2im_i}{m_0} \sum_{u\in M_0} d_G(u,r_0) \\
&\quad \text{(d'après le lemme 6 avec } r_i \text{ médian de } M_i \text{ et l'inégalité triangulaire)} \\
&\leq \left(\frac{2m_i}{m_0} + 2\right) C_G(M_i) + \frac{2im_i}{m_0} \sum_{u\in M_i} d_G(r_i,u) + \frac{2im_i}{m_0^2} C_G(M_0) \\
&\quad \text{(car } M_0 \subseteq M_i \text{ et d'après le lemme 6 avec } r_0 \text{ médian de } M_0\text{)} \\
&\leq \left(\frac{2m_i}{m_0} + 2 + \frac{2i}{m_0} + \frac{2im_i}{m_0^2}\right) C_G(M_i) \\
&\quad \text{(car } M_0 \subseteq M_i \text{ et d'après le lemme 6 avec } r_i \text{ médian de } M_i\text{)} \\
&= \left(4 + \frac{6i}{m_0} + \frac{2i^2}{m_0^2}\right) C_G(M_i) \\
&\quad \text{(car } m_i = m_0 + i\text{)} \\
&\leq \left(4 + \frac{6i}{m_0} + \frac{2i^2}{m_0^2}\right) C_{T_i^*}(M_i) \\
&\quad \text{(car pour tout arbre } T \text{ couvrant courant un} \\
&\quad \text{groupe } M \subseteq V\text{, on a } C_G(M) \leq C_T(M)\text{)}
\end{aligned}$$

□

Théorème 10 *L'algorithme* AS *est* $\max(2i, 12)$ *– compétitif, c'est-à-dire que pour toute séquence d'ajouts* $M_0 \subset \cdots \subset M_i$*, on a :*

$$C_{T_i}(M_i) \;\leq\; \max(2i,12) C_{T_i^*}(M_i)$$

Preuve.

– Si $i \geq m_0$, d'après le lemme 7, on a :

$$C_{T_i}(M_i) \ \leq \ m_i \cdot C_{T_i^*}(M_i) \ = \ (m_0 + i) C_{T_i^*}(M_i) \ \leq \ 2i \cdot C_{T_i^*}(M_i)$$

– Sinon, $i < m_0$. D'après le lemme 8, on a alors :

$$C_{T_i}(M_i) \ \leq \ \left(4 + \frac{6i}{m_0} + \frac{2i^2}{m_0^2}\right) C_{T_i^*}(M_i) \ \leq \ \left(4 + \frac{6m_0}{m_0} + \frac{2m_0^2}{m_0^2}\right) C_{T_i^*}(M_i) \ = \ 12 C_{T_i^*}(M_i)$$

On en déduit :

$$C_{T_i}(M_i) \ \leq \ \max(2i, 12) C_{T_i^*}(M_i)$$

□

L'algorithme AS a donc un rapport de compétitivité linéaire en nombre de sommets ajoutés. Si ce résulat est peu satisfaisant en termes de qualité de l'arbre construit pour la somme des distances, nous allons maintenant montrer que si les contraintes *on-line*, *arbre* et *emboîtement* sont respectées, il n'existe pas d'algorithme dont le rapport de compétitivité est meilleur (en ordre de grandeur) que celui de AS.

Borne inférieure

Dans cette section, nous montrons que si les contraintes *on-line*, *arbre* et *emboîtement* sont satisfaites, tout algorithme a un rapport de compétitivité pour la somme des distances en $\Omega(i)$ (où i est le nombre de sommets ajoutés).

Théorème 11 *Pour tout algorithme* A *respectant les contraintes* on-line, arbre *et* emboîtement, *pour tout i suffisamment grand, il existe un graphe G_p et une séquence de i ajouts tels que* A *a un rapport de compétitivité pour la somme des distances en $\Omega(i)$, c'est-à-dire que l'on a :*

$$\frac{C_{T_i}(M_i)}{C_{T_i^*}(M_i)} \in \Omega(i)$$

La preuve de ce théorème étant assez longue et figurant déjà dans [55, 56], nous ne la développons pas dans cette section. Néanmoins, dans un souci de complétude, nous la présentons en annexe (voir annexe 1).

Bilan de la section 2.2.1. Les résultats de la section 2.2.1 montrent que si nous voulons obtenir un rapport de compétitivité constant pour la somme des distances, nous devons relâcher les contraintes que nous imposons. En effet, d'après le théorème 11, il n'existe pas d'algorithme respectant les contraintes *on-line*, *arbre* et *emboîtement* dont le rapport de compétitivité est constant. Nous allons donc maintenant relâcher la contrainte *emboîtement* (voir section 2.2.2), dans le but d'obtenir une qualité constante pour la somme des distances.

2.2.2 Modèle *avec reconstructions*

Borne supérieure

L'idée principale de l'algorithme ASR (pour Ajouts pour la Somme des distances avec Reconstructions) est de définir des étapes d'ajouts particulières, appelés *étapes de reconstruction* durant lesquelles

l'arbre courant est (totalement) reconstruit (dans le but de maintenir une contrainte de qualité constante pour la somme des distances). Entre deux étapes successives de reconstruction, lorsqu'un membre est ajouté, l'arbre courant est simplement incrémenté d'un plus court chemin entre le nouveau membre et le médian du groupe correspondant à la dernière étape de reconstruction. Nous soulignons que l'idée principale de cet algorithme (par paliers de reconstructions totales était déjà présente dans l'algorithme AS, proposé en section 1.3.2).

La séquence (a_k) suivante définit les étapes de reconstruction de notre algorithme, avec $c \geq 12$ la constante de qualité requise pour la somme des distances et $m_0 = |M_0|$ la taille du groupe initial :

$$\left.\begin{array}{rcl} m_{a_0} & = & m_0 \\ m_{a_1} & = & m_{a_0}\left(1+\left\lfloor\frac{\sqrt{2c+1}-3}{2}\right\rfloor\right) \\ & \vdots & \\ m_{a_k} & = & m_{a_{k-1}}\left(1+\left\lfloor\frac{\sqrt{2c+1}-3}{2}\right\rfloor\right) \end{array}\right\} \Rightarrow\ \forall\ k \geq 0,\ m_{a_k} = m_0\left(1+\left\lfloor\frac{\sqrt{2c+1}-3}{2}\right\rfloor\right)^k$$

Algorithme Ajouts pour la Somme des distances avec Reconstructions – ASR

Soit $G = (V, E, w)$ un graphe.
Soit $M_0 \subseteq V$ le groupe initial.
À l'étape $a_0 = 0$:
    Construire un arbre de plus courts chemins T_0 couvrant M_0,
    enraciné en r_0 médian de M_0.
Après la dernière étape de reconstruction a_k :
    Soit M_{a_k+j} le groupe courant.
    Soit u_{a_k+j} le $j^{\text{ème}}$ sommet à ajouter depuis la dernière étape de reconstruction a_k.
    Si $j < m_{a_k}\left\lfloor\frac{\sqrt{2c+1}-3}{2}\right\rfloor$.
        Alors construire T_{a_k+j} couvrant $M_{a_k+j} = M_{a_k+j-1} \cup \{u_{a_k+j}\}$
        en incrémentant sans créer de cycle l'arbre T_{a_k+j} d'un plus court
        chemin de u_{a_k+j} à r_{a_k}, médian de M_{a_k+j}.
    Sinon, on a $j = m_{a_k}\left\lfloor\frac{\sqrt{2c+1}-3}{2}\right\rfloor$.
        Il s'agit d'une étape de reconstruction et
        on a $m_{a_k+j} = m_{a_k}\left(1+\left\lfloor\frac{\sqrt{2c+1}-3}{2}\right\rfloor\right) = m_{a_{k+1}}$.
        Casser l'arbre courant et construire $T_{a_{k+1}}$, un arbre de plus courts
        chemins couvrant $M_{a_{k+1}}$, enraciné en $r_{a_{k+1}}$ médian de $M_{a_{k+1}}$.
        a_{k+1} est alors la nouvelle dernière étape de reconstruction.

Application numérique. Supposons que $c = 12$ est la constante de qualité requise, $m_0 = 4$ la taille du groupe de départ et $i = 15$ le nombre de nouveaux sommets révélés.

À l'étape $a_0 = 0$, l'algorithme ASR construit un arbre de plus courts chemins enraciné en r_0 médian du groupe initial M_0, couvrant les 4 membres initiaux. Comme $c = 12$, on a $\left(1+\left\lfloor\frac{\sqrt{2c+1}-3}{2}\right\rfloor\right) = 2$. Les valeurs successives des tailles des groupes correspondant à des paliers de reconstruction sont donc : $m_{a_0} = m_0 = 4$, $m_{a_1} = 2m_{a_0} = 8$ et $m_{a_2} = 2m_{a_1} = 16$. Cela signifie que chaque nouveau membre u_1, u_2, u_3 est tel que $j < m_{a_0} = 4$ (car $j = 1, 2, 3$). Chaque nouveau membre u_1, u_2, u_3 est donc connecté à l'arbre courant par un plus court chemin à r_0 (médian de M_0).

Lorsque u_4 est révélé, on a $j = m_{a_0} = 4$. Il s'agit donc d'une étape de reconstruction. ASR calcule un médian r_4 de M_4 et construit un arbre de plus courts chemins couvrant M_4, enraciné en r_4. Lorsque chaque nouveau membre $u_5, \ldots, u_{11}$ est révélé, on a $j < m_{a_1} = 8$ (car $j = 1, \ldots, 7$). Chaque nouveau membre $u_5, \ldots, u_{11}$ est donc connecté à l'arbre courant par un plus court chemin à r_4 (médian de M_4).

Lorsque u_{12} est révélé, on a $j = m_{a_1} = 8$. Il s'agit donc d'une étape de reconstruction. ASR calcule un médian r_4 de M_4 et construit un arbre de plus courts chemins couvrant M_{12} , enraciné en r_{12}. Les 3 derniers nouveaux membres u_{13}, u_{14}, u_{15} sont tels que $j < m_{a_2} = 16$ (car $j = 1, 2, 3$). Chaque nouveau membre u_{13}, u_{14}, u_{15} est donc connecté à l'arbre courant par un plus court chemin à r_{12} (médian de M_{12}).

Remarques : les étapes de reconstruction de l'algorithme ASR sont potentiellement des étapes critiques, puisque l'arbre courant est totalement cassé puis reconstruit. Les autres étapes ne sont pas des étapes critiques puisque l'arbre courant est juste incrémenté (voir lignes 10, 11 et 12 de l'algorithme ASR) pour obtenir un nouvel arbre contenant l'arbre précédent. Nous soulignons également que la contrainte *arbre* est respectée.

Pour toute étape de reconstruction (et pour l'étape initiale), construire T_{a_k} un arbre de plus courts chemins enraciné en r_{a_k} (voir lignes 4, 5, 16, 17 et 18 de l'algorithme ASR) peut être fait en temps polynomial, en utilisant l'algorithme de Dijkstra. De plus, chaque étape non critique nécessite l'ajout d'un plus court chemin à l'arbre courant, ce qui peut être fait en temps polynomial en utilisant à nouveau l'algorithme de Dijkstra.

Nous soulignons également que contrairement à l'algorithme RDR (proposé en section 1.3.2), ASR est paramétrable. En effet, les paliers de reconstructions sont déterminés en fonction de la constante c de qualité fixée par l'utilisateur pour la somme des distances. Plus l'utilisateur exige une qualité importante (correspondant à une constante c faible), plus les paliers de reconstruction de l'algorithme ASR seront rapprochés. Il faudra donc choisir la constante c en fonction de la qualité exigée, sachant que celle-ci a une influence sur le nombre d'étapes critiques induites par l'algorithme.

L'algorithme ASR respecte la contrainte *qualité pour la somme des distances.* Le théorème 12 montre que l'algorithme ASR respecte la contrainte *qualité pour la somme des distance* de niveau $c \geq 12$ (c est fixée par l'utilisateur).

Théorème 12 *Soit $c \geq 12$ une constante quelconque fixée par l'utilisateur (représentant le niveau de qualité désiré pour la somme des distances). Pour toute séquence d'ajouts $M_0 \subset \cdots \subset M_i$, l'algorithme ASR respecte la contrainte* qualité pour la somme des distances *avec un niveau c, c'est-à-dire que l'on a :*

$$C_{T_i}(M_i) \leq c \cdot C_{T_i^*}(M_i)$$

Preuve. Soit a_k la dernière étape de reconstruction de l'algorithme ASR. Après l'étape a_k, il existe j, $0 \leq j \leq \left\lfloor \frac{\sqrt{2c+1}-3}{2} \right\rfloor m_{a_k}$ tel que $i = a_k + j$. Étant donné qu'entre deux étapes de reconstruction, l'algorithme ASR se comporte de la même manière que l'algorithme AS, nous pouvons appliquer le

lemme 8 avec M_{a_k} le groupe de départ et j le nombre de sommets ajoutés :

$$\begin{aligned}
C_{T_i}(M_i) &\leq \left(4 + \frac{6j}{m_{a_k}} + \frac{2j^2}{m_{a_k}^2}\right) C_G(M_i) \\
&\leq \left(4 + \frac{6\left\lfloor \frac{\sqrt{2c+1}-3}{2} \right\rfloor m_{a_k}}{m_{a_k}} + \frac{2\left(\left\lfloor \frac{\sqrt{2c+1}-3}{2} \right\rfloor m_{a_k}\right)^2}{m_{a_k}^2}\right) C_G(M_i) \\
&\quad (\text{car } j \leq \left\lfloor \frac{\sqrt{2c+1}-3}{2} \right\rfloor m_{a_k}) \\
&\leq \left(3\sqrt{2c+1} - 5 + \frac{\left(\sqrt{2c+1}-3\right)^2}{2}\right) C_G(M_i) \;=\; c \cdot C_G(M_i) \\
&\leq c \cdot C_{T_i^*}(M_i) \\
&\quad (\text{car pour tout arbre } T \text{ couvrant courant un} \\
&\quad \text{groupe } M \subseteq V, \text{ on a } C_G(M) \leq C_T(M))
\end{aligned}$$

□

Nous évaluons maintenant la qualité de l'algorithme ASR. Nous rappelons que l'évaluation de la qualité d'un algorithme dans le modèle *avec reconstructions* ne se mesure plus en termes de rapport de compétitivité (celui-ci étant désormais intégré dans les contraintes du modèle), mais suivant les deux critères suivants : d'une part le nombre d'étapes critiques induites et d'autre part le nombre moyen de changements élémentaires effectués par étape. Plus ces deux quantités sont faibles, meilleur est l'algorithme.

L'algorithme ASR induit un nombre d'étapes critiques logarithmique en i. Le théorème 13 montre que l'algorithme ASR induit un nombre d'étapes critiques en $O(\log i)$ (où i est le nombre de sommets ajoutés).

Théorème 13 *Pour toute séquence d'ajouts $M_0 \subset \cdots \subset M_i$, soit $T_0, \ldots, T_i$ la séquence d'arbres retournée par l'algorithme* ASR. *On a :*

$$\sharp EC(T_0, \ldots, T_i) \;=\; \left\lfloor \log_{\left(1+\left\lfloor \frac{\sqrt{2c+1}-3}{2} \right\rfloor\right)} \left(\frac{i}{m_0} + 1\right) \right\rfloor \;\in\; O(\log i)$$

Preuve. Par définition de la séquence (a_k), s'il y a p reconstructions (c'est-à-dire p étapes critiques), alors p est tel que :

$$\begin{aligned}
m_{a_p} \leq m_0 + i < m_{a_{p+1}} \quad &\Rightarrow \quad \left(1 + \left\lfloor \frac{\sqrt{2c+1}-3}{2} \right\rfloor\right)^p \leq \frac{i}{m_0} + 1 < \left(1 + \left\lfloor \frac{\sqrt{2c+1}-3}{2} \right\rfloor\right)^{p+1} \\
&\Rightarrow \quad p = \left\lfloor \log_{\left(1+\left\lfloor \frac{\sqrt{2c+1}-3}{2} \right\rfloor\right)} \left(\frac{i}{m_0} + 1\right) \right\rfloor \\
&\Rightarrow \quad p \in O(\log i) \\
&\quad (\text{car } c \geq 12 \text{ est une constante})
\end{aligned}$$

□

L'algorithme ASR effectue un nombre constant de changements élémentaires moyen par étape. Dans ce paragraphe, nous montrons que même si l'algorithme ASR induit un nombre logarithmique d'étapes critiques, il effectue au plus un nombre constant de changements élémentaires moyen par étape. Pour montrer cela, nous avons d'abord besoin du corollaire 2, (issu du lemme 3 de la section 1.3.2) ainsi que du lemme 9, résultats préliminaires au théorème 14.

Corollaire 2 *Soit i ($1 \leq i \leq m_0 - 1$) la $i^{ème}$ étape d'une séquence d'ajouts quelconque. Soient T_i l'arbre élagué couvrant le $i^{ème}$ groupe M_i construit par un algorithme quelconque et T_i^c l'arbre de connexion associé à T_i. Le nombre de changements élémentaires effectués à l'étape i, noté $\sharp CE(T_i^c)$, est :*

$$\sharp CE(T_i^c) \leq 4|M_i|$$

Preuve. D'après le lemme 3, on a $|E_{T_{i-1}^c}| \leq 2|M_{i-1}| - 3 \leq 2|M_i|$. Pour chaque étape i, chaque connexion (c'est-à-dire arête de l'arbre T_i^c) peut être cassée au plus une fois, et il y a au plus $|E_{T_i^c}| \leq 2|M_i| - 3 \leq 2|M_i|$ nouvelles connexions (arêtes) à construire. On en déduit qu'à l'étape i, n'importe quel algorithme fait au plus $|E_{T_{i-1}^c}| + |E_{T_i^c}| \leq 4|M_i|$ changements élémentaires. □

Le corollaire 2 montre entre autres que même si i est une étape critique, l'algorithme ASR effectue au plus $4|M_i|$ changements élémentaires à l'étape i.

Lemme 9 *Soit i la $i^{ième}$ étape d'une séquence d'ajouts quelconque telle que i n'est pas une étape critique. Soient T_i l'arbre retourné par l'algorithme ASR à la $i^{ième}$ étape et T_i^c l'arbre de connexion associé à T_i. On a :*

$$\sharp CE(T_i^c) \leq 4$$

Preuve. Par définition de ASR, à chaque étape i non critique (c'est-à-dire à chaque étape où l'arbre n'est pas cassé, puis reconstruit), le nouvel arbre T_i est obtenu par l'ajout d'un chemin entre le nouveau sommet u à connecter et un sommet v appartenant à l'arbre courant T_{i-1}. Nous évaluons le nombre de changements élémentaires dans T_i^c de la manière suivante :

- Si v est un sommet de connexion dans T_{i-1}^c, alors T_i^c est obtenu à partir de T_{i-1}^c par l'ajout de l'arête uv (correspondant au chemin joignant u à v dans T_i). On a donc :

$$\sharp CE(T_i^c) = 1 \leq 4$$

- Si v n'est pas un sommet de connexion dans T_{i-1}^c, alors v est un sommet de degré 2 dans T_{i-1}. T_i^c est alors obtenu à partir de T_{i-1}^c de la manière suivante : Soient a et b les sommets de connexion de l'arbre T_{k-1}^c tels que le chemin joignant a et b dans T_{i-1} contient le sommet v. L'arête ab est alors cassée. Cette première étape induit un changement élémentaire. Deux sous-cas peuvent maintenant se produire :
  - Si le nouveau sommet u appartient déjà à l'arbre T_{i-1}, alors $u = v$ et deux nouvelles arêtes au et ub sont créées pour construire T_i^c, induisant deux nouveaux changements élémentaires. On a donc :
  $$\sharp CE(T_i^c) = 1 + 2 = 3 \leq 4$$
  - Sinon, le nouveau sommet u n'appartient pas à l'arbre T_{i-1}. Dans ce cas, 3 nouvelles arêtes av, vb et uv sont créées pour construire T_i^c, induisant trois nouveaux changements élémentaires. On a donc :
  $$\sharp CE(T_i^c) = 1 + 3 = 4$$

□

Théorème 14 *Pour toute séquence d'ajouts $M_0 \subset \cdots \subset M_i$, soient $T_0, \ldots, T_i$ la séquence d'arbre retournée par l'algorithme* ASR *et $T_0^c, \ldots, T_i^c$ la séquence des arbres de connexion associée. On a :*

$$\sharp CEM(T_0^c, \ldots, T_i^c) \leq 12$$

Preuve. Deux cas peuvent se produire :

- Si $i < m_0 \left\lfloor \frac{\sqrt{2c+1}-3}{2} \right\rfloor$, alors, par définition de l'algorithme ASR, il n'y a pas d'étape critique (car pas d'étape de reconstruction). Donc, d'après le lemme 9, on a :

$$\sharp CEM(T_0^c, \ldots, T_i^c) = \frac{1}{i} \sum_{k=1}^{i} \sharp CE(T_k^c) \leq \frac{4i}{i} = 4$$

- Sinon, on a $i \geq m_0 \left\lfloor \frac{\sqrt{2c+1}-3}{2} \right\rfloor$. Par définition de l'algorithme ASR, il y a au moins une étape critique. Pour simplifier les notations, on pose $b = 1 + \left\lfloor \frac{\sqrt{2c+1}-3}{2} \right\rfloor$. Soit $\sharp CE_{\text{cr}}(T_0^c, \ldots, T_i^c)$ le nombre total de changements élémentaires effectués pendant les étapes critiques et soit $\sharp CE_{\text{ncr}}(T_0^c, \ldots, T_i^c)$ le nombre total de changements élémentaires effectués pendant les étapes non critiques. Nous allons maintenant majorer $\sharp CE_{\text{cr}}(T_0^c, \ldots, T_i^c)$ et $\sharp CE_{\text{ncr}}(T_0^c, \ldots, T_i^c)$.
  - Majoration de $\sharp CE_{\text{cr}}(T_0^c, \ldots, T_i^c)$.
    À chaque étape critique a_k, d'après le corollaire 2, on a $CE(T_k^c) \leq 4|M_{a_k}|$. De plus, par définition de l'algorithme ASR, on a $|M_{a_k}| = m_0 b^k$. Enfin, d'après le théorème 13, l'algorithme ASR induit au plus $\left\lfloor \log_b \left(\frac{i}{m_0} + 1 \right) \right\rfloor$ étapes critiques. On a donc :

$$\begin{aligned}
\sharp CE(T_0^c, \ldots, T_i^c) &\leq \sum_{k=1}^{\left\lfloor \log_b \left(\frac{i}{m_0}+1 \right) \right\rfloor} 4m_0 b^k = 4m_0 \left(\frac{b^{\left\lfloor \log_b \left(\frac{i}{m_0}+1 \right) \right\rfloor + 1} - b}{b-1} \right) \\
&\leq 4m_0 b \left(\frac{b^{\log_b \left(\frac{i}{m_0}+1 \right)} - 1}{b-1} \right) = 4m_0 b \left(\frac{\frac{i}{m_0} + 1 - 1}{b-1} \right) = \frac{4ib}{b-1} \\
&= 4i + \frac{4i}{\left\lfloor \frac{\sqrt{2c+1}-3}{2} \right\rfloor} \\
&\quad \text{(car } b = 1 + \left\lfloor \frac{\sqrt{2c+1}-3}{2} \right\rfloor \text{)} \\
&\leq 8i \\
&\quad \text{(comme } c \geq 12 \text{, on a } \left\lfloor \frac{\sqrt{2c+1}-3}{2} \right\rfloor \geq 1 \text{)}
\end{aligned}$$

  - Majoration de $\sharp CE_{\text{ncr}}(T_0^c, \ldots, T_i^c)$.
    D'après le lemme 9, on a :

$$\sharp CE_{\text{ncr}}(T_0^c, \ldots, T_i^c) \leq 4i$$

  Comme $\sum_{k=1}^{i} \sharp CE(T_k^c) = \sharp CE_{\text{cr}}(T_0^c, \ldots, T_i^c) + \sharp CE_{\text{ncr}}(T_0^c, \ldots, T_i^c)$, on obtient :

$$\sharp CEM(T_0^c, \ldots, T_i^c) = \frac{1}{i} \sum_{k=1}^{i} \sharp CE(T_k^c) \leq \frac{8i + 4i}{i} = 12$$

□

Borne inférieure

Dans cette section, nous prouvons que pour toute constante c (représentant le niveau de qualité désiré pour la somme des distances), pour tout i suffisamment grand, il existe une séquence particulière d'ajouts tel que tout algorithme respectant les contraintes *arbre* et *qualité* avec un niveau c pour la somme des distances induit un nombre d'étapes critiques au moins logarithmique en i (où i est le nombre de sommets ajoutés). Pour cela, nous décrivons d'abord le graphe G_p et la séquence particulière d'ajouts.

Description du graphe G_p. Soit $c \geq 1$ une constante quelconque. On pose $p = \lceil 100c \rceil$. Soit $G_p = (V_p, E_p, w_p)$ le graphe suivant. Pour toute arête $e \in E_p$, on a $w_p(e) = 1$ (on ne précisera donc plus w_p par la suite). G_p est le cycle $C = (V_C, E_C)$ de longueur p augmenté des éléments suivants :

- Les p sommets (resp. arêtes) du cycle C sont numérotés de s_1 à s_p (resp. e_1 à e_p) dans l'ordre d'un parcours de C, tel que l'on a $e_1 = s_p s_1$ et $\forall j,\ 2 \leq j \leq p,\ e_j = s_{j-1}s_j$.
- Chaque sommet s_j du cycle C ($1 \leq j \leq p$) est le centre d'une étoile $S_j = (V_{S_j}, E_{S_j})$ avec un nombre (fini) de feuilles aussi grand que nécessaire pour la suite.

Définition des sous-ensembles de sommets A_j^k. Pour tout j, $1 \leq j \leq p$, pour tout $k \geq 0$, on définit un ensemble de sommets A_j^k de la manière suivante :

- A_j^k est un sous-ensemble des feuilles de l'étoile S_j :

$$A_j^k \subseteq V_{S_j} \backslash \{s_j\}$$

- Les ensembles A_j^k sont deux à deux disjoints, c'est-à-dire que pour tout k_1, pour tout k_2, $k_1 \neq k_2$, $k_1 \geq 0$, $k_2 \geq 0$, on a :

$$A_j^{k_1} \cap A_j^{k_2} = \varnothing$$

- La taille de A_j^k est la suivante :

$$|A_j^0| = 2^j \text{ si } k = 0, \qquad |A_j^k| = 2^{kp+j} - 2^{(k-1)p+j} \text{ sinon.}$$

Pour tout j, $1 \leq j \leq p$, pour tout k, $0 \leq k$, on définit : $A_j^{k*} = \bigcup_{0 \leq l \leq k} A_j^l$.
On a donc :

$$|A_j^{k*}| = \sum_{l=0}^{k} |A_j^l| = |A_j^0| + \sum_{l=1}^{k} |A_j^l| = 2^j + \sum_{l=1}^{k} 2^{lp+j} - \sum_{l=0}^{k-1} 2^{lp+j} = 2^{kp+j}$$

La figure 2.1 illustre des sous-ensembles de sommets A_j^k et A_j^{k*} dans le graphe G_p (avec $p = 3$). Nous soulignons que nous ne représentons pas tous les sommets du graphe G_3, mais seulement ceux appartenant aux sous-ensembles A_1^0, A_2^0, A_3^0 et A_1^1. Nous soulignons également que le graphe G_3 est trop petit pour être de la forme $p = \lceil 100c \rceil$, mais il ne s'agit que d'une illustration.

Définition de la séquence d'ajouts $M_0 \subset \cdots \subset M_i$. Nous définissons d'abord une séquence générique d'ajouts des sommets. Nous soulignons que nous ne spécifions pas chaque étape élémentaire d'ajout, mais seulement les étapes "importantes", intéressantes pour notre analyse (que nous notons $\alpha(k, j)$). Pour tout $k \geq 1$, pour tout j, $1 \leq j \leq p = \lceil 100c \rceil$, nous définissons la séquence $\alpha(k, j)$ ($0 \leq \alpha(k, j)$) de la manière suivante :

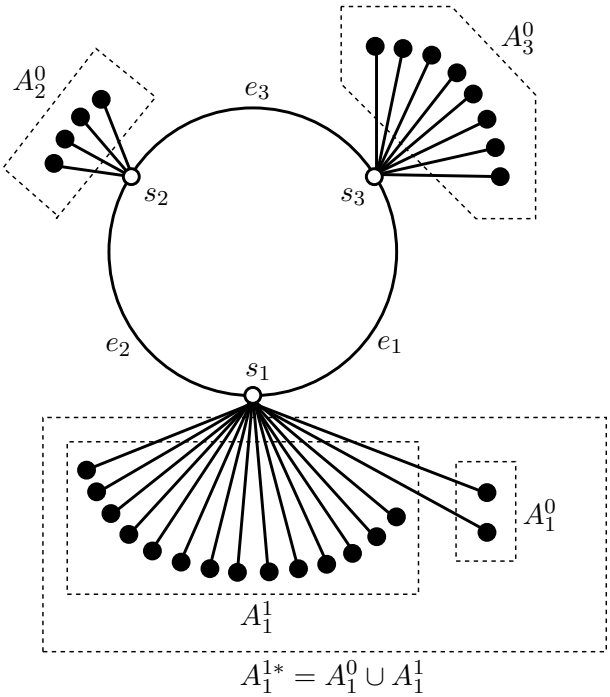


FIG. 2.1 – Les sous-ensembles A_1^0, A_2^0, A_3^0 et A_1^1 dans le graphe G_3

– À l'étape $\alpha(0,0) = 0$, on a :

$$M_0 = \emptyset$$

– À l'étape $\alpha(k,j) = \sum_{n=1}^{p} |A_n^{(k-1)*}| + \sum_{n=1}^{j} |A_n^k|$, on a :

$$M_{\alpha(k,j)} = \bigcup_{1 \le n \le p} A_n^{(k-1)*} \cup \bigcup_{1 \le n \le j} A_n^k$$

Pour tout $k \geq 1$, pour tout j, $1 \leq j \leq p$, on a $|M_{\alpha(k,j)}| = \alpha(k,j)$ avec :

$$\begin{aligned}
\alpha(k,j) &= \sum_{n=1}^{p} 2^{(k-1)p+n} + \sum_{n=1}^{j} 2^{kp+n} - \sum_{n=1}^{j} 2^{(k-1)p+n} \\
&= 2^{(k-1)p}.(2^{p+1} - 2) + 2^{kp}.(2^{j+1} - 2) - 2^{(k-1)p}.(2^{j+1} - 2) \\
&= 2^{kp+1} - 2^{(k-1)p+1} + 2^{kp+j+1} - 2^{kp+1} - 2^{(k-1)p+j+1} + 2^{(k-1)p+1} \\
&= 2^{kp+j+1} - 2^{(k-1)p+j+1}
\end{aligned}$$

On en déduit :

$$\text{Si } k < k' \text{ ou } (k = k' \text{ et } j < j'), \quad \text{alors } \alpha(k,j) < \alpha(k',j') \tag{2.1}$$

En effet, nous montrons (2.1) de la manière suivante :

– Si $k < k'$, on a :

$$\begin{aligned}\alpha(k',j') - \alpha(k,j) &= 2^{k'p+j'+1} - 2^{(k'-1)p+j'+1} - (2^{kp+j+1} - 2^{(k-1)p+j+1}) \\ &\geq 2^{k'p+2} - 2^{k'p+1} - 2^{(k+1)p+1} + 2^{(k-1)p+2} \\ &\quad (\text{car } 1 \leq j \leq p \text{ et } 1 \leq j' \leq p) \\ &\geq 2^{(k+1)p+1} - 2^{(k+1)p+1} + 2^{(k-1)p+2} \ > \ 0 \\ &\quad (\text{car } k < k+1 \leq k')\end{aligned}$$

– Si $k = k'$ et $j < j'$, on a :

$$\begin{aligned}\alpha(k',j') - \alpha(k,j) &= 2^{kp+j'+1} - 2^{(k-1)p+j'+1} - (2^{kp+j+1} - 2^{(k-1)p+j+1}) \\ &= 2^{kp+1}.(2^{j'} - 2^{j}) - 2^{(k-1)p+1}.(2^{j'} - 2^{j}) \\ &= (2^{j'} - 2^{j})(2^{kp+1} - 2^{(k-1)p+1}) \ > \ 0 \\ &\quad (\text{car } j < j')\end{aligned}$$

Nous avons spécifié avec $\alpha(k,j)$ les étapes correspondant à des étapes de "saturation" des sous-ensembles A_j^k (c'est-à-dire que $\alpha(k,j)$ est l'étape d'ajout à laquelle le dernier sommet appartenant à A_j^k est ajouté). Entre deux étapes "importantes" successives $\alpha(k,j)$ et $\alpha(k,j+1)$ (resp. $\alpha(k,p)$ et $\alpha(k+1,1)$), les sommets de A_{j+1}^k (resp. A_1^{k+1}) sont ajoutés un par un dans n'importe quel ordre. Nous soulignons que nous arrêtons d'ajouter des sommets après la dernière étape "importante", lorsque exactement i sommets ont été ajoutés. La figure 2.2 illustre la séquence d'ajouts définie ci-dessus. Dans cette illustration, les sommets sont ajoutés un par un et "saturent" les sous-ensembles A_j^k dans l'ordre suivant : A_1^0, A_2^0, A_3^0, A_4^0, A_1^1, A_2^1, A_3^1, A_4^1, A_1^2, A_2^2, A_3^2 et A_4^2.

Tout algorithme induit un nombre d'étapes critiques au moins logarithmique en i. Le lemme préliminaire suivant est central dans notre analyse. Il décrit les sous-séquences d'ajouts où *au moins* une étape critique a lieu.

Lemme 10 *Soit $c \geq 1$ une constante quelconque (représentant le niveau de qualité requis pour la somme des distances) et soit $p = \lceil 100c \rceil$. Soit G_p le graphe décrit dans cette section. Pour tout $k \geq 1$, soient $T_{\alpha(k,1)}, \ldots, T_{\alpha(k,p)}$ n'importe quels arbres couvrant respectivement $M_{\alpha(k,1)}, \ldots, M_{\alpha(k,p)}$. Si, pour tout i, $\alpha(k,1) \leq i \leq \alpha(k,p)$, on a $D_{T_i}(M_i) \leq c \cdot D_{T_i^*}(M_i)$, alors :*

$$\sharp EC(T_{\alpha(k,1)}, \ldots, T_{\alpha(k,p)}) \geq 1$$

Preuve. Nous prouvons le lemme 10 par l'absurde. Supposons qu'il existe $k \geq 1$ tel que pour tout i, $\alpha(k,1) \leq i \leq \alpha(k,p)$, la contrainte *qualité pour la somme des distances* soit satisfaite et qu'il n'existe aucune étape critique, c'est-à-dire qu'il existe $k \geq 1$ tel que pour tout i, $\alpha(k,1) \leq i \leq \alpha(k,p)$, on a $C_{T_i}(M_i) \leq c \cdot C_{T_i^*}(M_i)$ et $T_{\alpha(k,1)} \subseteq \cdots \subseteq T_{\alpha(k,p)}$. Ces $\alpha(k,p) - \alpha(k,1)$ arbres sont composés (entre autres) de toutes les arêtes du cycle C, sauf une arête, notée e_{j_c}. Nous soulignons le fait que, comme il n'y a pas d'étape critique, l'arête manquante e_{j_c} est toujours la même dans tous les $\alpha(k,p) - \alpha(k,1)$ arbres $T_{\alpha(k,1)}, T_{\alpha(k,2)}, \ldots, T_{\alpha(k,p)}$.

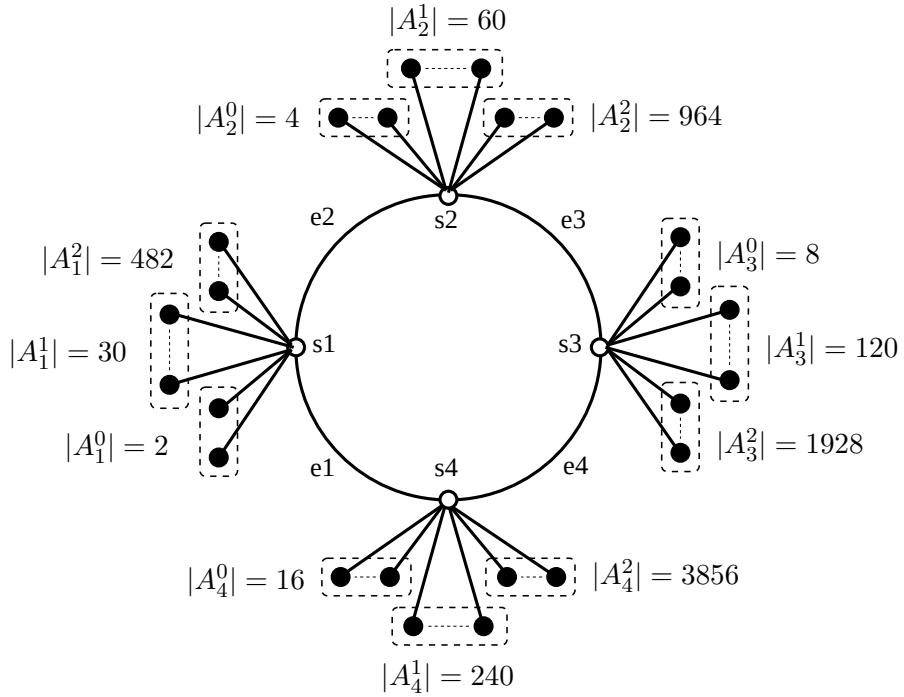


FIG. 2.2 – Illustration de la séquence d'ajouts sur le graphe G_4

Nous nous concentrons sur l'étape $\alpha(k, j_c) = 2^{kp+j_c+1} - 2^{(k-1)p+j_c+1}$. Pour simplifier les notations, on pose $i_c = \alpha(k, j_c)$. Nous avons $M_{i_c} = \bigcup_{1 \leq n \leq p} A_n^{(k-1)*} \cup \bigcup_{1 \leq n \leq j_c} A_n^k$. Nous allons maintenant minorer $C_{T_{i_c}}(M_{i_c})$ et majorer $C_{T_{i_c}^*}(M_{i_c})$ pour montrer qu'à cette étape particulière, la contrainte *qualité pour la somme des distances* n'est pas satisfaite. Cela va nous mener à une contradiction et donc prouver le lemme. Afin de simplifier les notations, pour tous groupes U et V disjoints, on pose :

$$C_{G_p}(U \leftrightarrow V) = 2 \sum_{u \in U} \sum_{v \in V} d_{G_p}(u, v)$$

Minoration de $C_{T_{i_c}}(M_{i_c})$.

– Si $j_c = 1$. Comme chaque sommet de A_1^{k*} est à une distance de $p+1$ de chaque sommet de $A_p^{(k-1)*}$ dans l'arbre T_{i_c} et comme $|A_j^{k*}| = 2^{kp+j}$, on a :

$$\begin{aligned} C_{T_{i_c}}(M_{i_c}) &\geq C_{T_{i_c}}\left(A_1^{k*} \leftrightarrow A_p^{(k-1)*}\right) \\ &\geq 2(p+1) \cdot 2^{kp+1} \cdot 2^{(k-1)p+p} \\ &= (p+1)2^{2kp+2} \\ &= (p+1)2^{2kp+2j_c} \\ &\quad (\text{car } j_c = 1) \end{aligned}$$

– Sinon, $j_c \geq 2$. Comme chaque sommet de $A_{j_c}^{k*}$ est à une distance de $p+1$ de chaque sommet de $A_{j_c-1}^{k*}$ dans l'arbre T_{i_c} et comme $|A_j^{k*}| = 2^{kp+j}$, on a :

$$\begin{aligned} C_{T_{i_c}}(M_{i_c}) &\geq C_{T_{i_c}}\left(A_{j_c}^{k*} \leftrightarrow A_{j_c-1}^{k*}\right) \\ &\geq 2(p+1).2^{kp+j_c}.2^{kp+j_c-1} \\ &= (p+1)2^{2kp+2j_c} \end{aligned}$$

Dans les deux cas, on obtient :

$$C_{T_{i_c}}(M_{i_c}) \geq (p+1)2^{2kp+2j_c} \tag{2.2}$$

Majoration de $C_{T_{i_c}}(M_{i_c})$. Pour cela, nous majorons d'abord $C_{G_p}(M_{i_c})$. Pour simplifier les calculs, on décompose $C_{G_p}(M_{i_c})$ en une somme de trois termes. $C_{G_p}(M_{i_c}) = A + B + C$ avec :

$$\begin{aligned} A &= C_{G_p}\left(\bigcup_{1\leq l\leq j_c} A_l^{k*}\right) \qquad B = C_{G_p}\left(\bigcup_{j_c+1\leq l\leq p} A_l^{(k-1)*}\right) \\ C &= C_{G_p}\left(\bigcup_{1\leq l\leq j_c} A_l^{k*} \leftrightarrow \bigcup_{j_c+1\leq l\leq p} A_l^{(k-1)*}\right) \end{aligned}$$

Majoration de A, B et C :

$$\begin{aligned} A &= \sum_{n=1}^{j_c} C_{G_p}\left(A_n^{k*}\right) + \sum_{n=1}^{j_c-1} C_{G_p}\left(A_{n+1}^{k*} \leftrightarrow \bigcup_{1\leq l\leq n} A_l^{k*}\right) \\ &\leq \sum_{n=1}^{j_c} 2^{2kp+2n+1} + \sum_{n=1}^{j_c-1}\left(2\cdot 2^{kp+n+1}\sum_{l=1}^{n} 2^{kp+l}(n-l+3)\right) \\ &\quad \text{(car } \forall\ n,\ 1\leq n\leq j_c-1 \text{ et } \forall\ l,\ 1\leq l\leq n, \text{ chaque sommet de} \\ &\quad A_l^{k*} \text{ de taille } |A_l^{k*}| = 2^{kp+l} \text{ est à une distance d'au plus } n-l+3 \\ &\quad \text{de chaque sommet de } A_{n+1}^{k*}, \text{ de taille } |A_{n+1}^{k*}| = 2^{kp+n+1}) \\ &\leq \frac{1}{3}\cdot 2^{2kp+2j_c+3} + \sum_{n=1}^{j_c-1}\left(2\cdot 2^{kp+n+1}\sum_{l=1}^{n} 2^{kp+l}(n-l+3)\right) \\ &= \frac{1}{3}\cdot 2^{2kp+2j_c+3} + \sum_{n=1}^{j_c-1}\left(2\cdot 2^{kp+n+1}\cdot 2^{kp}(2^{n+3}-2n-8)\right) \\ &\leq \frac{1}{3}\cdot 2^{2kp+2j_c+3} + \sum_{n=1}^{j_c-1} 2^{2kp+2n+5} \\ &\leq \frac{1}{3}\cdot 2^{2kp+2j_c+3} + \frac{1}{3}\cdot 2^{2kp+2j_c+5} = \frac{40}{3}\cdot 2^{2kp+2j_c} \end{aligned}$$

$$
\begin{aligned}
B &= \sum_{n=j_c+1}^{p} C_{G_p}\left(A_n^{(k-1)*}\right) + \sum_{n=j_c+1}^{p-1} C_{G_p}\left(A_{n+1}^{(k-1)*} \leftrightarrow \bigcup_{j_c+1\le l\le n} A_l^{(k-1)*}\right) \\
&\le \sum_{n=j_c+1}^{p} 2^{2(k-1)p+2n+1} + \sum_{n=j_c+1}^{p-1}\left(2\cdot 2^{(k-1)p+n+1}\sum_{l=j_c+1}^{n} 2^{(k-1)p+l}(n-l+3)\right) \\
&\quad (\text{car } \forall\ n,\ j_c+1\le n\le p \text{ et } \forall\ l,\ j_c+1\le l\le n, \text{ chaque sommet de } A_l^{(k-1)*} \\
&\quad \text{de taille } |A_l^{(k-1)*}| = 2^{(k-1)p+l} \text{ est à une distance d'au plus } n-l+3 \\
&\quad \text{de chaque sommet de } A_{n+1}^{(k-1)*}, \text{ de taille } |A_{n+1}^{(k-1)*}| = 2^{(k-1)p+n+1}) \\
&\le \frac{1}{3}\cdot 2^{2kp+3} + \sum_{n=j_c+1}^{p-1}\left(2^{(k-1)p+n+2}\sum_{l=j_c+1}^{n} 2^{(k-1)p+l}(n-l+3)\right) \\
&= \frac{1}{3}\cdot 2^{2kp+3} + \sum_{n=j_c+1}^{p-1}\left(2^{2(k-1)p+n+2}\left(2^{n+3} - 2^{j_c+1}(n-j_c+4)\right)\right) \\
&\le \frac{1}{3}\cdot 2^{2kp+3} + \sum_{n=j_c+1}^{p-1} 2^{2(k-1)p+2n+5} \le \frac{1}{3}\cdot 2^{2kp+3} + \frac{1}{3}\cdot 2^{2kp+5} \le 2^{2kp+4} \\
&\quad (\text{car } j_c \le n)
\end{aligned}
$$

$$
\begin{aligned}
C &= \sum_{n=1}^{j_c} C_{G_p}\left(A_n^{k*} \leftrightarrow \bigcup_{j_c+1\le l\le p} A_l^{(k-1)*}\right) \\
&\le \sum_{n=1}^{j_c}\left(2\cdot 2^{kp+n}\sum_{l=j_c+1}^{p}\left(2^{(k-1)p+l}(p-l+n+2)\right)\right) \\
&\quad (\text{car } \forall\ n,\ 1\le n\le j_c \text{ et } \forall\ l,\ j_c+1\le l\le p, \text{ chaque sommet de } A_l^{(k-1)*} \\
&\quad \text{de taille } |A_l^{(k-1)*}| = 2^{(k-1)p+l} \text{ est à une distance d'au plus } p-l+n+2 \\
&\quad \text{de chaque sommet de } A_n^{k*}, \text{ de taille } |A_n^{k*}| = 2^{kp+n}) \\
&= \sum_{n=1}^{j_c}\left(2^{2kp+n-p+1}\left((n+3)(2^{p+1}+2^{j_c+1}) - 2^{j_c+1}(p+j_c)\right)\right) \\
&\le \sum_{n=1}^{j_c}\left(2^{2kp+n-p+1}(2^{j_c}+2)(2^{p+1}+2^{j_c+1})\right) \\
&\quad (\text{car } \forall\ j_c,\ 1\le n\le j_c,\ n\le j_c+1\le 2^{j_c}) \\
&\le \sum_{n=1}^{j_c}\left(2^{2kp+n-p+1}\cdot 2^{p+2}(2^{j_c}+2)\right) \\
&\quad (\text{car } j_c\le p) \\
&\le 2^{2kp+j_c+4}(2^{j_c}+2) \le 2^{2kp+2j_c+5} \\
&\quad (\text{car } j_c\ge 1)
\end{aligned}
$$

On obtient donc :

$$\begin{aligned} C_{G_p}(M_{i_c}) &= A+B+C \;\leq\; \frac{40}{3}\cdot 2^{2kp+2j_c} + 2^{2kp+4} + 2^{2kp+2j_c+5} \\ &\leq \frac{40}{3}\cdot 2^{2kp+2j_c} + 2^{2kp+2j_c+2} + 2^{2kp+2j_c+5} \\ &\quad (\text{car } j_c \geq 1) \\ &= \left(\frac{40}{3}+4+32\right)\cdot 2^{2kp+2j_c} \;\leq\; 50\cdot 2^{2kp+2j_c} \end{aligned}$$

De plus, d'après [46], il existe un arbre $T_{i_c}^{\text{off}}$ couvrant M_{i_c} tel que $C_{T_{i_c}^{\text{off}}}(M_i) \leq 2C_G(M_{i_c})$. Comme $T_{i_c}^*$ est un arbre couvrant M_{i_c} optimal pour la somme des distances, on en déduit :

$$C_{T_{i_c}^*}(M_{i_c}) \;\leq\; C_{T_{i_c}^{\text{off}}}(M_{i_c}) \;\leq\; 2C_{G_p}(M_{i_c}) \;\leq\; 100\cdot 2^{2kp+2j_c} \tag{2.3}$$

D'après (2.2) et (2.3), on déduit (avec $p = \lceil 100c \rceil$) :

$$\frac{C_{T_{i_c}}(M_{i_c})}{C_{T_{i_c}^*}(M_{i_c})} \;\geq\; \frac{(p+1)2^{2kp+2j_c}}{100\cdot 2^{2kp+2j_c}} = \frac{\lceil 100c\rceil + 1}{100} \;\geq\; c + \frac{1}{100} \;> c$$

Ce résultat contredit l'hypothèse de départ qui nous dit que la contrainte *qualité pour la somme des distances* est respectée avec un niveau c. □

Le théorème suivant montre que si les contraintes *arbre* et *qualité* sont satisfaites, tout algorithme induit un nombre d'étapes critiques en $\Omega(\log i)$ dans le pire cas (où i est le nombre de sommets ajoutés).

Théorème 15 *Soit $c \geq 1$ une constante quelconque (représentant le niveau de qualité requis pour la somme des distances). Pour tout i suffisamment grand, il existe un graphe, il existe une séquence $M_0 \subset \cdots \subset M_i$ tels que tout algorithme, retournant une séquence d'arbres $T_0, \ldots, T_i$ couvrant respectivement $M_0 \subset \cdots \subset M_i$ et respectant la contrainte* qualité pour la somme des distances *avec un niveau c, induit :*

$$\sharp EC(T_0, \ldots, T_i) \in \Omega(\log i)$$

Preuve. Soit $c \geq 1$ une constante quelconque. On pose $p = \lceil 100c \rceil$. Soient $i \geq 2^{3p+1}$ le nombre de sommets ajoutés, G_p (resp. $M_0 \subset \cdots \subset M_i$) le graphe (resp. la séquence d'ajouts) défini dans cette section. Comme on a $i \geq 2^{3p+1}$, il existe $k \geq 2$ vérifiant :

$$\alpha(k+1,1) \leq i \leq \alpha(k+2,1) = 2^{(k+2)p+2} - 2^{(k+1)p+2}$$

On a donc :

$$i \leq 2^{(k+2)p+2} \quad\Rightarrow\quad \log_2 i \leq (k+2)p+2 \quad\Rightarrow\quad \frac{\log_2 i - 2}{p} - 2 \leq k$$

Comme $p = \lceil 100c \rceil$ est une constante, on obtient $k \in \Omega(\log i)$. De plus, d'après le lemme 10, on a :

$$\begin{cases} \sharp EC(T_{\alpha(1,1)}, \ldots, T_{\alpha(1,p)}) \;\geq\; 1 \\ \sharp EC(T_{\alpha(2,1)}, \ldots, T_{\alpha(2,p)}) \;\geq\; 1 \\ \qquad\qquad \vdots \\ \sharp EC(T_{\alpha(k,1)}, \ldots, T_{\alpha(k,p)}) \;\geq\; 1 \end{cases}$$

$$\begin{aligned}
&\Rightarrow \quad \sharp EC(T_{\alpha(1,1)}, \ldots, T_{\alpha(k,p)}) \ \geq \ k \\
&\Rightarrow \quad \sharp EC(T_0, \ldots, T_i) \ \geq \ k \\
&\qquad (\text{car } i \geq \alpha(k+1,1) \geq \alpha(k,p)) \\
&\Rightarrow \quad \sharp EC(T_0, \ldots, T_i) \ \in \ \Omega(\log i) \\
&\qquad (\text{car } k \in \Omega(\log i))
\end{aligned}$$

□

Remarque : nous soulignons que le théorème 15 a été obtenu sans l'utilisation d'un adversaire adaptatif. En effet, contrairement aux théorèmes 8 et 11 (voir sections 1.3.2 et 2.2.1), la séquence considérée pour prouver le théorème 15 a été choisie avant même de commencer l'exécution d'un algorithme on-line quelconque. Cette remarque est importante car elle souligne le résultat (surprenant) suivant : même si le futur est totalement connu (c'est-à-dire même si la séquence d'ajout des sommets est connue à l'avance), tout algorithme respectant les contraintes *arbre* et *qualité* pour la somme des distances induit dans le pire cas un nombre d'étapes critiques en $\Omega(\log i)$. Ce résultat peut sembler contre-intuitif. En effet, une idée naturelle serait de construire de manière totalement off-line un arbre couvrant le dernier groupe et respectant la contrainte *qualité pour la somme des distances* pour cette dernière étape. Puis, naturellement, on ajouterait un à un les sommets dans l'ordre de la séquence d'ajouts (connue dès le départ) en se calquant sur l'arbre final, c'est-à-dire en construisant de manière incrémentale une suite d'arbres couvrant les groupes intermédiaires, chaque nouvel arbre contenant l'arbre précédant, jusqu'à obtenir à la dernière étape l'arbre couvrant le dernier groupe (calculé *a priori*). Le théorème 15 montre que cette méthode (et toutes les autres n'induisant pas d'étapes critiques) ne permet(tent) pas de respecter la contrainte *qualité pour la somme des distances* à chaque étape.

Bilan de la section 2.2.2. Les théorèmes 13 et 15 montrent que l'algorithme avec reconstructions que nous proposons (ASR) permet d'obtenir un nombre optimal dans le pire cas (en ordre de grandeur) d'étapes critiques (en $O(\log i)$, avec i le nombre d'ajouts). De plus, le théorème 14 nous garantit que le nombre de changements élémentaires moyen par étape effectué par l'algorithme ASR est borné par la constante 20.

2.3 Cas des retraits seuls

Nous nous intéressons maintenant à la somme des distances lorsque l'on se restreint aux retraits on-line.

2.3.1 Modèle *sans reconstruction*

Borne supérieure

Nous allons montrer que dans le modèle *sans reconstruction*, l'algorithme RD (défini dans la section 1.3.1) est également $(m_0 - 1)-$compétitif pour la somme des distances (le théorème 3 de la section 1.3.1 montrait que RD est $(m_0 - 1)-$compétitif pour le diamètre). Le théorème suivant est un corollaire immédiat du lemme 2 (voir section 1.3.1).

Théorème 16 *L'algorithme* RD *est* $(m_0 - 1)-$*compétitif pour la somme des distances, c'est-à-dire que pour toute séquence de retraits* $M_0 \supset \cdots \supset M_i$*, on a :*

$$C_{T_i}(M_i) \leq (m_0 - 1) C_{T_i^*}(M_i)$$

Preuve. Pour toute étape i de retrait, par définition de l'algorithme RD, l'arbre T_0 contient l'arbre T_i. On a donc, d'après le lemme 2 (voir section 1.3.1), pour tout $u, v \in M_i \subseteq M_0$, $d_{T_i}(u,v) = d_{T_0}(u,v) \leq (m_0 - 1)d_G(u,v)$. On en déduit :

$$\begin{aligned} C_{T_i}(M_i) &= \sum_{u,v \in M_i} d_{T_i}(u,v) \leq (m_0 - 1) \sum_{u,v \in M_i} d_G(u,v) = (m_0 - 1)C_G(M_i) \\ &\leq (m_0 - 1)C_{T_i^*}(M_i) \\ &\quad \text{(car pour tout arbre } T \text{ couvrant courant un} \\ &\quad \text{groupe } M \subseteq V, \text{ on a } D_G(M) \leq D_T(M)) \end{aligned}$$

□

Le théorème 16 montre que l'algorithme RD est $(m_0 - 1)-$compétitif pour la somme des distances. Ce résultat est peu satisfaisant. En effet, si on veut avoir une qualité satisfaisante pour la somme des distances, cet algorithme n'est exploitable que pour le cas très restreint d'un groupe initial M_0 de taille constante.

Néanmoins, dans le pire cas, l'algorithme RD est optimal. En effet, nous allons maintenant prouver que si nous n'autorisons aucune reconstruction (c'est-à-dire si la contrainte *emboîtement* est respectée), dans le pire cas, il n'existe pas d'algorithme dont le rapport de compétitivité pour la somme des distances est meilleur que celui de RD.

Borne inférieure

Le théorème 17 montre que tout algorithme A est au moins $(m_0 - 1) -$ compétitif pour la somme des distances.

Théorème 17 *Pour tout algorithme* A *respectant les contraintes* arbre *et* emboîtement*, pour tout i, il existe un graphe et une séquence de i retraits tels que* A *est au moins* $(m_0 - 1) -$ *compétitif pour la somme des distances, c'est à dire que l'on a :*

$$C_{T_i}(M_i) \geq (m_0 - 1)C_{T_i^*}(M_i)$$

Preuve. La preuve du théorème 17 est similaire à celle du théorème 4 (voir section 1.3.1), à la différence près suivante. Les quantités à évaluer sont maintenant $C_{T_i}(M_i)$ (resp. $C_{T_i^*}(M_i)$) au lieu de $D_{T_i}(M_i)$ (resp. $D_{T_i^*}(M_i)$). Il suffit donc de remplacer $D_{T_i}(M_i) = m_0 - 1$ par $C_{T_i}(M_i) = 2(m_0 - 1)$ et $D_{T_i^*}(M_i) = 1$ par $C_{T_i^*}(M_i) = 2$. □

Bilan de la section 2.3.1. Les théorèmes 16 et 17 montrent que l'algorithme RD est optimal dans le pire cas. Néanmoins, ce résultat est peu satisfaisant, puisqu'il nous permet d'obtenir une qualité constante pour la somme des distances uniquement dans le cas très restreint d'un groupe initial M_0 de taille constante.

2.3.2 Modèle *avec reconstructions*

La version avec reconstruction du modèle restreint aux retraits pour le critère somme des distances est celle pour laquelle nous avons obtenu le moins de résultats. En effet, la seule méthode que nous sommes en mesure de proposer pour maintenir à chaque étape une qualité satisfaisante (c'est-à-dire satisfaire la contrainte *qualité pour la somme des distances* avec un niveau c constant) n'est pas raisonnable. Il s'agit de la méthode triviale qui consiste à casser la totalité de l'arbre courant à

chaque étape pour le reconstruire totalement de manière off-line (en utilisant la partie off-line de l'algorithme AS, défini en section 2.2.1), et obtenir une qualité constante $c = 2$ pour la somme des distances (d'après le corollaire 10 de la section 2.2.1, avec $i = 0$). Bien sûr, cette méthode n'est pas satisfaisante puisqu'elle induit un nombre d'étapes critiques linéaire en nombre de sommets retirés (le pire cas possible) et un nombre de changements élémentaires moyen par étape également linéaire en nombre de sommets retirés (à nouveau le pire cas possible).

En termes de borne inférieure, le seul résultat que nous avons pour l'instant est le théorème suivant, montrant que tout algorithme respectant les contraintes *arbre* et *qualité pour la somme des distances* induit un nombre d'étapes critiques au moins logarithmique en i dans le pire cas (où i est le nombre de sommets retirés).

Théorème 18 *Soit $c \geq 1$ une constante quelconque (représentant le niveau de qualité requis pour la somme des distances). Pour tout i suffisamment grand, il existe un graphe, il existe une séquence $M_0 \supset \cdots \supset M_i$ tels que tout algorithme, retournant une séquence d'arbres $T_0, \ldots, T_i$ couvrant respectivement $M_0 \supset \cdots \supset M_i$ et respectant la contrainte* qualité pour la somme des distances *avec un niveau c, induit :*

$$\sharp EC(T_0, \ldots, T_i) \in \Omega(\log i)$$

Preuve. Le théorème 18 se prouve de la même façon que le théorème 15 (voir section 2.2.2), à ceci près que nous considérons ici une séquence de retraits ($M_0 \supset \cdots \supset M_i$) au lieu d'une séquence d'ajouts ($M_0 \subset \cdots \subset M_i$). Il suffit alors de considérer le même graphe particulier que celui défini en section 2.2.2 ainsi que la séquence inverse à celle définie (également) en section 2.2.2. C'est-à-dire que si on note $M_0 \subset \cdots \subset M_i$ la séquence d'ajouts de la section 2.2.2, la séquence de retraits que nous considérons ici est $M'_0 \supset \cdots \supset M'_i$, avec $M'_0 = M_i, \ldots, M'_i = M_0$. Une fois cette différence identifiée, la preuve du théorème 18 est identique à celle du théorème 15. □

Remarque : nous soulignons que les preuves des théorèmes 15 et 18 ne dépendent pas du sens dans lequel la séquence d'ajouts ou de retraits est déroulée, uniquement parce que cette séquence est déterminée *à l'avance* (et non au fur et à mesure de l'exécution par un adversaire adaptatif), comme nous l'avons déjà fait remarquer à la fin de la section 1.3.2. Nous apportons cette précision pour souligner que dans le cas ou la séquence est construite au fur et à mesure par un adversaire adaptatif (comme par exemple pour la séquence ayant servi à démontrer le théorème 8 de la section 1.3.2), il n'est pas possible de la dérouler dans le sens inverse, puisque par définition, elle n'est pas connue à l'avance.

Bilan de la section 2.3.2. Les résultats de cette section ne sont pas totalement satisfaisants, puisqu'il reste un écart important entre la borne supérieure (obtenue par la méthode triviale qui consiste à reconstruire l'arbre à chaque étape) et la borne inférieure (théorème 18) sur le nombre d'étapes critiques (respectivement en $O(i)$ et $\Omega(i)$, où i est le nombre de sommets retirés). Une perspective immédiate de travail consiste donc à trouver une méthode permettant de respecter les contraintes *arbre* et *qualité pour la somme des distances* en reconstruisant moins souvent l'arbre courant ou/et trouver une borne inférieure sur le nombre d'étapes critiques (valable pour tout algorithme respectant les contraintes *arbre* et *qualité pour la somme des distances*) plus grande que $\Omega(\log i)$, jusqu'à obtenir le même ordre de grandeur pour les bornes inférieures et supérieures.

Chapitre 3

Étude du coût de l'incrémentalité

Dans ce chapitre, nous proposons de relâcher au maximum les contraintes de notre problème de départ. En effet, nous n'allons maintenant conserver que la notion d'*incrémentalité* du problème (c'est-à-dire l'ajout un par un de chaque nouveau sommet) et relâcher les contraintes *on-line* et *arbre*. Le nouveau problème consiste alors à construire une séquence d'ajouts avec garanties sur les diamètres des groupes successifs. L'intérêt de ce problème est de proposer un nouvel éclairage sur la notion d'incrémentalité. En effet, les difficultés liées à la non connaissance du futur et à la construction d'une structure de connexion (contraintes *arbre* et *on-line*, traitées dans les chapitres 2 et 3) sont maintenant relâchées, pour ne garder comme contrainte que l'inclusion des groupes successifs. Les travaux présentés dans ce chapitre ont été réalisés en collaboration avec Ralf Klasing et Joseph Peters (voir [44]).

3.1 Présentation du problème et définitions.

Nous présentons maintenant plus précisément le problème que nous allons traiter dans ce chapitre. Étant donné un graphe $G = (V, E, w)$ valué positivement, nous devons construire une séquence de $n = |V|$ sous-ensembles de sommets $M_1, \ldots, M_n$ dont les diamètres respectifs dans le graphe sont petits. La contrainte dite d'*incrémentalité* que nous imposons est la suivante : les n groupes que nous devons construire doivent être tels que $M_1 \subset M_2 \subset \cdots \subset M_n = V$. Nous rappelons que nous notons $D_G(M) = \max\{d_G(u, v) : u, v \in M\}$ le diamètre du groupe M dans le graphe G. Pour définir le *coût* d'une séquence (mesuré par le ratio maximum entre le diamètre de chaque groupe M_i et celui d'un groupe à i sommets et de diamètre minimum), nous avons besoin de la définition suivante.

Définition 13 (Groupe de diamètre minimum) *Un* groupe de taille i, $1 \leq i \leq n$, de diamètre minimum *est un groupe* $N_i^* \subseteq V$ *avec* $|N_i^*| = i$*, et satisfaisant :*

$$D_G(N_i^*) = \min\{D_G(M) : M \subseteq V, |M| = i\}$$

Notre but est alors de construire une séquence de groupes $M_1, M_2, \ldots, M_n$ telle que $M_1 \subset M_2 \subset \cdots \subset M_n$ et où chaque M_i a un diamètre aussi proche que possible du diamètre optimal (c'est-à-dire le diamètre de N_i^*). Nous mesurons la qualité d'une séquence incrémentale $M_1, M_2, \ldots, M_n$ par son coût, défini de la manière suivante.

Définition 14 (Coût d'une séquence incrémentale) *Une* séquence incrémentale *est une séquence de groupes* $M_1, M_2, \ldots, M_n$ *telle que* $M_1 \subset M_2 \subset \cdots \subset M_n = V$ *et* $|M_i| = i$ *(*$1 \leq i \leq n$*). Nous*

définissons le coût d'une séquence incrémentale $M_1, M_2, \ldots, M_n$ *par :*

$$\text{coût}(M_1, \ldots, M_n) = \max_{2 \leq i \leq n} \left\{ \frac{D_G(M_i)}{D_G(N_i^*)} \right\}$$

Puisque nous comparons le diamètre des groupes incrémentaux successifs à celui des groupes de diamètre minimum non contraints d'être incrémentaux, ce coût mesure l'impact sur le diamètre de la contrainte d'*incrémentalité*. Nous définissons maintenant la séquence incrémentale optimale que nous voulons obtenir de la manière suivante.

Définition 15 (Séquence incrémentale optimale) *Une* séquence incrémentale optimale *est une séquence incrémentale $N_1^{opt}, N_2^{opt}, \ldots, N_n^{opt}$ de coût minimum, c'est-à-dire vérifiant :*

$$\text{coût}(N_1^{opt}, \ldots, N_n^{opt}) = \min \left\{ \text{coût}(M_1, \ldots, M_n) : \begin{array}{l} M_1 \subset \cdots \subset M_n = V, \\ |M_i| = i, 1 \leq i \leq n \end{array} \right\}$$

L'approche que nous proposons dans ce chapitre diffère de l'approche on-line classique que nous avions adoptée jusqu'ici. En effet, si la notion d'incrémentalité est commune dans les deux cas, nous traitons maintenant un problème où toutes les données sont connues à l'avance (donc *off-line*). De plus, la séquence que nous devons construire n'est pas imposée (comme dans le cas on-line traité dans les deux précédents chapitres), mais *choisie* par l'algorithme lui-même.

Application. Nous soulignons qu'au-delà de l'intérêt théorique d'évaluer le coût de la contrainte d'*incrémentalité*, une séquence incrémentale optimale peut être utilisée pour l'application suivante. Supposons un calcul réparti entre les machines (identiques) d'un réseau point à point. Au début, le calcul ne nécessite que peu de ressources, et n'utilise donc qu'une seule machine. Puis, au fur et à mesure, les besoins en ressources augmentent. De nouvelles machines doivent alors être ajoutées une à une, donnant ainsi une séquence incrémentale de groupes de machines. Ces machines devant communiquer entre elles pour échanger des données et des résultats partiels, la performance totale du système dépend de la latence entre les machines du groupe courant. La latence maximum dans un groupe étant son diamètre, une séquence incrémentale optimale donnera la meilleure performance dans le pire cas.

Plan du chapitre. Dans la section 3.2, nous donnons des bornes générales (supérieure et inférieure) sur le coût d'une séquence incrémentale optimale. Dans la section 3.3, nous prouvons que construire une séquence incrémentale optimale ne peut pas être approché avec un rapport d'approximation inférieur à 2, sauf si $P = NP$ (voir [8, 36, 37, 64] pour des précisions sur les algorithmes d'approximation). Enfin, dans la section 3.4, nous développons un algorithme polynomial 4-approché pour notre problème (la construction d'une séquence incrémentale optimale), et nous montrons que cette borne est atteinte.

3.2 Bornes générales sur le coût d'une séquence incrémentale

Le théorème suivant montre que pour tout graphe dont le poids des arêtes est au moins 1, le coût d'une séquence incrémentale optimale est au plus égal à la racine carrée du diamètre du graphe.

Théorème 19 *Pour tout graphe $G = (V, E, w)$ tel que pour toute arête $e \in E$, $w(e) \geq 1$, on a :*

$$\text{coût}(N_1^{opt}, \ldots, N_n^{opt}) \leq \sqrt{D_G(V)}$$

Preuve. Soit $G = (V, E, w)$ un graphe tel que pour toute arête $e \in E,\ w(e) \geq 1$. Pour tout i, $1 \leq i \leq n$, soit N_i^* un groupe de taille i de diamètre minimum et soit i_0 le plus grand entier tel que $D_G(N_{i_0}^*) \leq \sqrt{D_G(V)}$. Comme on a $\forall e \in E,\ w(e) \geq 1$, on obtient :

$$1 \leq D_G(N_2^*) \leq \cdots \leq D_G(N_{i_0}^*) \leq \sqrt{D_G(V)} < D_G(N_{i_0+1}^*) \leq \cdots \leq D_G(N_n^*) \tag{3.1}$$

Soit $M_1, M_2, \ldots, M_n$ une séquence incrémentale quelconque telle que $M_{i_0} = N_{i_0}^*$. On a alors :

$$1 \leq D_G(M_2) \leq \cdots \leq D_G(M_{i_0}) = D_G(N_{i_0}^*) \leq \sqrt{D_G(V)} \tag{3.2}$$

Comme le diamètre du graphe $G = (V, E, w)$ est $D_G(V)$, on a :

$$D_G(M_{i_0+1}) \leq \cdots \leq D_G(M_n) \leq D_G(V) \tag{3.3}$$

D'après (3.1) et (3.2), on a $\max_{2 \leq i \leq i_0} \left\{ \frac{D_G(M_i)}{D_G(N_i^*)} \right\} \leq \sqrt{D_G(V)}$; d'après (3.1) et (3.3) on a $\max_{i_0+1 \leq i \leq n} \left\{ \frac{D_G(M_i)}{D_G(N_i^*)} \right\} \leq \frac{D_G(V)}{\sqrt{D_G(V)}} = \sqrt{D_G(V)}$. On obtient donc :

$$\text{coût}(N_1^{opt}, \ldots, N_n^{opt}) \ \leq \ \text{coût}(M_1, \ldots, M_n) \ = \ \max_{2 \leq i \leq n} \left\{ \frac{D_G(M_i)}{D_G(N_i^*)} \right\} \ \leq \ \sqrt{D_G(V)}$$

□

Le théorème suivant montre qu'il existe des graphes dont le poids des arêtes est au moins 1, tels que le coût d'une séquence incrémentale optimale est au moins égal à la racine carrée du diamètre du graphe.

Théorème 20 *Il existe une famille infinie de graphes $G_K = (V_K, E_K, w_K)$ tels que pour toute arête $e \in E_K$, $w_K(e) \geq 1$, satisfaisant :*

$$\textit{coût}(N_1^{opt}, \ldots, N_n^{opt}) \geq \sqrt{D_{G_K}(V_K)}$$

Preuve. Pour toute constante $K > 1$, soit $G_K = (V_K, E_K, w_K)$ le graphe défini par la figure 3.1. Le diamètre de G_K est $D_{G_K}(V_K) = K^2$. Pour tout i, $1 \leq i \leq 5$, soit N_i^* un groupe de taille i de diamètre minimum. Soit $M_1, M_2, \ldots, M_5$ une séquence incrémentale quelconque. Deux cas sont à distinguer :

- Si $M_2 \neq \{a, b\}$, alors on a :
$$\frac{D_{G_K}(M_2)}{D_{G_K}(N_2^*)} \geq \frac{K}{1} = K$$
- Sinon $M_2 = \{a, b\}$. Pour tout groupe M_3 tel que $M_2 \subset M_3$, on a :
$$\frac{D_{G_K}(M_3)}{D_{G_K}(N_3^*)} = \frac{K^2}{K} = K$$

On obtient donc :

$$\text{coût}(M_1, \ldots, M_5) \geq K = \sqrt{D_{G_K}(V_0)}$$

Cette preuve est facilement généralisable à tout graphe complet où toutes les arêtes ont un poids de K^2, sauf un triangle avec des arêtes de poids K et une paire de sommets disjoints du triangle et connectés entre eux par une arête de poids 1. □

Les théorèmes 19 et 20 nous donnent un encadrement (tendu dans le pire cas) du coût d'une séquence incrémentale optimale pour tous les graphes dont le poids des arêtes est au moins 1.

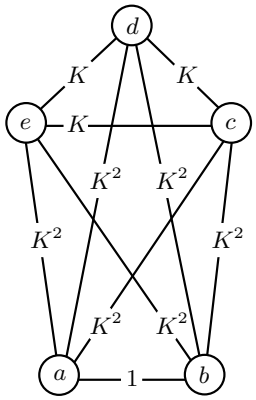


FIG. 3.1 – Le graphe G_K

3.3 Non-approximabilité du problème

Dans cette section, nous étudions la complexité algorithmique de la construction d'une séquence incrémentale optimale. Pour cela, nous définissons plus formellement deux variantes du problème (la première dans des graphes non valués, la deuxième dans des graphes valués).

SÉQUENCE INCRÉMENTALE NON VALUÉE

INSTANCE : Un graphe $G = (V, E)$.
SOLUTION : Une séquence incrémentale de groupes $M_1, M_2, \ldots, M_n$ dans G, c'est-à-dire $M_1 \subset M_2 \subset \cdots \subset M_n = V$ avec pour tout i, $1 \leq i \leq n$, $|M_i| = i$.
MESURE : coût$(M_1, \ldots, M_n)$.

SÉQUENCE INCRÉMENTALE VALUÉE

INSTANCE : Un graphe valué $G = (V, E, w)$ avec $\forall e \in E$, $w(e) > 0$.
SOLUTION : Une séquence incrémentale de groupes $M_1, M_2, \ldots, M_n$ dans G, c'est-à-dire $M_1 \subset M_2 \subset \cdots \subset M_n = V$ avec pour tout i, $1 \leq i \leq n$, $|M_i| = i$.
MESURE : coût$(M_1, \ldots, M_n)$.

Le théorème suivant montre qu'il n'existe pas d'algorithme d'approximation en temps polynomial avec un rapport d'approximation plus petit que 2 pour le problème de la construction de $N_1^{opt}, \ldots, N_n^{opt}$, sauf si $P = NP$.

Théorème 21 *Il n'existe pas d'algorithme d'approximation en temps polynomial avec un rapport d'approximation plus petit que 2 pour* SÉQUENCE INCRÉMENTALE NON VALUÉE*, sauf si* $P = NP$.

Preuve. Nous allons montrer que s'il existe un algorithme polynomial 2-approché pour le problème SÉQUENCE INCRÉMENTALE NON VALUÉE, alors on peut construire en temps polynomial une clique de taille maximum (dit problème CLIQUE MAXIMUM) de n'importe quel graphe, et donc résoudre en temps polynomial le problème de décision CLIQUE NP-complet (voir [29]).

Soit $G = (V, E)$ un graphe quelconque. Soit le sommet $r' \notin V$ et le graphe $G' = (V', E')$ tels que $V' = V \cup \{r'\}$ et $E' = E \cup \{ur' : u \in V\}$. On note $n' = |V'|$. Construire G' à partir de G peut être fait en temps polynomial. Nous allons montrer qu'à partir de n'importe quelle solution au problème SÉQUENCE INCRÉMENTALE NON VALUÉE avec un rapport d'approximation plus

petit que 2 dans G', on peut construire en temps polynomial une solution au problème CLIQUE MAXIMUM dans G.

Soit $M_1, \ldots, M_{n'}$ une séquence incrémentale pour le graphe G'. Pour tout i, $1 \leq i \leq n'$, soit N_i^* un groupe de taille i de diamètre minimum dans G'. Soit $V' = \{v'_1, v'_2, \ldots, v'_{n'}\}$ tel que $\{v'_1, \ldots, v'_{i_0}\}$ est une clique maximum de G' (de taille i_0). Considérons la séquence incrémentale $N_1^{opt}, \ldots, N_{n'}^{opt}$ dans G' obtenue en ajoutant les sommets de G' dans l'ordre suivant : $v'_1, v'_2, \ldots, v'_{n'}$. Comme G' est un graphe non valué de diamètre au plus 2, tout sous-ensemble de sommets $S \subseteq V'$ avec $|S| \geq 2$ est tel que $D_{G'}(S) = 1$ ou bien $D_{G'}(S) = 2$. On en déduit :

$$\forall i,\ 2 \leq i \leq i_0,\ D_{G'}(N_i^{opt}) = D_{G'}(N_i^*) = 1 \quad \text{et} \quad \forall i,\ i_0 + 1 \leq i \leq n',\ D_{G'}(N_i^{opt}) = D_{G'}(N_i^*) = 2$$

Pour tout i, $2 \leq i \leq n'$, la séquence incrémentale $N_1^{opt}, \ldots, N_{n'}^{opt}$ satisfait donc $D_{G'}(N_i^{opt}) = D_{G'}(N_i^*)$. On obtient alors :

$$\frac{\max_{2 \leq i \leq n'} \left\{ \frac{D_{G'}(M_i)}{D_{G'}(N_i^*)} \right\}}{\max_{2 \leq i \leq n'} \left\{ \frac{D_{G'}(N_i^{opt})}{D_{G'}(N_i^*)} \right\}} = \max_{2 \leq i \leq n'} \left\{ \frac{D_{G'}(M_i)}{D_{G'}(N_i^*)} \right\}$$

Dans G', tout sous-ensemble $S \subseteq V'$ avec $|S| \geq 2$ est tel que $D_{G'}(S) = 1$ ou bien $D_{G'}(S) = 2$, les deux seules valeurs possibles de $\max_{2 \leq i \leq n'} \left\{ \frac{D_{G'}(M_i)}{D_{G'}(N_i^*)} \right\}$ sont donc 1 et 2. Cela signifie que résoudre le problème SÉQUENCE INCRÉMENTALE NON VALUÉE avec un rapport d'approximation plus petit que 2 dans G' revient à retourner une séquence incrémentale $M_1, \ldots, M_{n'}$ satisfaisant $\max_{2 \leq i \leq n'} \left\{ \frac{D_{G'}(M_i)}{D_{G'}(N_i^*)} \right\} = 1$, et donc telle que pour tout i, $2 \leq i \leq n'$, on a $D_{G'}(M_i) = D_{G'}(N_i^*)$. Choisir le plus grand entier i_0 tel que $D_{G'}(M_{i_0}) = D_{G'}(N_{i_0}^*) = 1$ permet donc de construire une clique de taille maximum dans G' (c'est-à-dire M_{i_0}), et donc une clique de taille maximum dans G (c'est-à-dire $M_{i_0} \setminus \{r'\}$). □

Le corollaire suivant se déduit immédiatement du théorème 21.

Corollaire 3 *Il n'existe pas d'algorithme d'approximation en temps polynomial avec un rapport d'approximation plus petit que 2 pour* SÉQUENCE INCRÉMENTALE VALUÉE*, sauf si* $P = NP$.

3.4 Un algorithme $4-$approché

Dans cette section, nous proposons d'abord un algorithme pour trouver en temps polynomial une séquence incrémentale de groupes à faibles *excentricités*. Nous prouvons ensuite que cet algorithme est $4-$approché pour le problème SÉQUENCE INCRÉMENTALE VALUÉE. Pour cela, nous avons besoin des définitions suivantes.

Définition 16 (Excentricité) *L'*excentricité *d'un groupe* $M \subseteq V$ *de* racine $r \in M$ *est définie par :*

$$E(M, r) = \max\{d_G(u, r) : u \in M\}$$

Définition 17 (Groupe d'excentricité minimum) *Un groupe* $M_i^* \subseteq V$ *avec* $|M_i^*| = i$ *(*$1 \leq i \leq n$*) est un* groupe de taille i d'excentricité minimum *s'il existe un sommet* $r_i^* \in M_i^*$ *(appelé sa* racine associée*) tel que :*

$$E(M_i^*, r_i^*) = \min\{E(M, r) : M \subseteq V, |M| = i, r \in M\}$$

Définition 18 (Séquence incrémentale optimale pour l'excentricité)
Une séquence incrémentale optimale pour l'excentricité *est une séquence incrémentale de groupes* $M_1^{opt} = \{r^{opt}\}, M_2^{opt}, \ldots, M_n^{opt} = V$ *avec* $|M_i^{opt}| = i$ $(1 \leq i \leq n)$*, telle que :*

$$\max_{2\leq i\leq n}\left\{\frac{E(M_i^{opt}, r^{opt})}{E(M_i^*, r_i^*)}\right\} = \min\left\{\max_{2\leq i\leq n}\left\{\frac{E(M_i', r')}{E(M_i^*, r_i^*)}\right\} : \begin{array}{l} M_1' \subset \cdots \subset M_n' = V, \\ |M_i'| = i, M_1' = \{r'\} \end{array}\right\}$$

Définition de l'algorithme SEM**.** Pour définir l'algorithme SEM (pour Séquence d'Excentricité Minimum), nous avons besoin de la définition suivante et de l'algorithme intermédiaire EM_i (pour Excentricité Minimum), qui, pour tout i $(1 \leq i \leq n = |V|)$ renvoie un groupe de taille i d'excentricité minimum.

Définition 19 (Sous-ensemble en largeur à partir de la racine) *Soient* $r \in V$ *et* S *la séquence de valeurs* $\{d_G(r,u) : u \in V\}$ *triées par ordre croissant (on souligne que* $|S| \leq n = |V|$*). Soit la partition* $F_1(r), \ldots, F_n(r)$ *de* V *telle que pour tout* j*,* $1 \leq j \leq n$*, on a* $F_j(r) = \{u : d_G(r,u)$ *est la* $j^{ème}$*valeur dans* $S\}$*. Un groupe* $M \subseteq V$ *est un* sous-ensemble en largeur à partir de la racine $r \in M$ *s'il satisfait :*

Si $|M| = 1$*, alors* $M = \{r\}$*.*

Si $|M| \geq 2$*, alors il existe* $k \geq 2$ *tel que :*

$$\forall j,\ 1 \leq j \leq k-1,\ F_j(r) \cap M = F_j(r),$$

$$F_k(r) \cap M \neq \emptyset,$$

$$\forall l > k,\ F_l(r) \cap M = \emptyset.$$

La figure 3.2 donne une illustration de la partition $F_1(r), \ldots, F_n(r)$ de V dans un graphe $G = (V, E, w)$. Dans cet exemple, $M = \{r, a, b\}$ est un sous-ensemble en largeur à partir de la racine r.

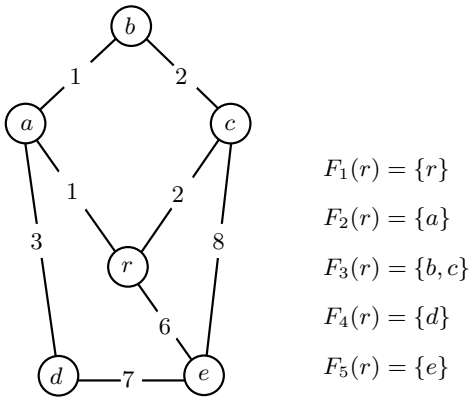


FIG. 3.2 – La partition $F_1(r), \ldots, F_n(r)$ de V dans un graphe $G = (V, E, w)$

Excentricité Minimum − EM_i

1 Soit $G = (V, E, w)$ un graphe.
2 Soit i, $1 \leq i \leq n = |V|$.
3 Pour chaque sommet $r \in V$:
4 Construire un sous-ensemble en largeur $M_i(r) \subseteq V$
5 à partir de la racine r avec $|M_i(r)| = i$.
6 Soient le sommet r_i et son groupe associé $M_i(r_i)$
7 tel que $E(M_i(r_i), r_i) = \min\{E(M_i(r), r) : r \in V\}$.
8 Retourner r_i et $M_i(r_i)$.

Remarque : nous soulignons que pour tout sommet $r \in V$, la partition $F_1(r), \ldots, F_n(r)$ et son groupe associé $M_i(r)$ peuvent être construits en temps polynomial en utilisant l'algorithme de Dijkstra (voir lignes 4 et 5 de l'algorithme EM_i). Ainsi, r_i et $M_i(r_i)$ peuvent être trouvés en temps polynomial.

Le lemme suivant montre que l'algorithme EM_i construit un groupe de taille i d'excentricité minimum. L'idée de la preuve est de montrer que pour un sommet $r \in V$ donné, le groupe de taille i d'excentricité minimum associé à r est un sous-ensemble en largeur à partir de r. Comme l'algorithme EM_i vérifie chaque racine $r \in V$ possible, EM_i renvoie nécessairement le minimum.

Lemme 11 *Pour tout i, $1 \leq i \leq n$, l'algorithme EM_i construit un groupe de taille i d'excentricité minimum.*

Preuve. Soit i, $1 \leq i \leq n$. Pour tout sommet $r \in V$, soit $M'_i(r) \subseteq V$ un groupe quelconque de taille i avec $r \in M'_i(r)$. Soit $M''_i(r) \subseteq V$ un sous-ensemble en largeur de taille i à partir de la racine r. Pour tout $r \in V$, on a donc :

$$E(M''_i(r), r) = \max\left\{d_G(u, r) : u \in M''_i(r)\right\} \leq \max\left\{d_G(v, r) : v \in M'_i(r)\right\} = E(M'_i(r), r)$$

On obtient alors :

$$\min\left\{E(M''_i(r), r) : r \in V\right\} \;\leq\; \min\left\{E(M'_i(r), r) : r \in V\right\}. \tag{3.4}$$

Soit $M_i(r_i)$ un groupe de taille i retourné par l'algorithme EM_i. Soit M^*_i un groupe de taille i d'excentricité minimum et de racine associée $r^*_i \in M^*_i$. Par définition de l'algorithme EM_i, on a $E(M_i(r_i), r_i) = \min\{E(M''_i(r), r) : r \in V\}$ et par définition de M^*_i, on a $E(M^*_i, r^*_i) = \min\{E(M'_i(r), r) : r \in V\}$. D'après (3.4), on obtient donc :

$$E(M_i(r_i), r_i) \leq E(M^*_i, r^*_i)$$

Enfin, comme M^*_i est un groupe de taille i d'excentricité minimum, on a :

$$E(M^*_i, r^*_i) = E(M_i(r_i), r_i)$$

□

Nous pouvons maintenant définir l'algorithme SEM, puis prouver que celui-ci construit une séquence incrémentale d'excentricité minimum (voir le lemme 12).

Excentricité Minimum – SEM

1 Soit $G = (V, E, w)$ un graphe.
2 Pour chaque sommet $r \in V$, pour chaque i, $2 \leq i \leq n = |V|$:
4 Construire un sous-ensemble en largeur $M_i(r) \subseteq V$
5 à partir de la racine r avec $|M_i(r)| = i$.
6 Calculer $\frac{E(M_i(r),r)}{E(M_i^*,r_i^*)}$ en utilisant l'algorithme EM_i pour obtenir $E(M_i^*, r_i^*)$.
7 Choisir $r_0 \in V$ et sa séquence associée $M_1(r_0), \ldots, M_n(r_0)$ telles que :
8 $\max_{2\leq i\leq n} \left\{ \frac{E(M_i(r_0),r_0)}{E(M_i^*,r_i^*)} \right\} = \min \left\{ \max_{2\leq i\leq n} \left\{ \frac{E(M_i(r),r)}{E(M_i^*,r_i^*)} \right\} : r \in V \right\}$

Remarques. nous soulignons que pour tout sommet $r \in V$, la séquence associée $M_1(r), \ldots, M_n(r)$ peut être construite en temps polynomial en utilisant l'algorithme de Dijkstra (voir lignes 2, 3 et 4 de l'algorithme SEM). De plus, pour tout sommet $r \in V$, pour tout i, $2 \leq i \leq n$, le ratio $\frac{E(M_i(r),r)}{E(M_i^*,r_i^*)}$ peut être calculé en temps polynomial en utilisant l'algorithme EM_i (voir ligne 6 de SEM). r_0 et sa séquence $M_1(r_0), \ldots, M_n(r_0)$ associée peuvent donc être trouvées en temps polynomial.

Lemme 12 *L'algorithme* SEM *renvoie une séquence incrémentale d'excentricité minimum.*

Preuve. Soit $M_1(r_0) = \{r_0\}, M_2(r_0), \ldots, M_n(r_0)$ la séquence incrémentale retournée par SEM, soit $M_1^{opt} = \{r^{opt}\}, M_2^{opt}, \ldots, M_n^{opt}$ la séquence incrémentale d'excentricité minimum, et pour tout i, $1 \leq i \leq n$, soit M_i^* un groupe de taille i d'excentricité minimum et $r_i^* \in M_i^*$ sa racine associée. L'algorithme SEM construit une séquence incrémentale à partir de toutes les racines possibles, et donc en particulier la séquence $M_1(r^{opt}), \ldots, M_n(r^{opt})$ à partir de la racine $\{r^{opt}\} = M_1(r^{opt})$. De plus, par définition de l'algorithme SEM, les groupes $M_1(r^{opt}), \ldots, M_n(r^{opt})$ sont des sous-ensembles en largeur à partir de la racine r^{opt}. Pour tout i ($1 \leq i \leq n$), on a donc $E(M_i(r^{opt}), r^{opt}) \leq E(M_i^{opt}, r^{opt})$. On obtient alors :

$$\max_{2\leq i\leq n} \left\{ \frac{E(M_i(r^{opt}), r^{opt})}{E(M_i^*, r_i^*)} \right\} \leq \max_{2\leq i\leq n} \left\{ \frac{E(M_i^{opt}, r^{opt})}{E(M_i^*, r_i^*)} \right\}$$

Par définition de l'algorithme SEM (voir lignes 7 et 8), et d'après le fait que la séquence incrémentale $M_1^{opt}, \ldots, M_n^{opt}$ est minimum pour l'excentricité, on obtient :

$$\max_{2\leq i\leq n} \left\{ \frac{E(M_i(r_0), r_0)}{E(M_i^*, r_i^*)} \right\} = \max_{2\leq i\leq n} \left\{ \frac{E(M_i^{opt}, r^{opt})}{E(M_i^*, r_i^*)} \right\}$$

□

L'algorithme SEM est 4-approché. Le théorème suivant montre que l'algorithme SEM est 4-approché pour le problème SÉQUENCE INCRÉMENTALE VALUÉE.

Théorème 22 *Soit $M_1, \ldots, M_n$ la séquence incrémentale retournée par l'algorithme* SEM *et soit $N_1^{opt}, \ldots, N_n^{opt}$ une séquence incrémentale optimale pour le diamètre. On a alors :*

$$\frac{\text{coût}(M_1, \ldots, M_n)}{\text{coût}(N_1^{opt}, \ldots, N_n^{opt})} \leq 4$$

Preuve. Pour tout i, $1 \leq i \leq n$, soit N_i^* un groupe de taille i de diamètre minimum. Soient M_i^* un groupe de taille i d'excentricité minimum et $r_i^* \in M_i^*$ sa racine associée. Soit $M_1^{opt}, M_2^{opt}, \ldots, M_n^{opt}$ une séquence incrémentale d'excentricité minimum. On a :

$$\begin{aligned}
\max_{2\leq i\leq n}\left\{\frac{D_G(M_i)}{D_G(N_i^*)}\right\} &\leq 2\max_{2\leq i\leq n}\left\{\frac{E(M_i,r)}{D_G(N_i^*)}\right\} && \text{(avec } M_1=\{r\} \text{ et parce que } D_G(M_i)\leq 2E(M_i,r)\text{)}\\
&\leq 2\max_{2\leq i\leq n}\left\{\frac{E(M_i,r)}{E(M_i^*,r_i^*)}\right\} && \text{(d'après la définition 17, } E(M_i^*,r_i^*)\leq E(N_i^*,c_i^*)\leq D_G(N_i^*)\text{, avec } c_i^*\in N_i^*\text{)}\\
&= 2\max_{2\leq i\leq n}\left\{\frac{E(M_i^{opt},r^{opt})}{E(M_i^*,r_i^*)}\right\} && \text{(d'après le lemme 12, avec } M_1^{opt}=\{r^{opt}\}\text{)}\\
&\leq 2\max_{2\leq i\leq n}\left\{\frac{E(N_i^{opt},c^{opt})}{E(M_i^*,r_i^*)}\right\} && \text{(car } M_1^{opt},\ldots,M_n^{opt} \text{ est une séquence incrémentale d'excentricité minimum, avec } N_1^{opt}=\{c^{opt}\}\text{)}\\
&\leq 2\max_{2\leq i\leq n}\left\{\frac{D_G(N_i^{opt})}{E(M_i^*,r_i^*)}\right\} && \text{(comme } c^{opt}\in N_i^{opt}\text{, on a } E(N_i^{opt},c^{opt})\leq D_G(N_i^{opt})\text{)}\\
&\leq 4\max_{2\leq i\leq n}\left\{\frac{D_G(N_i^{opt})}{D_G(N_i^*)}\right\} && \text{(car } D_G(N_i^*)\leq D_G(M_i^*)\leq 2E(M_i^*,r_i^*)\text{)}
\end{aligned}$$

□

Le théorème suivant montre que la borne supérieure 4 est atteinte, c'est-à-dire que l'analyse de l'algorithme SEM (voir théorème 22) ne peut pas être améliorée.

Théorème 23 *Pour tout ϵ, $0 < \epsilon < 1$, il existe un graphe valué tel que la séquence incrémentale $M_1, \ldots, M_n$ retournée par l'algorithme* SEM *est telle que :*

$$\frac{\text{coût}(M_1, \ldots, M_n)}{\text{coût}(N_1^{opt}, \ldots, N_n^{opt})} = \frac{4}{1+\epsilon}$$

Preuve. Soit $G_0(\epsilon)$ le graphe valué de la figure 3.3. Pour tout i, $1 \leq i \leq 7$, soit M_i^* un groupe de taille i d'excentricité minimum, soit $r_i^* \in M_i^*$ sa racine associée, et soit N_i^* un groupe de taille i de diamètre minimum. Étant donné $G_0(\epsilon)$, l'algorithme SEM construit une séquence incrémentale à partir de chaque sommet, puis choisit la séquence incrémentale d'excentricité minimum parmi celles-ci. La séquence incrémentale retournée par l'algorithme SEM est la séquence $M_1(a), \ldots, M_7(a)$ obtenue en ajoutant les sommets de $G_0(\epsilon)$ dans l'ordre a, b, c, d, e, f, g. En effet, cette séquence induit $\max_{2\leq i\leq 7}\left\{\frac{E(M_i(a),a)}{E(M_i^*,r_i^*)}\right\} = \frac{2}{1+\epsilon}$, ce qui est la valeur minimum possible pour une séquence incrémentale de $G_0(\epsilon)$. La séquence incrémentale optimale pour le diamètre $N_1^{opt}, \ldots, N_7^{opt}$, est

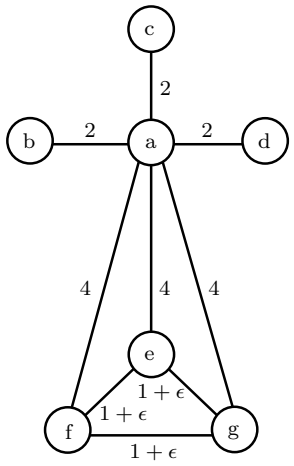


FIG. 3.3 – Le graphe $G_0(\epsilon)$

obtenue en ajoutant les sommets dans l'ordre e, f, g, a, b, c, d. La séquence $M_1(a), \ldots, M_7(a)$ induit $\max_{2\leq i\leq 7}\left\{\frac{D_G(M_i(a))}{D_G(N_i^*)}\right\} = \frac{4}{1+\epsilon}$, alors que $\max_{2\leq i\leq 7}\left\{\frac{D_G(N_i^{opt})}{D_G(N_i^*)}\right\} = 1$. On a donc :

$$\frac{\max_{2\leq i\leq 7}\left\{\frac{D_G(M_i(a))}{D_G(N_i^*)}\right\}}{\max_{2\leq i\leq 7}\left\{\frac{D_G(N_i^{opt})}{D_G(N_i^*)}\right\}} = \frac{4}{1+\epsilon}$$

□

Bilan du Chapitre 3. Dans ce chapitre, nous avons relâché les contraintes *on-line* et *arbre*, dans le but de mesurer l'impact de l'unique contrainte restante : l'*incrémentalité* imposée à la séquence des groupes successifs. Nous avons ensuite évalué la qualité de nos résultats en termes de diamètre, car c'est avec ce paramètre que nous avons obtenu les résultats les plus convaincants. Néanmoins, nous soulignons que le problème étudié ici est plus général, et qu'il se pose de la même façon si le critère de qualité est autre que le diamètre (par exemple s'il s'agit de la distance moyenne entre sommets, critère étudié dans le chapitre 3 avec les contraintes *on-line* et *arbre*).

Nous avons montré que même si nous nous donnons le choix de l'ordre dans lequel ajouter les sommets, il existe des situations dans lesquelles le *coût* de l'incrémentalité d'une séquence (tel que nous l'avons défini, pour le diamètre) est égal à la racine carrée du diamètre du graphe considéré (voir les théorèmes 19 et 20). Nous avons ensuite prouvé que construire une séquence incrémentale optimale ne pouvait pas être approché avec un rapport d'approximation inférieur à 2, sauf si $P = NP$ (voir le théorème 21). Puis, nous avons donné un algorithme 4-approché pour résoudre ce problème (voir le théorème 22). S'il reste toujours un écart entre la borne inférieure et la borne supérieure, nous soulignons qu'il ne s'agit pas d'une faiblesse l'analyse de notre algorithme (d'après le théorème 23, la borne est atteinte). Réduire cet écart nécessitera donc un nouvel algorithme ou une borne inférieure d'inapproximabilité plus forte.

Enfin, nous terminons ce chapitre par la remarque suivante. Bien que nous ayons ici étudié le problème de la construction d'une séquence *incrémentale* optimale, les résultats obtenus sont également valables pour le problème symétrique consistant à trouver une séquence *décrémentale* optimale. En effet, nous évaluons sans la contrainte *on-line* le *coût* d'une séquence par le ratio maximum entre le diamètre de chaque groupe successif et celui de chaque groupe de diamètre minimum du même nombre de sommets. La valeur de ce *coût* (et donc tous les résultats de ce chapitre) ne dépend(ent) donc pas du sens dans lequel nous déroulons la séquence.

Synthèse et perspectives de la première partie

Ce court chapitre fait la synthèse des résultats obtenus dans la première partie de la thèse et propose un certain nombre de perspectives.

Synthèse

Dans le chapitre 1 (resp. 2), nous nous sommes intéressés au diamètre (resp. la somme des distances entre les membres) du groupe. D'abord en termes de rapport de compétitivité dans le modèle *sans reconstruction*, puis en termes de contrainte *qualité pour le diamètre* (resp. *la somme des distances*) dans le modèle *avec reconstructions*. Dans les deux cas, le but a été de minimiser le rapport entre le diamètre (resp. la somme des distances entre les membres) du groupe dans l'arbre et celui (resp. celle) dans le meilleur arbre possible, permettant ainsi d'obtenir une garantie dans le pire cas sur la latence maximum (resp. moyenne) dans le groupe (voir [57, 60] pour une synthèse de ces résultats).

Dans le modèle *sans reconstruction*. Pour chaque sous-problème traité (séquences d'ajouts et de retraits mêlés, d'ajouts seuls, de retraits seuls), notre but a été de

- donner une borne supérieure sur le diamètre (resp. la somme des distances entre les membres) du groupe dans l'arbre en proposant un algorithme et l'analyse de son rapport de compétitivité.
- donner une borne inférieure universelle sur le rapport de compétitivité de tout algorithme pour le diamètre (resp. la somme des distances).

Dans le modèle *avec reconstructions*. Nous précisons que nous avons envisagé ce modèle uniquement lorsqu'il a été nécessaire de le faire, c'est-à-dire lorsque nous avons obtenu un rapport de compétitivité non constant dans le modèle *sans reconstruction*. Pour chaque sous-problème traité (séquences d'ajouts et de retraits mêlés, d'ajouts seuls, de retraits seuls), nous avons

- donné une borne supérieure sur le nombre d'étapes critiques et le nombre de changement élémentaire moyen par étape en proposant un algorithme et son analyse en fonction de ces deux critères. Nous rappelons que dans ce modèle, la qualité en termes de diamètre (resp. la somme des distances) est intégrée sous forme de contrainte *qualité pour le diamètre* (resp. *la somme des distances*). Par définition, le rapport c entre le diamètre (resp. le somme des distances entre les membres) du groupe dans l'arbre et celui (resp. celle) dans le meilleur arbre possible est constant.
- donné une borne inférieure universelle sur le nombre d'étapes critiques dans le pire cas induit par tout algorithme.

	Modèle *sans reconstruction* : Bornes sur le rapport de compétitivité pour le diamètre		Modèle *avec reconstructions* : Bornes sur le nombre de changements élémentaires moyen par étape ($\sharp CEM$) et le nombre d'étapes critiques ($\sharp EC$)		
	Supérieure	Inférieure	Supérieure pour $\sharp CEM$	Supérieure pour $\sharp EC$	Inférieure pour $\sharp EC$
Ajouts et Retraits	X	$m_0 - 1$	$O(i)$	$O(i)$	$\Omega(i)$
Ajouts	2	X	4	0	0
Retraits	$m_0 - 1$	$m_0 - 1$	12	$O(\log i)$	$\Omega(\log i)$

TAB. 3.1 – Tableau récapitulatif des résultats obtenus pour le diamètre

	Modèle *sans reconstruction* : Bornes sur le rapport de compétitivité pour la somme des distances		Modèle *avec reconstructions* : Bornes sur le nombre de changements élémentaires moyen par étape ($\sharp CEM$) et le nombre d'étapes critiques ($\sharp EC$)		
	Supérieure	Inférieure	Supérieure pour $\sharp CEM$	Supérieure pour $\sharp EC$	Inférieure pour $\sharp EC$
Ajouts et Retraits	X	$m_0 - 1$	$O(i)$	$O(i)$	$\Omega(i)$
Ajouts	$\max(2i, 12)$	$\Omega(i)$	12	$O(\log i)$	$\Omega(\log i)$
Retraits	$m_0 - 1$	$m_0 - 1$	$O(i)$	$O(i)$	$\Omega(\log i)$

TAB. 3.2 – Tableau récapitulatif des résultats obtenus pour la somme des distances

Le tableau 3.1 (resp. 3.2) récapitule la valeur des bornes supérieures et inférieures sur les différents critères d'évaluations en fonction des quantités ci-dessus, pour le problème de la minimisation du diamètre (resp. de la somme des distances entre les membres) du groupe dans l'arbre. Les cases marquées par un X désignent les cas pour lesquels nous n'avons pas de borne inférieure ou supérieure intéressante. Nous rappelons les notations suivantes :

- i est le nombre de requêtes on-line (correspondant au nombre d'ajouts et de retraits, d'ajouts uniquement ou bien de retraits uniquement selon le cas étudié).
- m_0 est la taille du groupe initial M_0 et m_i la taille du $i^{\text{ème}}$ groupe M_i.

Remarques : dans le tableau 3.1 (resp. 3.2), la borne inférieure $m_0 - 1$ sur le rapport de compétitivité pour le diamètre (resp. la somme des distances) dans le modèle *sans reconstruction* pour les ajouts et retraits mêlés est une conséquence immédiate de la borne inférieure $m_0 - 1$ pour les retraits seuls. En effet, le cas des retraits seuls étant un cas particulier du cas général des ajouts et retraits mêlés, toute borne inférieure pour le premier cas (particulier) est une borne inférieure pour le deuxième cas (général).

Dans le tableau 3.1, les bornes supérieures 4 pour $\sharp CEM$, 0 pour $\sharp EC$ et la borne inférieure 0 pour $\sharp EC$ dans le cas des ajouts seuls sont une conséquence immédiate du résultat suivant. Dans le modèle *sans reconstruction*, l'algorithme AD est $2-$compétitif. AD respecte donc la contrainte *qualité pour le diamètre* avec un niveau $c = 2$ en induisant (par définition du modèle *sans reconstruction*) 0 étape critique et au plus 4 changements élémentaires par étape (d'après une adaptation immédiate du lemme 9 de la section 2.2.2, en remplaçant chaque occurrence de ASR par AD).

Et enfin, dans les tableaux 3.1 et 3.2, les bornes supérieures en $O(i)$ pour $\sharp CEM$ et $\sharp EC$ dans le cas des ajouts et retraits mêlés sont obtenues en considérant la solution triviale qui consiste à casser puis reconstruire totalement l'arbre à chaque étape.

Bilan sur la performance des algorithmes proposés. Lorsque les ajouts et les retraits sont traités séparément, aussi bien pour le diamètre que pour la somme des distances, les résultats que nous obtenons sont assez satisfaisants puisque l'ordre de grandeur des bornes inférieures et supérieures sont les mêmes (sauf pour le nombre d'étapes critiques dans le cas des retraits pour la somme des distances).

Dans le cas des ajouts et retraits mêlés, aussi bien pour le diamètre que pour la somme des distances, même s'il reste des bornes supérieures à donner sur le rapport de compétitivité et même si la borne supérieure pour $\sharp CEM$ reste à améliorer, nous sommes limités par le résultat très négatif montrant qu'un nombre linéaire d'étapes critiques est nécessaire pour le maintien d'une qualité constante pour le diamètre (resp. la somme des distances).

Quelques résultats bicritères. Bien que cette première partie de thèse ne soit pas dédiée à l'étude bicritère du problème considéré (c'est-à-dire avec garanties *simultanées* en termes de diamètre et de somme des distances), il est intéressant de remarquer que certaines de nos bornes supérieures (sur les rapports de compétitivité, le nombre de changements élémentaires et le nombre d'étapes critiques) peuvent être obtenues *simultanément*. Plus précisément, on a :

- Dans le modèle *sans reconstruction*, pour le cas des ajouts seuls, d'après le théorème 2 (voir section 1.2), l'algorithme AD est $2-$compétitif pour le diamètre et d'après le théorème 10 (voir section 2.2.1), l'algorithme AS est $\max(2i, 12)-$compétitif pour la somme des distances. La seule propriété de l'algorithme AD utilisée dans la preuve du théorème 2 est la suivante :

AD construit un arbre de plus courts chemins enraciné en un sommet du groupe courant. Or, par définition, l'arbre courant construit à chaque étape par AS est aussi un arbre de plus courts chemins enraciné en un sommet du groupe courant. La preuve du théorème 2 prouve donc aussi bien la $2-$compétitivité de AD pour le diamètre que la $2-$compétitivité de AS pour le diamètre. On en déduit que l'algorithme AS est *simultanément* $2-$compétitif pour le diamètre et $\max(2i, 12)-$compétitif pour la somme des distances.

– Dans le modèle *avec reconstructions*, d'après les théorèmes 13 et 14 (voir section 2.2.2), l'algorithme ASR induit au plus 12 changements élémentaires moyens par étape et un nombre d'étapes critiques en $O(\log i)$. D'après le théorème 12 (voir section 2.2.2), ASR respecte la contrainte *qualité pour la somme des distances* (avec un niveau constant paramétrable). De plus, par définition, l'arbre courant construit à chaque étape par ASR est un arbre de plus courts chemins enraciné en un sommet du groupe courant. On en déduit (pour les mêmes raisons qui nous ont conduits à déduire la $2-$compétitivité de AS pour le diamètre) que ASR respecte simultanément la contrainte *qualité pour le diamètre* avec un niveau 2 et la contrainte *qualité pour la somme des distances* (avec un niveau constant paramétrable).
– Dans le modèle *sans reconstruction*, pour le cas des retraits seuls, d'après les théorèmes 3 (voir section 1.3.1) et 16 (voir section 2.3.1), l'algorithme RD est simultanément $(m_0-1)-$compétitif pour le diamètre et $(m_0-1)-$compétitif pour la somme des distances.

	Modèle *sans reconstruction*	Modèle *avec reconstructions*
Ajouts	Algorithme AS $c_d = 2$ $c_s = \max(2i, 12)$	Algorithme ASR $\sharp CEM \leq 12$ $\sharp EC \in O(\log i)$
Retraits	Algorithme RD $c_d = m_0 - 1$ $c_s = m_0 - 1$	X

Tab. 3.3 – Tableau récapitulatif bicritère

Les résultats bicritères sont récapitulés dans le tableau 3.2. Chaque case du tableau indique l'algorithme à utiliser suivant le modèle (*avec* ou *sans reconstruction*(s)) et le cas (ajouts ou retraits) considéré, et rappelle ses performances : on note c_d pour le rapport de compétitivité pour le diamètre et c_s pour le rapport de compétitivité pour la somme des distances. La case marquée par un X désigne le cas pour lesquels nous n'avons pas d'algorithme bicritère.

Simplicité des algorithmes proposés. Tous les algorithmes on-line que nous avons proposés dans cette première partie sont simples. En effet, AD, RD, AS sont des algorithmes *sans reconstruction* et reposent principalement sur l'utilisation de l'algorithme de Dikjstra (ainsi qu'une adaptation

de l'algorithme de Prim pour RD). Leur partie dynamique consiste simplement à connecter chaque nouveau membre par l'ajout d'un plus court chemin ou déconnecter un membre par l'élagage des branches mortes. L'algorithme *avec reconstructions* pour l'ajout ASR (resp. le retrait RDR) est également simple. En effet, chaque étape sans reconstruction gère de la même façon que AS (resp. RD) la mise à jour de l'arbre. Quant aux étapes où l'arbre est totalement reconstruit (les étapes critiques), un arbre de plus courts chemins enracinés en un sommet simple à calculer est construit. De plus, nous insistons sur le fait que la gestion des étapes où une reconstruction est nécessaire est particulièrement simple, puisqu'elle a lieu à chaque fois que la taille du groupe courant est multiplié par une constante (resp. divisé par deux) dans le cas de l'ajout pour la somme des distances (resp. du retrait pour le diamètre). Cela signifie que les étapes de reconstructions sont décidées de manière totalement indépendantes de l'arbre courant, de sa qualité, des membres révélés, etc. Un compteur entier, incrémenté à chaque fois qu'un membre est ajouté (resp. retiré), est suffisant pour la détermination de ces étapes. Cette simplicité permettrait par exemple une mise en œuvre répartie simplifiée de ces algorithmes. En effet, la coordination de la reconstruction pourrait avoir lieu de la manière suivante : chaque membre, lorsqu'il est connecté (resp. déconnecté), envoie sa demande à la racine de l'arbre qui incrémente le compteur. Lorsque la valeur du compteur est multipliée par la constante fixée au départ (resp. divisé par deux), la racine diffuse simplement l'information de reconstruction à chaque membre du groupe dans l'arbre.

Néanmoins, si l'argument de simplicité n'est pas la priorité de l'opérateur, et que celui-ci préfère à tout prix minimiser le nombre d'étapes critiques, on peut raffiner les algorithmes RDR et ASR de la manière suivante. Plutôt que de reconstruire l'arbre à des étapes prédéfinies par un compteur, on peut envisager sa reconstruction uniquement si la contrainte de qualité n'est effectivement plus respectée. Bien sûr, une telle stratégie exige un surplus de calculs, puisqu'à chaque étape, la somme des distances entre les membres du groupe (resp. le diamètre du groupe) dans l'arbre et dans le graphe doivent être calculés. Il faut également noter que cela ne permettrait évidemment pas d'améliorer le pire cas, mais permettrait juste de ne pas reconstruire systématiquement l'arbre comme nous le faisons, et donc d'avoir une performance moyenne meilleure en termes de nombre d'étapes critiques.

Coût de l'incrémentalité. Dans le chapitre 3, nous avons relâché au maximum les contraintes de notre problème de départ, afin d'étudier l'impact de la notion d'*incrémentalité*, indépendamment des difficultés liées à la non connaissance du futur (contrainte *on-line*) et à la construction d'une structure de connexion (contrainte *arbre*). Nous avons montré que le problème (off-line) de la construction d'une séquence incrémentale $M_1 \subset \cdots \subset M_n$ dans un graphe G de *coût* $\max_{2\leq i\leq n} \left\{ \frac{D_G(M_i)}{D_G(N_i^*)} \right\}$ minimum (avec $D_G(M_i)$ le diamètre du groupe M_i dans G et $D_G(N_i^*)$ le diamètre du groupe de taille i de diamètre minimum dans G) ne pouvait pas être approché avec un rapport d'approximation plus petit que 2, sauf si $P = NP$. Nous avons également proposé pour ce problème un algorithme 4-approché et donné deux bornes (supérieure et inférieure) égales à la racine carrée du diamètre du graphe G sur la valeur du *coût* d'une séquence incrémentale optimale.

Perspectives

Un certain nombre de questions restent ouvertes. Dans le modèle *avec reconstructions*, pour le problème du retrait pour la somme des distances, nous n'avons pour l'instant qu'une borne inférieure en $\Omega(\log i)$ sur le nombre d'étapes critiques nécessaires. Une perspective de travail immédiate est donc l'amélioration de la borne inférieure et/ou la proposition d'un algorithme et de son analyse

pour l'obtention d'une borne supérieure meilleure que $\Omega(i)$ (borne supérieure triviale).

De même, les seuls résultats que nous avons dans le cas général des ajouts et retraits mêlés sont des bornes inférieures pour tout algorithme on-line. En effet, nous avons montré que tout algorithme on-line est $\Omega(i)$ − compétitif pour le diamètre (resp. la somme des distances) dans le modèle *sans reconstruction* et induit un nombre d'étapes critiques en $\Omega(i)$ dans le modèle *avec reconstructions*. Si ces résultats montrent qu'il n'est pas possible d'obtenir d'algorithme (*avec* ou *sans reconstruction*(s)) dont les performances soient satisfaisantes, il serait néanmoins intéressant de proposer un algorithme $\Omega(i)$ − compétitif pour le diamètre (resp. la somme des distances) dans le modèle *sans reconstruction* (ce qui permettrait d'obtenir une borne supérieure du même ordre de grandeur que la borne inférieure). La seule borne supérieure sur le diamètre (resp. la somme des distances) que nous pouvons garantir provient de l'analyse d'un algorithme naïf (sans reconstruction) qui ajoute un plus court chemin entre chaque nouveau membre et un sommet quelconque r_0 du groupe de départ, et élague l'arbre à chaque fois qu'un membre veut se déconnecter (tout en prenant soin de garder r_0 dans l'arbre à chaque étape, même si celui-ci n'est plus membre du groupe). Cet algorithme simple est simultanément $(2D_{\max})$-compétitif pour le diamètre et $\left(\left(2+\frac{2}{m_i-1}\right)D_{\max}\right)$ − compétitif (où $D_{\max}$ est le diamètre maximum calculé à partir de tous les sommets des groupes successifs). Nous n'avons présenté cet algorithme ni dans le chapitre 2, ni dans le chapitre 3, car les garanties qu'il offre sont faibles, puisque la valeur de $D_{\max}$ peut être arbitrairement grande. Les deux rapports de compétitivité obtenus ne sont constants que dans le cas très restreint d'un groupe évoluant dans une zone du graphe dont le diamètre $D_{\max}$ est borné. Nous donnons néanmoins une description de cet algorithme ainsi que son analyse en annexe (voir annexe 2).

Le critère poids. La minimisation du poids de l'arbre construit à chaque étape de manière on-line est le critère qui a été le plus étudié (voir section) dans la littérature consacrée au problème de la construction d'une structure couvrante pour des groupes dynamiques. Dans [39], Imaze et Waxman proposent pour le modèle *avec reconstructions* un algorithme on-line traitant le cas des ajouts et des retraits mêlés. Cette stratégie a un rapport de compétitivité constant pour le critère poids et induit un *nombre de changements élémentaires moyen par étape* en $0(\sqrt{i})$. En revanche, les auteurs ne donnent pas de garantie sur le nombre d'étapes critiques. La raison pour laquelle nous n'avons pas inclus dans la première partie de cette thèse d'étude sur le poids de l'arbre tient à plusieurs raisons. La première est qu'il s'agit du critère qui a jusqu'ici été le plus étudié (il s'agit en effet de la version on-line du problème classique de l'arbre de Steiner) ; il nous a donc semblé plus approprié de se focaliser sur le problème de la minimisation des distances dans l'arbre. La deuxième raison est que nous avons obtenu un résultat très négatif sur le nombre d'étapes critiques minimum de tout algorithme. En effet, nous avons montré que si on veut garantir à chaque étape une qualité constante pour le poids de l'arbre construit (même si on se limite à des ajouts on-line), tout algorithme induit un nombre d'étapes critiques en $\Omega(i)$. La méthode triviale consistant à casser l'arbre pour reconstruire à chaque étape de manière off-line une approximation constante de l'arbre de Steiner optimal est donc optimale en ordre de grandeur pour le nombre d'étapes critiques. Nous n'avons pas inclus dans la thèse la preuve de cette borne inférieure car il s'agit en fait d'une généralisation du contre-exemple proposé dans [39] permettant de montrer la borne inférieure en $\Omega(\log i)$ sur le rapport de compétitivité de tout algorithme on-line *sans reconstruction* pour le problème de l'ajout de membres. L'idée de notre généralisation consiste à montrer que le piège proposé par Imaze et Waxman peut arriver un nombre linéaire de fois en nombre de sommets ajoutés, et donc aboutir à un nombre d'étapes critiques en $\Omega(i)$.

Néanmoins, malgré ces résultats très négatifs, il serait intéressant de voir s'il est possible d'obtenir (dans le cas d'ajouts on-line pour commencer) des garanties simultanées sur les critères diamètre,

somme des distances et poids, tout en minimisant le nombre d'étapes critiques (et de reconstructions élémentaires). Bien sûr, les bornes inférieures obtenues nous montrent que si nous voulons une qualité simultanément constante pour les trois critères à chaque étape, $\Omega(i)$ étapes critiques seront nécessaires (nous soulignons que ce résultat est atteignable en utilisant la méthode off-line tricritère proposée dans [46]). En revanche, nous pouvons espérer obtenir une méthode en $\Omega(\log i)$ étapes critiques avec une qualité simultanément constante pour le diamètre et la somme des distances, et logarithmique en i pour le poids.

Relâchement des contraintes. Dans le chapitre 3, nous avons étudié le relâchement des contraintes *on-line* et *arbre*, aboutissant à un problème incrémental off-line, où le but est de trouver la meilleure séquence possible (en fonction du diamètre des groupes successifs). Il faut souligner que l'interprétation que nous avons faite du relâchement des contraintes *on-line* et *arbre* n'est pas la seule envisageable par rapport à notre problème de départ. En effet, il serait par exemple intéressant de commencer par l'étude du relâchement de la contrainte *arbre* uniquement. Ainsi, la structure à construire (toujours de manière on-line) à chaque étape pourrait être maintenant une structure connectant chaque membre du groupe courant et dont le poids serait maîtrisé (cette dernière restriction est nécessaire pour que le problème garde de l'intérêt, sinon il suffit de prendre comme structure le graphe tout entier). L'abandon de la structure d'arbre pourrait être motivé par une exigence concernant la tolérance aux pannes (par exemple en exigeant une structure 2-connexe, c'est-à-dire telle que pour tout couple de membres u et v différents, il existe deux chemins arêtes-disjointes reliant u et v). De plus, le relâchement de la contrainte *arbre* permettrait d'obtenir de meilleures performances pour le diamètre et la somme des distances. En effet, toutes nos bornes inférieures utilisent le fait que la structure construite par tout algorithme doit être un arbre. Si cette contrainte est relâchée, aucune de nos bornes inférieures (sauf celle brièvement décrite dans cette section et égale à $\Omega(i)$ étapes critiques pour le critère poids) ne reste valable. La connaissance du nombre d'étapes critiques nécessaires garantissant une qualité constante pour le diamètre (resp. la somme des distances) dans ce cas reste donc ouvert (qu'il s'agisse du problème de l'ajout, du retrait ou des deux mêlés).

Le relâchement de la contrainte *on-line* peut également s'interpréter différemment de la façon dont nous l'avons faite au chapitre 3, où nous devions *choisir* la séquence des sommets à ajouter plutôt que de la subir sans connaissance du futur. Il serait également intéressant d'étudier le cas intermédiaire où la séquence d'ajouts est *connue à l'avance*. Il ne s'agit alors plus d'un problème on-line, mais d'un problème incrémental off-line où les données (connues dès le départ) doivent être intégrées à la solution au fur et à mesure, dans un ordre déterminé dès le départ. Une application possible de ce modèle pourrait être la suivante : une réunion sur réseau où la liste des participants ainsi que leur ordre d'arrivée est connue dès le début de la réunion. Notons que nous avons déjà obtenu des bornes inférieures dans ce modèle. Comme nous l'avons souligné en remarque à la fin de la section 2.3.2, le théorème 15 montre que même lorsque la contrainte *on-line* est relâchée, tout algorithme respectant les contraintes *arbre* et *qualité pour la somme des distances* induit un nombre d'étapes critiques en $\Omega(\log i)$. Nous avons également montré dans le modèle *sans reconstruction* que tout algorithme respectant la contrainte *arbre* a un rapport de compétitivité en $\Omega(\log i)$ pour la somme des distances (la preuve de ce résultat utilise des arguments très similaires à celle du théorème 15 et se trouve dans [58]). À nouveau, nous insistons sur le fait que si ce résultat semble contre-intuitif, il vient du fait que le rapport de compétitivité doit par définition être vérifié *à chaque étape*, et non uniquement à la dernière étape comme dans le cas d'un problème off-line classique. Une perspective de travail intéressante serait d'utiliser cette connaissance du futur pour obtenir dans le modèle *sans reconstruction* un algorithme plus performant pour la somme des distances que l'algorithme AS (dont le rapport de compétitivité est en $O(i)$).

Seconde partie

Ordonnancements on-line

Présentation de la seconde partie

Définitions et notations

Dans cette seconde partie, nous proposons des algorithmes d'ordonnancement on-line pour résoudre le problème suivant. On se place maintenant au niveau du lien dans un réseau de communications, que l'on considère composé de k sous-canaux de capacité identique (par exemple une fibre optique dans laquelle k fréquences indépendantes peuvent être utilisées simultanément). Ce lien est géré par un opérateur qui le loue à des utilisateurs. Lorsqu'un utilisateur veut réserver le lien (pour transférer des données) pendant une certaine durée entre deux dates, l'opérateur doit choisir sur quel sous-canal l'ordonnancer. Lorsque trop de requêtes ont lieu simultanément, l'opérateur doit choisir lesquelles satisfaire, lesquelles rejeter. Dans la version du problème que nous étudions, nous proposons de prendre en compte le fait que l'opérateur n'a pas toujours la possibilité de connaître à l'avance les requêtes à ordonnancer. À nouveau, il s'agit donc d'un problème *on-line*, que nous modélisons de la manière suivante.

Modélisation du problème. On considère un système de $k \geq 1$ machines parallèles identiques (représentant les k sous-canaux du lien). Une *tâche* Γ est définie par un triplet de réels $\Gamma = (l, r, p)$, où l est son *bord gauche* (l pour *left border*), r son *bord droit* (r pour *right border*) et $p \leq r - l$ sa longueur, correspondant au temps d'exécution de la tâche (p pour *processing time*). Une tâche Γ est *ordonnancée* sur la machine numéro j ($1 \leq j \leq k$) du système si elle occupe un intervalle numérique continu $[l_0, r_0[$ de longueur $p = r_0 - l_0$ entre les bornes l et r sur la machine j, c'est-à-dire si on a $[l_0, r_0[\subseteq [l, r[$ sur la machine j. On dit alors que la tâche Γ est ordonnancée sur la machine j comme l'intervalle $[l_0, r_0[$ *associé à* Γ. Nous soulignons que nous notons tous nos intervalles ouverts à droite car dans notre modèle, deux intervalles $[a, b[$ et $[b, c[$ ne s'intersectent pas. Le but d'un algorithme A résolvant le problème d'ordonnancement on-line que nous venons de décrire est le suivant : à chaque nouvelle tâche Γ_i révélée, l'algorithme A doit construire un *ordonnancement valide* de $\{\Gamma_1, \ldots, \Gamma_i\}$, que nous définissons de la manière suivante.

Remarque : nous soulignons que pour être totalement rigoureux, nous devrions ajouter à la définition d'une tâche un *numéro de client*. En effet, considérons l'exemple problématique suivant. Soient $\Gamma^1 = (l^1, r^1, p^1)$ la première tâche révélée et $\Gamma^2 = (l^2, r^2, p^2)$ la seconde, telles que $l^1 = l^2$, $r^1 = r^2$ et $p^1 = p^2$ (notre modèle autorise une telle situation). Cela signifie que $\Gamma^1 = \Gamma^2$. Pourtant, comme Γ^1 modélise la requête du client n°1 et Γ^2 celle du client n°2, nous considérons ici que $\Gamma^1 \neq \Gamma^2$. Nous avons choisi de ne pas surcharger les notations (en gardant $\Gamma = (l, r, p)$ au lieu de $\Gamma = (l, r, p, i)$) car dans toute cette seconde partie, à aucun moment la confusion entre deux tâches différentes définies par le même triplet ne sera possible.

Définition 20 (Ordonnancement) *Soit* $\{\Gamma_1, \ldots, \Gamma_i\}$ *un ensemble de tâches quelconques et soit* $\{\Gamma'_1, \ldots, \Gamma'_l\}$ *le sous-ensemble de* $\{\Gamma_1, \ldots, \Gamma_i\}$ *des tâches effectivement ordonnancées. Soit* $\{\sigma_1, \ldots, \sigma_l\}$ *l'ensemble des intervalles tels que pour tout* l, $1 \leq l \leq i$, Γ'_l *est ordonnancé comme l'intervalle* σ_l. *On appelle* ordonnancement *de* $\{\Gamma_1, \ldots, \Gamma_i\}$ *sur* k *machines le résultat de l'affectation de chaque intervalle* σ_l *sur une des* k *machines du système.*

Un ordonnancement de $\{\Gamma_1, \ldots, \Gamma_i\}$ *est dit* valide *si chaque tâche de* $\{\Gamma'_1, \ldots, \Gamma'_l\}$ *est ordonnancée au plus une fois et si pour tout couple d'intervalles* σ_a, σ_b *associés aux tâches* Γ'_a, Γ'_b *et ordonnancés sur la même machine, on a* $\sigma_a \cap \sigma_b = \emptyset$.

Par la suite, on dira indifféremment qu'une tâche Γ *ou son intervalle* σ *associé appartient à l'ordonnancement* S *(* $\Gamma \in S$ *ou* $\sigma \in S$*) si* Γ *est ordonnancé dans* S *comme l'intervalle* σ.

Modèle on-line adopté. Les tâches à ordonnancer sont révélées une par une, dans un ordre quelconque, sans aucune connaissance du futur. Lorsqu'une tâche $\Gamma = (l, r, p)$ est révélée, toutes les informations contenues dans le triplet (l, r, p) sont révélées. Nous soulignons que nous ne faisons aucune hypothèse sur l'ordre de révélation des tâches. Cela signifie (entre autres) que nous n'imposons pas (comme c'est souvent le cas) que les tâches soient révélées par ordre croissant de leur bord gauche. Le modèle que nous proposons est général et permet donc de traiter tout type de séquence on-line de tâche. Pour la principale application que nous envisageons, c'est-à-dire le problème de réservation on-line de sous-canaux par des utilisateurs, ceux-ci seront généralement révélés par bord gauche croissant, le bord gauche l correspondant également à l'instant de révélation du triplet $\Gamma = (l, r, p)$. Néanmoins, les résultats que nous présentons (sauf dans le cas bicritère) étant valides également pour des séquences on-line quelconques, nous avons choisi de présenter le problème de la manière la plus générale possible. Nous ajoutons que cette généralisation peut également avoir dans certains cas un intérêt pratique non négligeable. En effet, on peut tout à fait imaginer qu'un client demande longtemps à l'avance, à une date t, la réservation d'un sous-lien pour une durée p entre les dates l et r, avec $t \ll l$, et exige de l'opérateur de savoir immédiatement si sa requête est rejetée ou ordonnancée, et, le cas échéant, pendant quelle plage horaire. Dans ce cas, il n'y a plus correspondance entre l'axe du temps des requêtes successives et celui des début de fenêtres horaires demandées (les bords gauches). Pour résoudre ce problème, nous ne pouvons pas faire l'hypothèse usuelle qui consiste à n'envisager que les séquences on-line de tâches révélées par bords gauches croissants. Notre modèle traitant toutes les séquences on-line de tâches est donc justifiée également d'un point de vue applicatif.

Lorsqu'une nouvelle tâche est révélée, un algorithme on-line peut la *rejeter* ou l'*ordonnancer* sur une des k machines. Dans ce cas, tous les intervalles déjà ordonnancés sur cette machine et intersectant la nouvelle tâche sont *interrompus* (afin d'obtenir à chaque étape un ordonnancement valide). Les tâches rejetées ou interrompues sont *définitivement* perdues.

Nous soulignons que nous autorisons l'interruption d'une tâche déjà ordonnancée car le cas contraire a déjà été étudié et s'est révélé beaucoup trop restrictif (voir section 3.4). L'adoption d'un tel modèle on-line (permettant l'interruption, donc la remise en cause d'une partie de la solution construite) est à mettre en parallèle avec le modèle *avec reconstructions* décrit dans la première partie de la thèse. En effet, dans les deux cas, il s'agit de considérer un modèle on-line souple, autorisant la remise en cause de la solution déjà construite, dans le but d'obtenir une solution (un arbre couvrant ou bien un ordonnancement) de meilleure qualité.

Les critères d'évaluation. Nous avons choisi d'évaluer les algorithmes que nous proposons en fonction des deux paramètres suivants : la *taille* et le *poids* de l'ordonnancement construit.

Définition 21 (La *taille* d'un ordonnancement) *Soit S un ordonnancement valide de $\{\Gamma_1, \ldots, \Gamma_i\}$. On note $|S|$ la* taille *de S, c'est-à-dire le nombre de tâches ordonnancées dans S.*

Si nous considérons que l'opérateur qui gère le lien composé de k sous-canaux identiques (les k machines de notre système) loue ces ressources à des clients utilisateurs, la *taille* correspond au nombre d'utilisateurs dont la requête est satisfaite. Lorsque notre but est la maximisation du nombre d'utilisateurs satisfaits, l'algorithme que nous proposons doit donc maximiser la *taille* de l'ordonnancement. Nous allons évaluer un algorithme en termes de *taille* en définissant le *rapport de compétitivité* d'un algorithme pour la *taille*. Nous rappelons que le rapport de compétitivité est une notion standard de l'algorithmique on-line et consiste à comparer la qualité de la solution construite avec la qualité de la meilleure solution off-line possible (c'est-à-dire la meilleure solution que l'on aurait pu construire si toutes les données du problème nous avaient été révélées en une fois, et non au fur et à mesure).

Définition 22 (Rapport de compétitivité pour la *taille*) *Soit $\Gamma_1, \ldots, \Gamma_i, \ldots$ une séquence on-line quelconque de tâches. Soit* A *un algorithme on-line retournant à chaque étape i un ordonnancement valide S_i de $\{\Gamma_1, \ldots, \Gamma_i\}$ et soit S_i^* un ordonnancement valide de $\{\Gamma_1, \ldots, \Gamma_i\}$ optimal pour la* taille *(c'est-à-dire un ordonnancement de* taille *maximum). L'algorithme* A *a un rapport de compétitivité c (ou est $c-$compétitif) pour la* taille *si on a :*

$$\forall i, \quad c \cdot |S_i| \geq |S_i^*|$$

L'interprétation du rapport de compétitivité pour la *taille* est le suivant. Un algorithme A est $c-$compétitif si à chaque étape i, il construit un ordonnancement valide de *taille* au moins $\frac{1}{c}$ fois la *taille* du meilleur ordonnancement possible.

Définition 23 (Le *poids* d'un ordonnancement) *Soit S un ordonnancement valide de $\{\Gamma_1, \ldots, \Gamma_i\}$. Pour toute tâche $\Gamma = (l, r, p)$ ordonnancée comme l'intervalle $\sigma = [l_0, r_0[$, on note*

$$w(\Gamma) = w(\sigma) = p = r_0 - l_0 \quad \textit{le} \text{ poids } \textit{de } \Gamma \textit{ (resp. } \sigma\textit{).}$$

On définit le poids *d'un ordonnancement S par :*

$$w(S) = \sum_{\Gamma \in S} w(\Gamma)$$

Dans notre exemple d'application, si on considère que l'opérateur loue aux utilisateurs une ressource (machine) à un prix proportionnel au temps d'utilisation, le *poids* d'un ordonnancement correspond au profit total de l'opérateur. Lorsque notre but est la maximisation du profit de l'opérateur, l'algorithme que nous proposons doit maximiser le *poids* de l'ordonnancement. De la même façon que pour la *taille*, nous allons évaluer un algorithme en fonction du critère *poids* en définissant le *rapport de compétitivité* d'un algorithme pour le *poids*.

Définition 24 (Rapport de compétitivité pour le *poids*) *Soit $\Gamma_1, \ldots, \Gamma_i, \ldots$ une séquence on-line quelconque de tâches. Soit* A *un algorithme on-line retournant à chaque étape i un ordonnancement valide S_i de $\{\Gamma_1, \ldots, \Gamma_i\}$ et soit S_i^* un ordonnancement valide de $\{\Gamma_1, \ldots, \Gamma_i\}$ optimal pour le* poids *(c'est-à-dire un ordonnancement de* poids *maximum). L'algorithme* A *a un rapport de compétitivité c (ou est $c-$compétitif) pour le* poids *si on a :*

$$\forall i, \quad c \cdot w(S_i) \geq w(S_i^*)$$

État de l'art

Le premier critère d'optimisation pour les ordonnancements à avoir été étudié de manière on-line est le *makespan*, c'est-à-dire la date de complétion de la dernière tâche ordonnancée. Le résultat le plus fameux dans ce domaine est du à Graham (voir [34, 35]). Il s'agit d'un algorithme $\left(2 - \frac{1}{k}\right) -$ compétitif pour l'ordonnancement sur k machines parallèles identiques d'un ensemble de tâches révélées au fur et à mesure, chaque tâche étant définie par sa longueur. De nombreuses variantes de ce problème ont été par la suite étudiées (voir [4, 5, 6, 22, 24, 40, 52, 70]).

Le problème que nous étudions est connu sous le nom de TCSP (pour *Time-Constrained Scheduling Problem*) (voir [21]). Il existe plusieurs variantes de TCSP, notamment à propos de la fonction qui associe à chaque tâche un profit. Les deux versions du problème que nous étudions ici sont les suivantes : la fonction de profit qui associe à chaque tâche un profit unitaire (correspondant à ce que nous avons appelé le critère *taille*) et la fonction de profit qui associe à chaque tâche un profit proportionnel à sa longueur (correspondant à ce que nous avons appelé le critère *poids*). Il existe une troisième fonction de profit souvent étudiée, associant à chaque tâche un profit arbitraire. Dans [16], ces trois variantes de TCSP sont étudiées dans un contexte off-line. Comme chacune des variantes du problème est NP-complète, Bar-Noy et al. ont proposé des algorithmes avec rapport d'approximation constants. Dans [11], la version bicritère de TCSP est étudié et un algorithme pour la maximisation simultanée de deux critères correspondant à deux fonctions de profits différentes est proposé (les rapports d'approximation pour chacun des critères sont constants). Bien sûr, les algorithmes proposés dans [11, 16] étant off-line, nous ne pouvons pas les utiliser pour résoudre la version on-line du problème que nous avons présentée.

Dans le contexte on-line, des cas particuliers de TCSP ont déjà été étudiés. Le cas des intervalles (c'est-à-dire des tâches telles que $p = r - l$, donc avec une seule alternative d'ordonnancement sur chaque machine du système) a été plus particulièrement étudié. Deux approches ont été envisagées. La première n'autorise pas l'interruption des intervalles déjà ordonnancés (un nouvel intervalle ne peut donc pas interrompre un intervalle déjà ordonnancé). Ce modèle (correspondant au cas où la remise en cause de la solution courante est interdite) est à mettre en parallèle avec le modèle *sans reconstruction* que nous avons décrit dans la première partie de la thèse. En effet, dans ce modèle, l'arbre couvrant construit au fur et à mesure ne peut pas être remis en cause. Dans [26, 33, 31, 48], plusieurs algorithmes on-line sans interruption sont proposés, mais ces algorithmes n'ont un rapport de compétitivité constant qu'au prix de fortes restrictions sur la longueur des intervalles soumis. En effet, dans [48], R. J. Lipton et A. Tomkins montrent qu'il n'existe pas d'algorithmes on-line dont le rapport de compétitivité est constant lorsque l'interruption n'est pas autorisée. Dans [15, 23, 27, 67], l'interruption est autorisée et des algorithmes on-line avec rapport de compétitivité constant sont proposés. Ce modèle avec interruption (correspondant au cas où la remise en cause, totale ou partielle, de la solution courante est autorisée) est à mettre en parallèle avec le modèle *avec reconstructions* que nous avons décrit dans la première partie de la thèse. Au delà de la différence de nature des objets construits (arbres couvrants dans la première partie, ordonnancements dans la deuxième), la démarche est similaire.

Plus récemment, plusieurs travaux [2, 28, 32, 43, 51] ont étudié des variantes on-line de TCSP (avec interruptions autorisées) pour l'ordonnancement de tâches (c'est-à-dire avec $p \leq r - l$). Des algorithmes avec rapports de compétitivité constants ont été proposés. Néanmoins, ces travaux se placent dans le cas particulier d'un système à une seule machine. Le problème est alors simplifié, puisque la liberté supplémentaire consistant à choisir sur quelle machine ordonnancer une tâche n'a pas à être prise en compte.

Nous soulignons que dans tous les travaux cités ci-dessus et dont le but est la maximisation du

profit pour des tâches de longueur quelconque, le poids de chaque tâche (ou intervalle) est soit égal à 1, soit un fonction de sa longueur p. Cette restriction vient du fait qu'il n'existe pas d'algorithme on-line dont le rapport de compétitivité est constant pour des poids arbitraires, même lorsque l'interruption est autorisée (voir la preuve de ce résultat dans [10]). Cette remarque montre donc l'intérêt de l'étude des critères *taille* et *poids* tels que nous les avons définis dans ce chapitre.

Plan de la seconde partie

Dans la seconde partie de la thèse (les chapitres 4, 5 et 5.3), nous traitons le problème décrit dans le chapitre 3.4, de manière monocritère pour les tâches dans le chapitre 4 et de manière bicritère pour les intervalles dans le chapitre 5.

Chapitre 4. Dans la section 4.1, nous proposons un algorithme on-line pour la maximisation de la *taille* pour l'ordonnancement de tâches sur $k \geq 1$ machines. Nous analysons la qualité de cet algorithme en termes de rapport de compétitivité. Nous montrons que celui-ci est constant dans un certain nombre de cas particuliers, correspondant à des restrictions (raisonnables) sur la longueur des tâches.

Dans la section 4.2, nous proposons un algorithme on-line pour la maximisation du *poids* pour l'ordonnancement de tâches sur $k \geq 1$ machines. Pour analyser plus finement la qualité de cet algorithme, nous introduisons la notion de *poids avec pénalité*, dont le principe est de pénaliser (de manière paramétrable) l'interruption d'une tâche par l'opérateur (pour prendre en compte dans la fonction de profit la gêne ainsi créée envers le client utilisateur dont la requête est interrompue). Nous analysons également la qualité de cet algorithme en termes de rapport de compétitivité. Nous montrons que celui-ci est constant pour le *poids avec pénalité*. Une conséquence directe de ce résultat est que l'algorithme que nous proposons a un rapport de compétitivité constant pour le critère *poids*.

Dans la section 4.3, nous montrons qu'il n'existe d'algorithme on-line de rapport de compétitivité 1 ni pour le problème de la maximisation de la *taille*, ni pour celui de la maximisation du *poids*.

Dans ce chapitre, notre apport par rapport à l'état de l'art est le suivant. Nous proposons le premier algorithme on-line pour l'ordonnancement de tâches sur k machines parallèles identiques pour la maximisation de la *taille* (resp. du *poids*). Les travaux précédents ont déjà traité l'ordonnancement on-line sur k machines pour chacun des deux critères, mais pour le cas particulier des intervalles uniquement, ou bien ont traité l'ordonnancement de tâches pour chacun des deux critères, mais uniquement pour le cas particulier d'une seule machine (voir section 3.4). En plus de la généralisation des intervalles aux tâches et d'une à plusieurs machines, nous proposons de prendre en compte la nuisance créée par l'interruption d'une tâche en instaurant un principe de pénalité lorsqu'une tâche est interrompue.

Chapitre 5. Dans la section 5.2, nous présentons un algorithme on-line bicritère pour la maximisation *simultanée* de la *taille* et du *poids* pour l'ordonnancement d'intervalles sur $k \geq 4$ machines, lorsque ceux-ci sont révélés dans l'ordre croissant de leur bord gauche. Nous montrons que notre algorithme a des rapports de compétitivité *simultanément* constants pour chacun des deux critères.

Dans la section 5.3, nous montrons qu'il n'existe pas d'algorithme dont les rapports de compétitivité sont simultanément strictement inférieurs à 2 pour la *taille* et le *poids*.

Enfin, nous soulignons qu'à notre connaissance, il n'existe pas de travaux antérieurs à ceux que nous présentons dans le chapitre 5 traitant le problème de maximisation du poids et de la taille de manière on-line *et* bicritère.

Chapitre 4

Ordonnancements monocritères de tâches

Ce chapitre est dédié à la maximisation de la *taille* et du *poids avec pénalité* pour l'ordonnancement on-line de tâches. La section 4.1 traite le problème de la maximisation de la *taille* (voir [59]) et la section 4.2 celle du *poids avec pénalité* (voir [61]). Enfin, nous montrons dans la section 4.3 qu'il n'existe d'algorithme on-line optimal (c'est-à-dire de rapport de compétitivité 1) ni pour la *taille*, ni pour le *poids*.

4.1 Un algorithme pour la maximisation de la *taille*

Dans cette section, nous présentons l'algorithme MT (pour Maximisation de la Taille). L'idée de l'algorithme MT est de favoriser la tâche de petite taille en remplaçant une tâche déjà ordonnancée par la nouvelle tâche révélée si celle-ci est de longueur moins de 2 fois plus petite que la tâche interrompue. Plus formellement, nous définissons MT de la manière suivante.

Algorithme Maximisation de la Taille – MT

Soit k le nombre de machines parallèles identiques du système.
Soit S_j $(1 \leq j \leq k)$ le sous-ordonnancement sur la machine j
de l'ordonnancement courant S.
Soit $\Gamma = (l, r, p)$ la nouvelle tâche révélée.
    Si $\exists$ une machine j et un intervalle $\sigma \subseteq [l, r[$ satisfaisant $\forall \sigma' \in S_j,\ \sigma \cap \sigma' = \emptyset$
    Alors ordonnancer Γ comme l'intervalle σ sur la machine j
    Sinon
        Si $\exists$ une machine j et un intervalle $\sigma \subseteq [l, r[$
        satisfaisant $\exists \sigma_0 \in S_j,\ \sigma \subset \sigma_0$ et $2p(\sigma) \leq p(\sigma_0)$
        Alors interrompre σ_0 et ordonnancer Γ comme l'intervalle σ sur la machine j
        Sinon rejeter Γ.

Remarque : par définition, à chaque étape, l'algorithme MT construit un ordonnancement valide. Pour simplifier les notations, nous dirons par la suite qu'un intervalle σ satisfait $\text{cond}_j(\sigma)$ si et seulement si

$$(\forall \sigma' \in S_j,\ \sigma \cap \sigma' = \emptyset) \text{ ou } (\exists \sigma_0 \in S_j,\ \sigma \subset \sigma_0 \text{ et } 2p(\sigma) \leq p(\sigma_0)).$$

En utilisant l'algorithme BI (pour Bords Importants) suivant, trouver un intervalle $\sigma \subseteq [l, r[$ de longueur p satisfaisant $\text{cond}_j(\sigma)$ s'il en existe un (correspondant aux lignes 5 et 8 de l'algorithme MT) peut être fait en temps polynomial pour chaque machine j.

Algorithme Bords Importants – BI

1 Soit $\Gamma = (l, r, p)$ la nouvelle tâche révélée.
2 Soit j $(1 \leq j \leq k)$ le numéro de la machine vérifiée.
3 Sur cette machine j :
4 Soit $\{[l_1, r_1[, \ldots, [l_n, r_n[\}$ l'ensemble des intervalles déjà ordonnancés
5 sur la machine j tels que $l < r_1 \leq l_2 < r_2 \leq \cdots \leq l_n < r_n < r$.
6 Soit $l, l_1, r_1, l_2, r_2, \ldots, l_n, r_n$ la séquence des *bords importants*.
7 Soit d le premier bord important (dans l'ordre croissant) satisfaisant :
8 $[d, d + p[\subseteq [l, r[$ et $\text{cond}_j([d, d + p[)$.
9 Si un tel d existe
10 Alors il existe un intervalle $\sigma = [d, d + p[\subseteq [l, r[$ satisfaisant $\text{cond}_j(\sigma)$
11 Sinon il n'existe pas d'intervalle $\sigma \subseteq [l, r[$ de longueur p satisfaisant $\text{cond}_j(\sigma)$.

Remarque : l'idée principale derrière l'algorithme BI est la suivante. Pour trouver s'il existe ou non une machine j sur laquelle il existe un intervalle pouvant accueillir la nouvelle tâche soumise (soit parce que la machine est libre, soit parce que la condition de remplacement est satisfaite, voir $\text{cond}_j(\sigma)$), il n'est pas nécessaire de tester tous les intervalles possibles (ce qui ne serait de toute façon pas possible puisqu'il en existe un nombre infini indénombrable, notre modèle temporel n'étant pas discrétisé). L'algorithme BI choisit donc un sous-ensemble de taille polynomial parmi tous les intervalles à tester pour voir si un intervalle σ satisfaisant $\text{cond}_j(\sigma)$ existe. Chaque intervalle effectivement testé par BI est alors le représentant d'une zone (bornée) pouvant contenir une infinité d'intervalles équivalents du point de vue de la condition $\text{cond}_j(\sigma)$. Le théorème suivant prouve la correction de l'algorithme BI, c'est-à-dire qu'il montre que ce sous-ensemble de taille polynomial d'intervalles testés est suffisant pour répondre à notre problème.

Théorème 24 *Pour chaque machine j $(1 \leq j \leq k)$, l'algorithme* BI *trouve un intervalle $\sigma = [d, d + p[\subseteq [l, r[$ de longueur p satisfaisant $\text{cond}_j([d, d + p[)$ si et seulement si il existe $\sigma \subseteq [l, r[$ de longueur p satisfaisant $\text{cond}_j(\sigma)$.*

Preuve. Par définition, si l'algorithme BI trouve un intervalle $[d, d + p[\subseteq [l, r[$, celui-ci est de longueur p et satisfait bien $\text{cond}_j([d, d + p[)$.

Nous allons maintenant prouver que s'il existe un intervalle $\sigma \subseteq [l, r[$ de longueur p satisfaisant $\text{cond}_j(\sigma)$, alors l'algorithme BI en trouve un. En effet, supposons, par contradiction, qu'il existe un intervalle $[l_0, r_0[\subseteq [l, r[$ de longueur p (donc tel que $r_0 - l_0 = p$) satisfaisant $\text{cond}_j([l_0, r_0[)$ et que l'algorithme BI n'en trouve aucun. Soit $d_0 \in \{l, l_1, r_1, l_2, r_2, \ldots, l_n, r_n\}$ le plus grand bord important tel que $d_0 \leq l_0$. Par définition de BI, l'intervalle $[d_0, d_0 + p[$ est testé par l'algorithme. Deux cas peuvent se produire :

- Si $[l_0, r_0[$ n'intersecte aucun intervalle ordonnancé sur la machine j. Comme d_0 est le plus grand bord important tel que $d_0 \leq l_0$ et comme $[l_0, r_0[$ et $[d_0, d_0 + p[$ sont de longueur p, $[d_0, d_0 + p[$ n'intersecte aucun intervalle ordonnancé sur la machine j. Cela signifie que

$[d_0, d_0+p[$ satisfait $\text{cond}_j([d_0, d_0+p[)$. Par définition, l'algorithme BI choisit donc cet intervalle ; contradiction.

– Sinon, il existe un intervalle σ ordonnancé sur la machine j, intersectant $[l_0, r_0[$. Comme $[l_0, r_0[$ satisfait $\text{cond}_j([l_0, r_0[)$, on a $[l_0, r_0[\subseteq \sigma$. De plus, comme d_0 est le plus grand bord important tel que $d_0 \leq l_0$ et comme $[l_0, r_0[$ et $[d_0, d_0+p[$ sont de longueur p, on a également $[d_0, d_0+p[\subseteq \sigma$. Cela signifie que $[d_0, d_0+p[$ satisfait $\text{cond}_j([d_0, d_0+p[)$. Par définition, l'algorithme BI choisit donc cet intervalle ; contradiction.

□

Analyse du rapport de compétitivité de l'algorithme MT. Pour donner l'expression du rapport de compétitivité de l'algorithme MT, nous avons besoin de définir les quantités suivantes. On suppose ici qu'une séquence de tâche $\Gamma_1, \ldots, \Gamma_i$ a été révélée à MT, dans cet ordre et de manière on-line.

Définition 25 (Nombre de longueurs de tâches) *Pour toute séquence on-line $\Gamma_1, \ldots, \Gamma_i$, on notera λ le nombre de longueurs de tâche différentes parmi les tâches apparaissant dans la séquence $\Gamma_1, \ldots, \Gamma_i$:*

$$\lambda = |\{p(\Gamma_1), \ldots, p(\Gamma_i)\}|$$

Définition 26 (Rapport maximum entre longueurs de tâches)
Pour toute séquence on-line $\Gamma_1, \ldots, \Gamma_i$, on notera γ le rapport entre la longueur de la plus grande tâche et la longueur de la plus petite tâche dans la séquence $\Gamma_1, \ldots, \Gamma_i$:

$$\gamma = \frac{\max\{p(\Gamma) : \Gamma \in \{\Gamma_1, \ldots, \Gamma_i\}\}}{\min\{p(\Gamma') : \Gamma' \in \{\Gamma_1, \ldots, \Gamma_i\}\}}$$

Le théorème 25 montre que l'algorithme MT est $(4\min(\lambda, \lfloor \log_2(\gamma) \rfloor + 1)) -$ compétitif.

Théorème 25 *L'algorithme MT est $(4\min(\lambda, \lfloor \log_2(\gamma) \rfloor + 1)) -$ compétitif, c'est-à-dire que pour toute séquence on-line $\Gamma_1, \ldots, \Gamma_i$, on a :*

$$4\min(\lambda, \lfloor \log_2(\gamma) \rfloor + 1)\,|S| \geq |S^*|$$

Remarque : nous soulignons le fait que dans de nombreuses situations (correspondant aux séquences de tâches "homogènes"), ce rapport de compétitivité est constant. Par exemple, lorsque $\lambda = c$ (avec c une constante), l'algorithme MT est $4c -$ compétitif et lorsque $\gamma = 2^{c'}$ (avec c' une constante), l'algorithme MT est $(4c' + 4) -$ compétitif.

Pour prouver le théorème 25, nous avons besoin des définitions suivantes, utilisant les notations précédentes, avec S l'ordonnancement produit par MT lorsque cette séquence a été traitée et S^* un ordonnancement de taille maximum de $\{\Gamma_1, \ldots, \Gamma_i\}$.

Définition 27 (Sous-ordonnancement S_j (resp. S_j^*)) *Pour tout j ($1 \leq j \leq k$), on note S_j (resp. S_j^*) le sous-ordonnancement de S (resp. S^*) sur la machine j.*

Définition 28 (Ensemble d'intervalles T_j) *Pour chaque machine j ($1 \leq j \leq k$), soit T_j l'ensemble des intervalles associés aux tâches ordonnancées par l'algorithme MT sur la machine j, y compris les intervalles provenant des tâches ayant été ordonnancées puis interrompues ultérieurement.*

Remarque : nous soulignons que deux intervalles appartenant à T_j peuvent s'intersecter, puisqu'en général, T_j n'est pas un ordonnancement valide.

Définition 29 (Sous-ordonnancement S_j^{*A} (resp. S_j^{*B})) *Soit S_j^{*A} le sous-ordonnancement de S_j^* ($S_j^{*A} \subseteq S_j^*$) contenant les tâches ordonnancées dans S^* sur la machine j et qui ont été acceptées et ordonnancées par* MP *(sur n'importe quelle machine, et qui ont potentiellement été interrompues ultérieurement). Soit S_j^{*B} le sous-ordonnancement de S_j^* ($S_j^{*B} \subseteq S_j^*$) contenant les tâches ordonnancées dans S^* sur la machine j et qui ont été rejetées par* MP *au moment où elles ont été révélées. Notons que S_j^{*A} et S_j^{*B} forment clairement une partition de S_j^*.*

Définition 30 (La fonction f) *Soit Γ une tâche de la séquence $\{\Gamma_1, \ldots, \Gamma_i\}$, ordonnancée dans la solution optimale sur la machine j comme l'intervalle x et rejetée par* MT *(c'est-à-dire que l'on a $x \in S_j^{*B}$). Soit S_j le sous-ordonnancement sur la machine j à l'étape où Γ est révélée. On définit l'ensemble Y_x d'intervalles par :*

$$Y_x = \{y \in S_j : (p(y) < 2p(x) \text{ et } x \subseteq y) \text{ ou } (x \cap y \neq \emptyset \text{ et } x \nsubseteq y)\}.$$

On définit alors la fonction f de la manière suivante :

$$\begin{array}{rccl} f : & S_j^{*B} & \rightarrow & T_j \\ & x & \mapsto & y = [l_y, r_y[\text{ tel que } y \in Y_x \\ & & & \text{et } l_y = \min\{l_{y'} : y' \in Y_x\} \end{array}$$

Remarques : l'interprétation intuitive de la fonction f est la suivante : $f(x)$ est un intervalle (associé à une tâche) ordonnancé sur la machine j qui a empêché la tâche Γ d'être ordonnancée sur la machine j. Nous donnons maintenant les lemmes techniques suivants, préliminaires à la preuve du théorème 25.

Lemme 13 *Pour toute machine j ($1 \leq j \leq k$), pour tout intervalle $x \in S_j^{*B}$, on a :*

$$|\{y : y = f(x)\}| = 1$$

Preuve. Soit Γ une tâche de la séquence $\{\Gamma_1, \ldots, \Gamma_i\}$, ordonnancée dans S_j^{*B} comme l'intervalle x. Soit S_j le sous-ordonnancement courant sur la machine j lorsque Γ est révélée. Puisque $x \in S_j^{*B}$, Γ a été rejetée par l'algorithme MT. En particulier, cela signifie que l'algorithme MT a rejeté Γ de la machine j. Par définition de MT et d'après le théorème 24, il existe donc $y_0 \in S_j$ tel que $(p(y_0) < 2p(x)$ *et* $x \subseteq y_0)$ *ou* $(x \cap y_0 \neq \emptyset$ *et* $x \nsubseteq y_0)$. S'il existe plusieurs y_0, on choisit (arbitrairement) celui dont la valeur du bord gauche est minimum. Par définition de la fonction f, c'est cet unique intervalle qui est choisi et on a ainsi $|\{y : y = f(x)\}| = 1$. □

Lemme 14 *Soit Γ une tâche de la séquence $\{\Gamma_1, \ldots, \Gamma_i\}$, ordonnancée dans S_j^{*B} comme l'intervalle x. Pour tout intervalle $y \in T_j$, on a :*

$$|\{x : y = f(x)\}| \leq 3$$

Preuve. Par contradiction, on suppose qu'il existe $y \in T_j$ tel que $|\{x : y = f(x)\}| \geq 4$. On note $\{x_1 = [l_1, r_1[, x_2 = [l_2, r_2[, \ldots, x_n = [l_n, r_n[\}$ (avec $n \geq 4$) l'ensemble d'intervalles tels que

$\bigcup_{1\leq l\leq n}\{x_l\} = \{x : y = f(x)\}$, triés par ordre croissant de leur bord gauche (c'est-à-dire que l'on a $l_1 < l_2 < \cdots < l_n$). Comme $y = f(x)$, on a :

$$x \cap y \neq \emptyset \tag{4.1}$$

De plus, comme $x \in S_j^{*B} \subseteq S_j^*$, pour tout a, b $(1 \leq a \leq b \leq n)$, on a $x_a \cap x_b = \emptyset$ (car S_j^* est un ordonnancement valide sur une machine). On a donc :

$$r_1 \leq l_2 < r_2 \leq l_3 < \cdots < r_{n-1} \leq l_n \tag{4.2}$$

D'après (4.1) et (4.2), on obtient :

$$\forall l,\ 2 \leq l \leq n-1,\ x_l \subset y \tag{4.3}$$

Sans perte de généralité, on suppose que $p(x_2) = \min\left\{p(x) : x \in \bigcup_{2\leq l\leq n-1}\{x_l\}\right\}$. On a donc :

$$\begin{array}{lrcll} & (n-2)p(x_2) & \leq & \sum_{l=2}^{n-1} p(x_l) & \\ \Rightarrow & 2p(x_2) & \leq & \sum_{l=2}^{n-1} p(x_l) & (\text{car } n \geq 4) \\ \Rightarrow & 2p(x_2) & \leq & p(y) & (\text{d'après (4.3) et car les } x_l \text{ sont deux à deux disjoints}) \end{array}$$

Ce résultat contredit le fait que $y = f(x_2)$ (en effet, par définition de la fonction f, on a $p(y) < 2p(x_2)$). □

Lemme 15 *On a :*

$$|S^*| \leq 4\sum_{j=1}^{k} |T_j|$$

Preuve. D'après le lemme 13, pour toute machine j $(1 \leq j \leq k)$, on a : $\{x : y \in T_j \text{ et } y = f(x)\} = S_j^{*B}$. On a donc :

$$|\{x : y \in T_j \text{ et } y = f(x)\}| \ = \ |S_j^{*B}| \tag{4.4}$$

D'après le lemme 14, on a :

$$|\{x : y \in T_j \text{ et } y = f(x)\}| \ \leq \ 3|T_j| \tag{4.5}$$

D'après (4.4) et (4.5), on obtient :

$$|S_j^{*B}| \ \leq \ 3|T_j| \tag{4.6}$$

Deux cas peuvent se produire :

- Si $|S^*| \leq 4\sum_{j=1}^{k} |S_j^{*A}|$. Par définition de S_j^{*A}, $\forall \sigma_a \in S_j^{*A}$, $\exists \sigma_b \in \bigcup_{1\leq j\leq k} T_j$ tel que σ_a et σ_b sont associés à la même tâche Γ. on a donc : $\sum_{j=1}^{k} |S_j^{*A}| \leq \sum_{j=1}^{k} |T_j|$ On obtient :

$$|S^*| \leq 4\sum_{j=1}^{k} |T_j|$$

- Si $|S^*| \geq 4\sum_{j=1}^{k} |S_j^{*A}|$. Par définition de S_j^{*A} et S_j^{*B}, on a $S_j^{*A} \cap S_j^{*B} = \emptyset$ et $S_j^{*A} \cup S_j^{*B} = S_j^*$.

On obtient donc :

$$\begin{array}{rrcll} & |S^*| & = & \sum_{j=1}^{k} |S_j^{*A}| + \sum_{j=1}^{k} |S_j^{*B}| & \\ \Rightarrow & |S^*| & \leq & \frac{|S^*|}{4} + \sum_{j=1}^{k} |S_j^{*B}| & \\ \Rightarrow & \frac{3}{4}|S^*| & \leq & \sum_{j=1}^{k} |S_j^{*B}| & \\ \Rightarrow & |S^*| & \leq & \frac{4}{3} \sum_{j=1}^{k} |S_j^{*B}| & \\ \Rightarrow & |S^*| & \leq & 4 \sum_{j=1}^{k} |T_j| & \text{(d'après (4.6))} \end{array}$$

□

Lemme 16 *Pour toute machine j ($1 \leq j \leq k$), on a :*

$$\min\left(\lambda, \lfloor \log_2(\gamma) \rfloor + 1\right) \cdot |S_j| \geq |T_j|$$

Preuve. Pour tout intervalle $\sigma \in S_j$, on définit la séquence des intervalles *enracinée* en σ par $R(\sigma) = x_1, x_2, \ldots, x_i$ où $\sigma = x_i$ et i est le plus grand entier tel que pour tout l ($1 \leq l \leq i-1$), x_l a été remplacé par x_{l+1} au cours de l'exécution de l'algorithme MT. Par définition de MT, on a :

$$\begin{array}{rll} & p(x_1) \geq 2p(x_2) \geq 2^2 p(x_3) \geq \cdots \geq 2^{i-1} p(x_i) & \\ \Rightarrow & \dfrac{p(x_1)}{p(x_i)} \geq 2^{i-1} & \\ \Rightarrow & \gamma \geq 2^{i-1} & (\text{car } \gamma = \frac{\max\{p(\Gamma):\Gamma \in \{\Gamma_1,\ldots,\Gamma_i\}\}}{\min\{p(\Gamma'):\Gamma' \in \{\Gamma_1,\ldots,\Gamma_i\}\}} \text{ et car } x_1 \text{ et } x_i \\ & & \text{sont des intervalles associés à des tâches de } \{\Gamma_1, \ldots, \Gamma_i\}) \\ \Rightarrow & \lfloor \log_2(\gamma) \rfloor \geq |R(\sigma)| - 1 & (\text{car } |R(\sigma)| = i \text{ est un entier}) \end{array}$$

On obtient donc :

$$\lfloor \log_2(\gamma) \rfloor + 1 \geq |R(\sigma)| \tag{4.7}$$

Par définition de λ, on a :

$$\begin{array}{rcll} \lambda & = & \left| \bigcup_{\Gamma \in \{\Gamma_1,\ldots,\Gamma_i\}} \{p(\Gamma)\} \right| & \\ & \geq & \left| \bigcup_{\sigma' \in R(\sigma)} \{p(\sigma')\} \right| & \\ & = & |R(\sigma)| & \begin{array}{l}(\text{car, par définition de MT,} \\ p(x_1) > p(x_2) > \cdots > p(x_i))\end{array} \end{array}$$

On obtient alors :

$$\lambda \geq |R(\sigma)| \tag{4.8}$$

Par définition de S_j, T_j et $R(\sigma)$, on a :

$$\begin{aligned}
& \bigcup_{\sigma \in S_j} R(\sigma) \;=\; T_j \\
\Rightarrow \quad & \left| \bigcup_{\sigma \in S_j} R(\sigma) \right| \;=\; |T_j| \\
\Rightarrow \quad & \sum_{\sigma \in S_j} |R(\sigma)| \;\geq\; |T_j| \\
\Rightarrow \quad & \sum_{\sigma \in S_j} \min\left(\lambda, \lfloor \log_2(\gamma) \rfloor + 1\right) \;\geq\; |T_j| \qquad \text{(d'après (4.7) et (4.8))} \\
\Rightarrow \quad & \min\left(\lambda, \lfloor \log_2(\gamma) \rfloor + 1\right) \cdot |S_j| \;\geq\; |T_j|
\end{aligned}$$

□

Nous pouvons maintenant donner une preuve du théorème 25.

Preuve du théorème 25. D'après les lemmes 15 et 16, on a :

$$|S^*| \;\leq\; 4 \sum_{j=1}^{k} |T_j| \;\leq\; 4 \sum_{j=1}^{k} \min\left(\lambda, \lfloor \log_2(\gamma) \rfloor + 1\right) \cdot |S_j| \;=\; 4 \min\left(\lambda, \lfloor \log_2(\gamma) \rfloor + 1\right) \cdot |S|$$

□

4.2 Un algorithme pour la maximisation du *poids avec pénalité*

Dans cette section nous proposons un algorithme on-line pour la maximisation du *poids avec pénalité* pour l'ordonnancement de tâches sur $k \geq 1$ machines. Afin de prendre en compte la nuisance créée par l'éventuelle interruption de tâches déjà ordonnancées (comme le permet notre modèle), nous définissons une nouvelle fonction de poids, que nous appelons *poids avec pénalité*.

Définition 31 (Le *poids avec pénalité*) *Soit $\beta \geq 0$ une constante de pénalité fixée. Soit Γ la nouvelle tâche révélée exécutée comme l'intervalle σ, de poids $w(\Gamma) = w(\sigma)$. On définit le* poids avec pénalité *de Γ par*

$$\mathsf{w_p}(\Gamma) = \mathsf{w_p}(\sigma) = w(\sigma) - \beta \cdot \sum_{\substack{\{\sigma' \,:\, \sigma' \text{ est inter-} \\ \text{-rompue par } \sigma\}}} w(\sigma')$$

On définit alors le poids avec pénalité *d'un ordonnancement S par*

$$\mathsf{w_p}(S) = \sum_{\sigma \in S} \mathsf{w_p}(\sigma)$$

Remarques : nous soulignons que le *poids avec pénalité* n'a de sens que pour un ordonnancement on-line, celui-ci n'étant pas défini pour un ordonnancement off-line, où la notion d'interruption n'a pas de sens. Nous soulignons également que lorsque l'on pose $\beta = 0$, le *poids* et le *poids avec pénalité* d'une tâche (resp. un ordonnancement) sont identiques. La notion de *poids avec pénalité* que nous proposons est donc une généralisation de la notion de *poids*. Enfin, nous insistons sur le fait que nous n'avons pas défini de pénalité pour le critère *taille* car la quantité à maximiser ne correspond pas à un profit de l'opérateur. Pénaliser l'opérateur en "pénalisant la *taille* de l'ordonnancement" n'aurait donc pas de sens.

Motivation du *poids avec pénalité*. Le premier intérêt de la définition d'une telle fonction de *poids* vient de l'application du problème à la réservation de ressources par des utilisateurs d'un sous-canal à l'opérateur qui gère le lien. Dans cette application, on considère que le profit de l'opérateur est égal au *poids* de l'ordonnancement construit. Nous savons (voir section 3.4) qu'interdire à l'opérateur toute interruption est trop restrictif, puisque le rapport de compétitivité pour le *poids* peut alors être arbitrairement grand : il suffit pour s'en convaincre de considérer l'exemple d'une nouvelle tâche révélée dont le poids est un nombre de fois arbitrairement plus grand que l'ordonnancement déjà construit et ne pouvant être placée que si au moins une tâche est interrompue ; interdire l'interruption empêche donc tout algorithme on-line d'être compétitif pour le critère *poids*. Cette première observation montre qu'il est nécessaire d'autoriser l'interruption, afin d'éviter que l'opérateur ait un profit arbitrairement petit. Mais autoriser à l'opérateur d'interrompre autant qu'il le souhaite l'exécution de tâches déjà en cours présente l'inconvénient d'être peu satisfaisant du point de vue du client utilisateur. En effet, celui-ci peut voir sa tâche s'interrompre alors qu'il comptait sur sa complétion. Le *poids avec pénalité* permet donc d'instaurer un principe de dédommagement des clients dont la tâche Γ est interrompue : non seulement cette tâche n'est pas facturée (une tâche, lorsqu'elle est interrompue n'est plus comptabilisée dans le poids de l'ordonnancement, donc dans le profit de l'opérateur), mais en plus, le retrait de la quantité $\beta \cdot w(\Gamma)$ au *poids* de l'ordonnancement (donc au profit de l'opérateur) peut être versée par l'opérateur à l'utilisateur en dédommagement de la nuisance créée.

D'une manière plus théorique, le *poids avec pénalité* permet de faire le lien entre les deux modèles on-line classiques *avec* et *sans* interruption. En effet, lorsque la constante de pénalité β est égale à 0, nous sommes dans le cas de l'étude du *poids* où l'interruption est autorisée sans aucune restriction. À l'inverse, lorsque β tend vers l'infini, un algorithme n'aura aucun intérêt à remplacer une tâche, et nous nous retrouvons dans un modèle équivalent au modèle interdisant l'interruption. Le paramètre β permet donc d'atteindre tous les intermédiaires possibles entre ces deux modèles extrêmes, puisqu'il permet de sanctionner de manière quantifiable l'interruption d'une tâche.

Ce modèle avec pénalité est à mettre en relation avec la minimisation du *nombre d'étapes critiques* défini dans le cadre du modèle *avec reconstructions* (voir chapitre 1). Dans les deux cas, le but est de prendre en compte des perturbations dues à la remise en cause de la solution courante (modifications internes de l'arbre dans la première partie et interruption de tâches dans l'ordonnancement dans la seconde). Dans la première partie de la thèse, l'approche consiste à minimiser le nombre d'étapes où ces perturbations ont lieu, dans la seconde, un principe de dédommagement pour chaque perturbation est proposé.

Notons également que l'idée consistant à pénaliser les interruptions a déjà été envisagée dans un certain nombre de travaux, mais dans des buts et contextes très différents des nôtres. Par exemple, dans [49, 50], les auteurs traitent le problème de la minimisation du makespan sur une machine, où chaque tâche a une date de disponibilité avant laquelle elle ne peut pas être ordonnancée. L'interruption d'une tâche (avec reprise ultérieure de son exécution) est autorisée, mais elle est sanctionnée par une pénalité consistant à rendre inactive la machine pendant un certain temps, juste après l'interruption. Les auteurs justifient cette pénalité par le coût en temps du stockage mémoire des informations nécessaires à la reprise ultérieure de l'exécution de la tâche interrompue. En revanche, pour le problème de la maximisation du poids, il n'existe pas à notre connaissance de travaux proposant un principe de pénalités semblable au nôtre.

Afin d'évaluer la qualité d'un algorithme, nous devons maintenant définir le rapport de compétitivité pour le *poids avec pénalité* de la manière suivante.

Définition 32 (Rapport de compétitivité pour le *poids avec pénalité*) *Soit* $\Gamma_1, \ldots, \Gamma_i, \ldots$ *une séquence on-line quelconque de tâches. Soit* A *un algorithme on-line retournant à chaque étape* i *un ordonnancement valide* S_i *de* $\{\Gamma_1, \ldots, \Gamma_i\}$ *et soit* S_i^* *un ordonnancement valide de* $\{\Gamma_1, \ldots, \Gamma_i\}$ *optimal pour le* poids *(c'est-à-dire un ordonnancement de* poids *maximum). L'algorithme* A *a un rapport de compétitivité* c *(ou est* $c-$ *compétitif) pour le* poids avec pénalité *si on a :*

$$c \cdot \mathsf{w_p}(S_i) \geq w(S_i^*)$$

L'algorithme MP**.** Soit $\alpha > 1$ une constante fixée par l'opérateur au début de l'exécution de l'algorithme. Dans la suite de cette section, nous choisirons α de sorte que le rapport de compétitivité de l'algorithme soit le plus petit possible. L'idée de l'algorithme MP est de favoriser les tâches les plus longue (donc dont le poids est le plus grand) en remplaçant une (ou plusieurs) tâche(s) déjà ordonnancée(s) par la nouvelle tâche révélée si celle-ci a un poids au moins α fois plus grand que la somme des tâches interrompues. Plus formellement, nous définissons MP (pour Maximisation du Poids) de la manière suivante.

Algorithme Maximisation du Poids – MP

1 Soit k le nombre de machines parallèles identiques du système.
2 Soit S_j $(1 \leq j \leq k)$ le sous-ordonnancement sur la machine j
3 de l'ordonnancement courant S.
4 Soit $\Gamma = (l, r, p)$ la nouvelle tâche révélée.
5 S'il existe une machine j telle que
6 il existe un intervalle $\sigma \subseteq [l, r[$ de longueur p satisfaisant :
7 $w(\sigma) \geq \alpha \sum_{\sigma' \in S_j \text{ et } \sigma' \cap \sigma \neq \emptyset} w(\sigma')$
8 Alors interrompre (si nécessaire) les intervalles σ' et ordonnancer Γ
9 comme l'intervalle σ sur la machine j
10 Sinon rejeter Γ.

Remarques : par définition, à chaque étape, l'algorithme MP construit un ordonnancement valide. Pour simplifier les notations, nous dirons par la suite qu'un intervalle σ satisfait $\text{cond2}_j(\sigma)$ si et seulement si

$$w(\sigma) \;\geq\; \alpha \sum_{\substack{\sigma' \in S_j \text{ et} \\ \sigma' \cap \sigma \neq \emptyset}} w(\sigma').$$

On appelle BI_2 l'adaptation de l'algorithme BI de la section 4.1 qui consiste à remplacer dans BI chaque occurrence de cond_j par cond2_j. En utilisant BI_2, trouver un intervalle $\sigma \subseteq [l, r[$ de longueur p satisfaisant $\text{cond2}_j(\sigma)$ s'il en existe un (correspondant aux lignes 6 et 7 de l'algorithme MP) peut être fait en temps polynomial pour chaque machine j.

Le théorème suivant prouve la correction de l'algorithme BI_2.

Théorème 26 *Pour chaque machine j ($1 \leq j \leq k$), l'algorithme* BI_2 *trouve un intervalle $\sigma = [d, d+p[\subseteq [l, r[$ de longueur p satisfaisant $cond2_j([d, d+p[)$ si et seulement si il existe $\sigma \subseteq [l, r[$ de longueur p satisfaisant $cond2_j(\sigma)$.*

Preuve. Par définition, si l'algorithme BI trouve un intervalle $\sigma = [d, d+p[\subseteq [l, r[$, celui-ci est de longueur p et satisfait bien $\text{cond2}_j([d, d+p[)$.

Nous allons maintenant prouver que s'il existe un intervalle $\sigma \subseteq [l, r[$ de longueur p satisfaisant $\text{cond2}_j(\sigma)$, alors l'algorithme BI_2 en trouve un. En effet, supposons, par contradiction, qu'il existe un intervalle $[l_0, r_0[\subseteq [l, r[$ de longueur p (donc tel que $r_0 - l_0 = p$) satisfaisant $\text{cond2}_j([l_0, r_0[)$ et que l'algorithme BI_2 n'en trouve aucun. Soit $d_0 \in \{l, l_1, r_1, l_2, r_2, \ldots, l_n, r_n\}$ le plus grand bord important tel que $d_0 \leq l_0$. Par définition de BI, l'intervalle $[d_0, d_0 + p[$ est testé par l'algorithme. Deux cas peuvent se produire :

- Si $d_0 + p \leq l_0$. Si $[d_0, d_0 + p[$ n'intersecte aucun intervalle déjà ordonnancé, l'algorithme BI_2 choisit cet intervalle, puisque $\text{cond2}_j([d_0, d_0 + p[)$ est satisfaite. Comme cela contredit le fait que l'algorithme BI ne trouve aucun intervalle, cela signifie qu'il en existe un $\sigma' \in S_j$ tel que σ' intersecte $[d_0, d_0 + p[$. Cet intervalle σ' intersecte nécessairement $[l_0, r_0[$, puisque d_0 est le plus grand bord important tel que $d_0 \leq l_0$. On a donc :
$$\sum_{\substack{\sigma \in S_j \text{ et} \\ \sigma \cap [l_0, r_0[\neq \emptyset}} w(\sigma) \;\geq\; w(\sigma') \;=\; \sum_{\substack{\sigma \in S_j \text{ et} \\ \sigma \cap [d_0, d_0 + p[\neq \emptyset}} w(\sigma)$$
- Nous montrons maintenant que le second cas possible mène à la même inégalité. En effet, si $d_0 + p > l_0$, comme $[l_0, r_0[$ et $[d_0, d_0 + p[$ sont de longueur p et comme $d_0 \leq l_0$, on a $d_0 + p \leq r_0$. On obtient :
$$\sum_{\substack{\sigma \in S_j \text{ et} \\ \sigma \cap [l_0, r_0[\neq \emptyset}} w(\sigma) \quad \geq \quad \sum_{\substack{\sigma \in S_j \text{ et} \\ \sigma \cap [l_0, d_0 + p[\neq \emptyset}} w(\sigma)$$
$$\Rightarrow \quad \sum_{\substack{\sigma \in S_j \text{ et} \\ \sigma \cap [l_0, r_0[\neq \emptyset}} w(\sigma) \quad \geq \quad \sum_{\substack{\sigma \in S_j \text{ et} \\ \sigma \cap [d_0, d_0 + p[\neq \emptyset}} w(\sigma)$$
(comme d_0 est le plus grand bord important tel que $d_0 \leq l_0$, pour tout $\sigma \in S_j$, si $\sigma \cap [l_0, d_0 + p[\neq \emptyset$, alors on a $\sigma \cap [d_0, d_0 + p[\neq \emptyset$)

  Étant donné que les deux cas mènent à la même inégalité, on obtient :
$$w([l_0, r_0[) \quad \geq \quad \alpha \sum_{\substack{\sigma \in S_j \text{ et} \\ \sigma \cap [l_0, r_0[\neq \emptyset}} w(\sigma) \quad \geq \quad \alpha \sum_{\substack{\sigma \in S_j \text{ et} \\ \sigma \cap [d_0, d_0 + p[\neq \emptyset}} w(\sigma)$$
(car $[l_0, r_0[$ satisfait $\text{cond2}_j([l_0, r_0[)$)
$$\Rightarrow \quad w([d_0, d_0 + p[) \quad \geq \quad \alpha \sum_{\substack{\sigma \in S_j \text{ et} \\ \sigma \cap [d_0, d_0 + p[\neq \emptyset}} w(\sigma)$$
(car $w([d_0, d_0 + p[) = p = w([l_0, r_0[)$)

  Comme l'intervalle $[d_0, d_0 + p[\subseteq [l_i, r_i[$ est de longueur p et satisfait $\text{cond2}_j([d_0, d_0 + p[)$, l'algorithme BI_2 le choisit ; contradiction.

□

Analyse du rapport de compétitivité de l'algorithme MP. Le théorème 27 montre que l'algorithme MP est $\left((2\alpha+3)\left(1+\frac{\beta}{\alpha-\beta}+\frac{1}{\alpha-\beta-1}\right)\right)$ – compétitif pour le *poids avec pénalité* (où β est la constante de pénalité).

Théorème 27 *Soit $\beta \geq 0$ une constante (représentant la pénalité associée à l'interruption d'une tâche). Soit $\alpha > \beta + 1$ une constante fixée par l'opérateur au début de l'exécution de l'algorithme* MP. *L'algorithme* MP *est $\left((2\alpha+3)\left(1+\frac{\beta}{\alpha-\beta}+\frac{1}{\alpha-\beta-1}\right)\right)$ – compétitif pour le* poids avec pénalité. *C'est-à-dire que pour toute séquence on-line $\Gamma_1, \ldots, \Gamma_i$, on a :*

$$\left((2\alpha+3)\left(1+\frac{\beta}{\alpha-\beta}+\frac{1}{\alpha-\beta-1}\right)\right)\mathsf{w_p}(S) \geq w(S^*)$$

Remarques : nous rappelons que la constante de pénalité β, fixée au début de l'exécution de l'algorithme, représente le coût pour l'opérateur associé à l'interruption d'une tâche déjà ordonnancée (proportionnel à la somme des poids des tâches interrompues). Par exemple :

- Si $\beta = 0$. Cela signifie qu'aucune pénalité n'est payée lors de l'interruption d'une tâche déjà ordonnancée. Le rapport de compétitivité obtenu est alors $(2\alpha+3)\left(1+\frac{1}{\alpha-1}\right)$. La valeur minimum de la fonction $\alpha \mapsto (2\alpha+3)\left(1+\frac{1}{\alpha-1}\right)$ est ≈ 13.32 et est atteinte pour $\alpha \approx 2.58$.

- Si $\beta = 0.2$. Cela signifie que l'opérateur paie une pénalité égale à 20 % de la somme des poids des tâches interrompues. Le rapport de compétitivité obtenu est alors $(2\alpha+3)\left(1+\frac{0.2}{\alpha-0.2}+\frac{1}{\alpha-1.2}\right)$. La valeur minimum de la fonction $\alpha \mapsto (2\alpha+3)\left(1+\frac{0.2}{\alpha-0.2}+\frac{1}{\alpha-1.2}\right)$ est ≈ 14.63 et est atteinte pour $\alpha \approx 2.88$.

- Si $\beta = 1$. Cela signifie que l'opérateur paie une pénalité égale à la somme des poids des tâches interrompues. Le rapport de compétitivité obtenu est alors $(2\alpha+3)\left(1+\frac{1}{\alpha-1}+\frac{1}{\alpha-2}\right)$. La valeur minimum de la fonction $\alpha \mapsto (2\alpha+3)\left(1+\frac{1}{\alpha-1}+\frac{1}{\alpha-2}\right)$ est ≈ 20.14 et est atteinte pour $\alpha \approx 4.16$.

- Si $\beta = 10$. Cela signifie que l'opérateur paie une pénalité égale à 10 fois la somme des poids des tâches interrompues. Le rapport de compétitivité obtenu est alors $(2\alpha+3)\left(1+\frac{10}{\alpha-10}+\frac{1}{\alpha-11}\right)$. La valeur minimum de la fonction $\alpha \mapsto (2\alpha+3)\left(1+\frac{10}{\alpha-10}+\frac{1}{\alpha-11}\right)$ est ≈ 90.38 et est atteinte pour $\alpha \approx 21.4$.

Le choix de la valeur de β doit être fait par l'opérateur au début de l'exécution de l'algorithme en fonction de deux aspects : la garantie que celui-ci veut avoir en termes de profit par rapport à une solution optimale (la valeur du rapport de compétitivité) et l'attractivité pour les clients potentiels en termes de dédommagement offert. En effet, plus β est proche de 0, plus le rapport de compétitivité est petit, et donc plus la garantie en termes de profit est importante. En revanche, un client est peu dédommagé lorsque sa requête est interrompue (donc un service peu attractif pour le client). À l'inverse, si β est grand (par exemple $\beta = 10$), l'opérateur a une garantie beaucoup moins importante (un rapport de compétitivité d'environ 90.38), mais un dédommagement très attractif pour le client, puisqu'une tâche interrompue implique (en plus du remboursement intégral du profit généré par la tâche) un dédommagement égal à 10 fois ce que le client aurait payé pour

la complétion de sa requête. La valeur de β peut donc être vue comme un indicateur d'attractivité pour le client. En conclusion, β aura intérêt à être important lorsque l'opérateur se trouve dans un environnement fortement concurrentiel (les clients ont le choix entre de nombreux opérateurs) et pourra être réduit dans un environnement faiblement concurrentiel.

Le tableau 4.1 récapitule le rapport de compétitivité de l'algorithme MP en fonction de la constante de pénalité β. La première ligne correspond aux différentes valeurs de β et la deuxième aux valeurs minimum de la fonction $\alpha \mapsto \left((2\alpha+3)\left(1+\frac{\beta}{\alpha-\beta}+\frac{1}{\alpha-\beta-1}\right)\right)$ correspondant (arrondies au centième) pour $\alpha > \beta+1$ (c'est-à-dire le rapport de compétitivité de MP).

β	0	0.2	1	2	3	4	5	10	100	1000
c	13.32	14.63	20.14	27.49	35.12	42.88	55.77	90.38	810.04	8010.00

TAB. 4.1 – Le rapport de compétitivité c de MP en fonction de β

Pour prouver le théorème 27, nous avons besoin des définitions suivantes, dont certaines sont des adaptations pour l'algorithme MP de définitions données en section 4.1. Afin de faciliter la lecture de cette section, nous rappelons que pour tout j ($1 \leq j \leq k$), on note S_j (resp. S_j^*) le sous-ordonnancement de S (resp. S^*) de la machine j (voir définition 27 en remplaçant MT par MP).

Définition 33 (Poids d'un ensemble d'intervalles) *Soit $I = \{\sigma_1, \ldots, \sigma_n\}$ un ensemble d'intervalles :*

$$w(I) = \sum_{\sigma \in I} w(\sigma)$$

Remarque : nous soulignons que deux intervalles de I peuvent s'intersecter, puisque I n'est pas nécessairement un ordonnancement valide.

Définition 34 (Poids d'un intervalle discontinu) *Le poids d'un intervalle discontinu $\sigma = [a_1, b_1[\cup [a_2, b_2[\cup \cdots \cup [a_p, b_p[$ (avec $a_1 < b_1 < a_2 < b_2 < \cdots < a_p < b_p$) est égal à :*

$$w(\sigma) = \sum_{n=1}^{p} (b_n - a_n)$$

Remarque : dans la suite de cette section, nous utilisons souvent l'*union* mathématique classique. Nous insistons ici particulièrement sur la différence entre $\bigcup_{\sigma \in I}\{\sigma\}$ et $\bigcup_{\sigma \in I}\sigma$. En effet, $\bigcup_{\sigma \in I}\{\sigma\}$ est un ensemble d'intervalles, alors que $\bigcup_{\sigma \in I}\sigma$ est un intervalle (éventuellement discontinu). Par exemple, si on considère les intervalles $\sigma_1 = [0,2[$, $\sigma_2 = [1,4[$, $\sigma_3 = [5,10[$ et l'ensemble d'intervalles $I = \{\sigma_1, \sigma_2, \sigma_3\}$, on a :

$$\bigcup_{\sigma \in I}\{\sigma\} = \{\ [0,2[, [1,4[, [5,10[\ \} \quad \text{et} \quad \bigcup_{\sigma \in I}\sigma = [0,4[\cup [5,10[$$

Cela signifie (d'après les définitions 33 et 34) que l'on a :

$$w(I) = 2+3+5 = 10 \quad \text{et} \quad w\left(\bigcup_{\sigma \in I}\sigma\right) = 4+5 = 9$$

Définition 35 (L'ensemble I_j, la zone $Z(\sigma)$ et l'ensemble $E(\sigma)$) *Soit $\alpha > 1$ une constante. Soit k le nombre de machines du système. Soit Γ une tâche ordonnancée dans l'ordonnancement optimal sur la machine j ($1 \leq j \leq k$) comme l'intervalle $\sigma = [l, r[\in S_j^*$, et rejetée de la machine j par l'algorithme* MP *(nous rappelons que $w(\Gamma) = w(\sigma) = p = r - l$). On appelle I_j l'ensemble d'intervalles (disjoints) intersectant σ et présents sur la machine j à l'étape à laquelle Γ a été rejetée. L'intervalle $Z(\sigma)$ est la* zone associée *à σ et est définie de la manière suivante :*

1. *S'il existe $\sigma' = [l', r'[\in I_j$ tel que $\sigma' \cap \sigma \neq \emptyset$ et $l' < l - \frac{p}{\alpha}$, alors :*
$$Z(\sigma) = \left[l - \frac{p}{\alpha}, l + p\right[$$
2. *Sinon, s'il existe $\sigma' = [l', r'[\in I_j$ tel que $\sigma' \cap \sigma \neq \emptyset$ et $l - \frac{p}{\alpha} \leq l' < l$, alors :*
$$Z(\sigma) = \left[l', l' + p + \frac{p}{\alpha}\right[$$
3. *Sinon :*
$$Z(\sigma) = \left[l, l + p + \frac{p}{\alpha}\right[$$

*Soit $E(\sigma) = \{\sigma' \cap Z(\sigma) : \sigma' \in I_j\}$. Pour tout $\sigma' \in I_j$, on dit que $\sigma' \cap Z(\sigma) \in E(\sigma)$ est l'*intervalle tronqué associé à σ' *si on a $\sigma' \nsubseteq Z(\sigma)$ (c'est-à-dire si $\sigma' \neq \sigma' \cap Z(\sigma)$).*

Remarque : nous soulignons que pour tout intervalle σ, on a $w(Z(\sigma)) = p + \frac{p}{\alpha} = \left(1 + \frac{1}{\alpha}\right) w(\sigma)$ et que pour tout intervalle $\sigma' \in E(\sigma)$, on a $\sigma' \subseteq Z(\sigma)$.

La figure 4.1 illustre la définition 35.
La définition suivante est identique à la définition 29 de la section 4.1. Néanmoins, dans un souci de lisibilité, nous rappelons son énoncé.

Définition *Soit S_j^{*A} le sous-ordonnancement de S_j^* ($S_j^{*A} \subseteq S_j^*$) contenant les tâches ordonnancées dans S^* sur la machine j et qui ont été acceptées et ordonnancées par* MP *(sur n'importe quelle machine). Soit S_j^{*B} le sous-ordonnancement de S_j^* ($S_j^{*A} \subseteq S_j^*$) contenant les tâches ordonnancées dans S^* sur la machine j et qui ont été rejetées par* MP *au moment où elles ont été révélées. Notons que S_j^{*A} et S_j^{*B} forment clairement une partition de S_j^*.*

Nous rappelons que $\alpha > 1$ est une constante choisie par l'opérateur au début de l'exécution de l'algorithme MP. Pour prouver le théorème 27, nous avons besoin des lemmes préliminaires suivants.

Lemme 17 *Pour tout j ($1 \leq j \leq k$), pour tout $\sigma \in S_j^{*B}$, on a :*
$$\alpha \cdot w(E(\sigma)) \geq w(\sigma)$$

Preuve. Soit Γ la tâche ordonnancée dans l'ordonnancement optimal comme l'intervalle $\sigma = [l, r[\in S_j^{*B}$ sur la machine j. Comme σ appartient à S_j^{*B}, Γ a été rejetée par l'algorithme MP lorsque celle-ci a été soumise. En particulier, Γ n'a pas été ordonnancée sur la machine j. Par définition de MP, il existe donc I_j (où I_j est un ensemble d'intervalles disjoints présents sur la machine j lorsque Γ a été rejetée) tel que la propriété suivante est vérifiée (sinon, d'après le théorème 26, MP aurait ordonnancé la tâche Γ sur la machine j) :

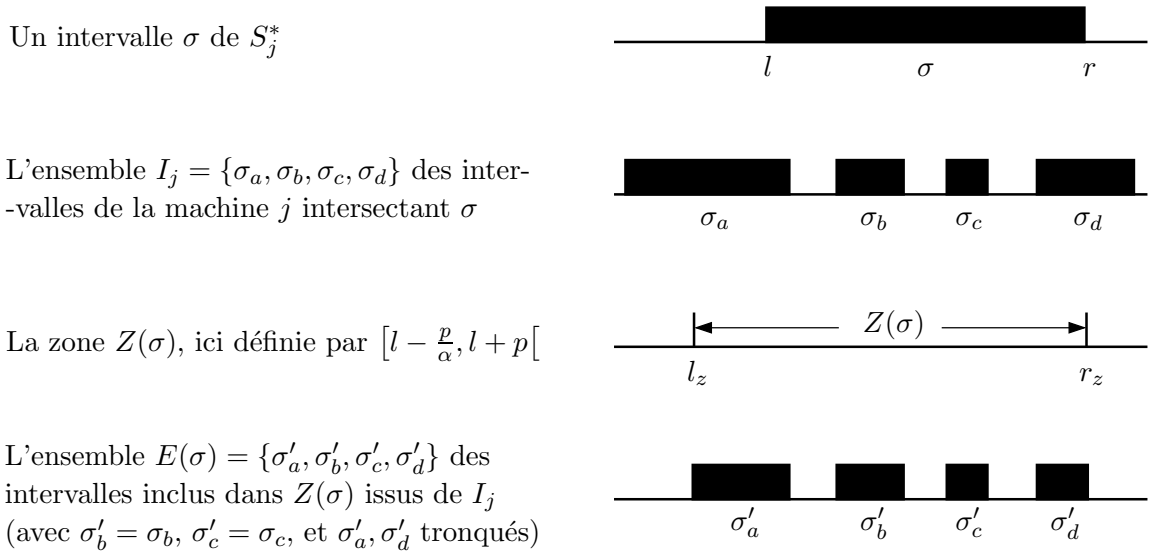


FIG. 4.1 – Illustration de la zone $Z(\sigma)$ et de l'ensemble $E(\sigma)$

– Si $I_j \subseteq E(\sigma)$, par définition de MP, comme Γ a été rejetée de la machine j, on a :

$$\alpha \cdot w(I_j) \geq w(\sigma) \;\Rightarrow \alpha \cdot w(E(\sigma)) \geq w(\sigma) \quad (\text{car } I_j \subseteq E(\sigma))$$

– Si $I_j \nsubseteq E(\sigma)$. Trois cas peuvent se produire :

1. D'après la définition 35, s'il existe $\sigma_b = [l_b, r_b[\in I_j$ tel que $\sigma_b \cap \sigma \neq \emptyset$ et $l_b < l - \frac{p}{\alpha}$, alors $Z(\sigma) = \left[l - \frac{p}{\alpha}, l + p\right[$. On a donc :

$$\begin{array}{rcll} w(\sigma_a) & \geq & \frac{p}{\alpha} & (\text{avec } \sigma_a \text{ un intervalle tronqué de } \sigma_b \text{ et on a } l_a = l - \frac{p}{\alpha} \text{ et } \sigma_a \cap \sigma \neq \emptyset) \\ \Rightarrow\ \alpha \cdot w(E(\sigma)) & \geq & w(\sigma) & (\text{car } \sigma_a \in E(\sigma) \text{ et } p = w(\sigma)) \end{array}$$

2. D'après la définition 35, s'il existe $\sigma_b = [l_b, r_b[\in I_j$ tel que $\sigma_b \cap \sigma \neq \emptyset$ et $l - \frac{p}{\alpha} \leq l_b < l$, alors $Z(\sigma) = \left[l_b, l_b + p + \frac{p}{\alpha}\right[$. Comme $\sigma_b \cap \sigma \neq \emptyset$, on a donc :

$$w(\sigma_b) \geq w([l_b, l[) = l - l_b \tag{4.9}$$

De plus, comme $I_j \nsubseteq E(\sigma)$, il existe un intervalle tronqué. Deux sous-cas peuvent alors se produire :

– S'il existe un intervalle tronqué $\sigma_a = [l_a, r_a[$ venant de $\sigma_b \in I_j$, on a $l_a = l_b$ et $r_a > r$ (car σ_a est tronqué sur la droite, puisqu'il ne peut pas ici être tronqué sur la gauche). De plus, comme $\sigma_a \in E(\sigma)$ et $\alpha > 1$, on a :

$$\alpha \cdot w(E(\sigma)) > w(E(\sigma)) \geq w(\sigma_a) = r_a - l_a > r - l = p = w(\sigma)$$

– Sinon, il existe un intervalle tronqué sur la droite $\sigma_a = [l_a, r_a[$. Ainsi, on a $r_a = l_b + p + \frac{p}{\alpha}$. Comme σ_a est un intervalle tronqué et $\sigma_a \cap \sigma \neq \emptyset$, on a $l_a < r = l + p$ et

donc :

$$w(\sigma_a) \geq w([l+p, l_b+p+\frac{p}{\alpha}[) = l_b+p+\frac{p}{\alpha}-l-p = l_b-l+\frac{p}{\alpha} \qquad (4.10)$$

Comme nous sommes dans le cas où $w(E(\sigma)) \geq w(\sigma_a)+w(\sigma_b)$ (car σ_b n'est pas tronqué), en sommant (4.9) et (4.10), on obtient (nous rappelons que $p = w(\sigma)$) :

$$w(\sigma_a)+w(\sigma_b) \geq \frac{p}{\alpha} \quad \Rightarrow \quad \alpha \cdot w(E(\sigma)) \geq w(\sigma)$$

3. D'après la définition 35, tous les autres cas mènent à $Z(\sigma) = \big[l, l+p+\frac{p}{\alpha}\big[$. Comme il existe un intervalle tronqué $\sigma_a = [l_a, r_a[\in E(\sigma)$, on a :

$$\begin{array}{rrll} & w(\sigma_a) & \geq & \frac{p}{\alpha} \quad \text{(car dans ce cas, } \sigma_a \text{ est tronqué et on a } r_a = l+p+\frac{p}{\alpha} \text{ et } \sigma_a \cap \sigma \neq \emptyset) \\ \Rightarrow & \alpha \cdot w(E(\sigma)) & \geq & w(\sigma) \quad \text{(car } \sigma_a \in E(\sigma) \text{ et } p = w(\sigma)) \end{array}$$

□

Pour prouver le lemme 18, nous avons besoin de la définition suivante.

Définition 36 (Sous-ensemble S_j^{*R} minimal pour l'inclusion) *Pour tout j ($1 \leq j \leq k$), S_j^{*R} est un sous-ensemble de S_j^{*B}* minimal pour l'inclusion *s'il vérifie :*

$$\bigcup_{\sigma \in S_j^{*R}} Z(\sigma) = \bigcup_{\sigma \in S_j^{*B}} Z(\sigma) \qquad et \qquad \forall \sigma' \in S_j^{*R}, \bigcup_{\sigma \in (S_j^{*R} \setminus \{\sigma'\})} Z(\sigma) \neq \bigcup_{\sigma \in S_j^{*B}} Z(\sigma).$$

Lemme 18 *Pour tout j ($1 \leq j \leq k$), on a :*

$$w(S_j^{*B}) \leq \frac{\alpha+1}{\alpha} w(S_j^{*R})$$

Preuve.

$$\begin{array}{rcll} w(S_j^{*B}) & = & w\left(\bigcup_{\sigma \in S_j^{*B}} \sigma\right) & (\text{car } \forall \sigma, \sigma' \in S_j^{*B} \subseteq S_j^* \ (\sigma \neq \sigma'), \text{ on a } \sigma \cap \sigma' = \emptyset) \\ & \leq & w\left(\bigcup_{\sigma \in S_j^{*B}} Z(\sigma)\right) & (\text{car par définition de } Z(\sigma),\ w(\sigma) \leq w(Z(\sigma))) \\ & = & w\left(\bigcup_{\sigma \in S_j^{*R}} Z(\sigma)\right) & (\text{car, par définition de } S_j^{*R}, \text{ on a } \bigcup_{\sigma \in S_j^{*B}} Z(\sigma) = \bigcup_{\sigma \in S_j^{*R}} Z(\sigma)\ ) \\ & \leq & \sum_{\sigma \in S_j^{*R}} w(Z(\sigma)) & \\ & = & \left(1+\frac{1}{\alpha}\right) \sum_{\sigma \in S_j^{*R}} w(\sigma) & (\text{car, par définition de } Z(\sigma), \forall \sigma \in S_j^{*B},\ w(Z(\sigma)) = \left(1+\frac{1}{\alpha}\right) w(\sigma)) \end{array}$$

□

Pour prouver le lemme 19, nous avons besoin de la définition suivante.

Définition 37 (Ensemble d'intervalles triés par bords gauches croissants) *Pour tout j ($1 \leq j \leq k$), $\{[l_1, r_1[, [l_2, r_2[, \ldots, [l_n, r_n[\}$ est l'ensemble des intervalles de $\bigcup_{\sigma \in S_j^{*R}} \{Z(\sigma)\}$ triés par bords gauches croissants (donc tels que $l_1 \leq l_2 \leq \cdots \leq l_n$). On définit $P_j^1 \subseteq S_j^{*R}$ et $P_j^2 \subseteq S_j^{*R}$ les deux ensembles d'intervalles satisfaisant :*

$$\bigcup_{\sigma \in P_j^1} \{Z(\sigma)\} = \{[l_1, r_1[, [l_3, r_3[, [l_5, r_5[, \ldots, [l_{2i+1}, r_{2i+1}[, \ldots\} \text{ (l'ensemble des zones "impaires")}$$

et

$$\bigcup_{\sigma \in P_j^2} \{Z(\sigma)\} = \{[l_2, r_2[, [l_4, r_4[, [l_6, r_6[, \ldots, [l_{2i}, r_{2i}[, \ldots\} \text{ (l'ensemble des zones "paires").}$$

Lemme 19 *Pour tout j ($1 \leq j \leq k$), on a :*

(a) $P_j^1 \cup P_j^2 = S_j^{*R}$

(b) pour tout σ_a, σ_b appartenant à P_j^1 (resp. P_j^2) tel que $\sigma_a \neq \sigma_b$, $Z(\sigma_a) \cap Z(\sigma_b) = \emptyset$

Preuve. La propriété (a) est triviale. Si la propriété suivante $\forall i,\ 1 \leq i \leq n-2,\ r_i \leq l_{i+2}$ est vérifiée, alors la propriété (b) l'est aussi. C'est pourquoi nous allons maintenant prouver cette propriété. Supposons, par contradiction, qu'il existe $i,\ 1 \leq i \leq n-2$, tel que $l_{i+2} < r_i$. Soit $r_{\max} = \max\{r_i, r_{i+1}, r_{i+2}\}$. Trois cas peuvent se produire :

- Si $r_{\max} = r_i$, comme $l_i \leq l_{i+1} \leq l_{i+2}$, on a $l_i \leq l_{i+2} \leq r_{i+2} \leq r_i$. Cela signifie que $[l_{i+2}, r_{i+2}[\subseteq [l_i, r_i[$. S_j^{*R} n'est donc pas minimal ; contradiction.
- Si $r_{\max} = r_{i+1}$, comme $l_i \leq l_{i+1} \leq l_{i+2}$, on a :

$$\begin{cases} & l_i \leq l_{i+1} \leq l_{i+2} < r_i \leq r_{i+2} \leq r_{i+1} \\ \text{ou} & \\ & l_i \leq l_{i+1} \leq l_{i+2} \leq r_{i+2} \leq r_i \leq r_{i+1} \end{cases}$$

  Cela signifie que $[l_{i+2}, r_{i+2}[\subseteq [l_{i+1}, r_{i+1}[$. S_j^{*R} n'est donc pas minimal ; contradiction.
- Si $r_{\max} = r_{i+2}$, comme $l_i \leq l_{i+1} \leq l_{i+2}$, on a :

$$\begin{cases} & l_i \leq l_{i+1} \leq r_{i+1} \leq l_{i+2} < r_i \leq r_{i+2} \\ \text{ou} & \\ & l_i \leq l_{i+1} \leq l_{i+2} \leq r_{i+1} \leq r_i \leq r_{i+2} \\ \text{ou} & \\ & l_i \leq l_{i+1} \leq l_{i+2} < r_i \leq r_{i+1} \leq r_{i+2} \end{cases}$$

  Cela signifie que $[l_{i+1}, r_{i+1}[\subseteq [l_i, r_i[\cup [l_{i+2}, r_{i+2}[$. S_j^{*R} n'est donc pas minimal ; contradiction.

□

Pour énoncé le lemme 20, nous rappelons la définition suivante (voir définition 28 en remplaçant MT par MP).

Définition *Pour chaque machine j ($1 \leq j \leq k$), soit T_j l'ensemble des intervalles associés aux tâches ordonnancées par l'algorithme* MP *sur la machine j, y compris les intervalles provenant des tâches ayant été ordonnancées puis interrompues ultérieurement.*

Lemme 20 *Pour tout j ($1 \leq j \leq k$), on a :*

$$(2\alpha + 2) w(T_j) \geq w(S_j^{*B})$$

Preuve. D'après le lemme 17, on a $\alpha \sum_{\sigma \in S_j^{*R}} w(E(\sigma)) \geq w(S_j^{*R})$. En appliquant le lemme 18, on obtient donc :

$$(\alpha + 1) \sum_{\sigma \in S_j^{*R}} w(E(\sigma)) \ \geq \ w(S_j^{*B}) \tag{4.11}$$

D'après le lemme 19, il existe P_j^1 et P_j^2 tels que pour tout $\sigma_a \neq \sigma_b$ dans P_j^1 (resp. P_j^2), $Z(\sigma_a) \cap Z(\sigma_b) = \emptyset$ et $P_j^1 \cup P_j^2 = S_j^{*R}$. $\bigcup_{\sigma \in P_j^1} Z(\sigma)$ (resp. $\bigcup_{\sigma \in P_j^2} Z(\sigma)$) est donc un ensemble d'intervalles disjoints. Par définition de $E(\sigma)$ et T_j, on obtient donc :

$$w(T_j) \geq \sum_{\sigma \in P_j^1} w(E(\sigma)) \qquad \text{et} \qquad w(T_j) \geq \sum_{\sigma \in P_j^2} w(E(\sigma))$$

En sommant ces deux inégalités, on obtient :

$$2w(T_j) \geq \sum_{\sigma \in P_j^1} w(E(\sigma)) + \sum_{\sigma \in P_j^2} w(E(\sigma)) \tag{4.12}$$

De plus, comme $P_j^1 \cup P_j^2 = S_j^{*R}$, on a :

$$\sum_{\sigma \in P_j^1} w(E(\sigma)) + \sum_{\sigma \in P_j^2} w(E(\sigma)) \geq \sum_{\sigma \in S_j^{*R}} w(E(\sigma)) \tag{4.13}$$

D'après (4.11), (4.12) et (4.13) on obtient alors :

$$(2\alpha + 2) w(T_j) \geq w(S_j^{*B})$$

□

Lemme 21 *On a :*

$$w(S^*) \leq (2\alpha + 3) \sum_{j=1}^{k} w(T_j)$$

Preuve.

- Si $w(S^*) \leq (2\alpha + 3) \sum_{j=1}^{k} w(S_j^{*A})$. Par définition de S_j^{*A}, $\forall \sigma_a \in S_j^{*A}$, $\exists \sigma_b \in \bigcup_{1 \leq j \leq k} T_j$ tel que σ_a et σ_b viennent de la même tâche Γ ($w(\sigma_a) = w(\sigma_b)$). On a donc $\sum_{j=1}^{k} w(S_j^{*A}) \leq \sum_{j=1}^{k} w(T_j)$. On en déduit :

$$w(S^*) \leq (2\alpha + 3) \sum_{j=1}^{k} w(T_j)$$

- Si $w(S^*) \geq (2\alpha + 3) \sum_{j=1}^{k} w(S_j^{*A})$. Par définition de S_j^{*A} et S_j^{*B}, on a $S_j^{*A} \cap S_j^{*B} = \emptyset$ et $S_j^{*A} \cup S_j^{*B} = S_j^*$. On a alors :

$$\begin{array}{lrcl}
 & w(S^*) & = & \sum_{j=1}^{k} w(S_j^{*A}) + \sum_{j=1}^{k} w(S_j^{*B}) \\
\Rightarrow & w(S^*) & \leq & \frac{w(S^*)}{2\alpha+3} + \sum_{j=1}^{k} w(S_j^{*B}) \\
\Rightarrow & \left(1 - \frac{1}{2\alpha+3}\right) w(S^*) & \leq & \sum_{j=1}^{k} w(S_j^{*B}) \\
\Rightarrow & w(S^*) & \leq & \frac{2\alpha+3}{2\alpha+2} \sum_{j=1}^{k} w(S_j^{*B}) \\
\Rightarrow & w(S^*) & \leq & (2\alpha + 3) \sum_{j=1}^{k} w(T_j) \qquad \text{(d'après le lemme 20)}
\end{array}$$

□

Nous rappelons que $\mathsf{w_p}$ est la notation utilisée pour la fonction de *poids avec pénalité* (voir définition 31).

Lemme 22 *Pour tout j ($1 \leq j \leq k$), on a :*

$$\left(1 + \frac{\beta}{\alpha - \beta} + \frac{1}{\alpha - \beta - 1}\right) \mathsf{w_p}(S_j) \geq w(T_j)$$

Preuve. Pour tout $\sigma \in S_j$, pour tout $l \geq 0$, nous définissons récursivement l'ensemble d'intervalles $f_\sigma(l) \subseteq T_j$ de la manière suivante :

- $f_\sigma(0) = \{\sigma\}$
- $f_\sigma(l) = \bigcup_{\sigma' \in f_\sigma(l-1)} \{\sigma'' : \sigma''$ a été interrompu par $\sigma'\}$

Soit $p(\sigma)$ le plus petit entier tel que $f_\sigma(p(\sigma)) = \emptyset$. Pour tout l, $1 \leq l \leq p(\sigma)$, pour tout $\sigma' \in f_\sigma(l-1)$, par définition de l'algorithme MP, on a :

$$w(\sigma') \geq \alpha \cdot w(\{\sigma'' : \sigma'' \text{ a été interrompu par } \sigma'\})$$

De plus, par définition de $\mathsf{w_p}$ (fonction de *poids avec pénalité* β), on a :

$$w(\sigma') = \mathsf{w_p}(\sigma') + \beta \cdot w(\{\sigma'' : \sigma'' \text{ a été interrompu par } \sigma'\}) \tag{4.14}$$

On obtient donc :

$$\mathsf{w_p}(\sigma') \geq (\alpha - \beta) \cdot w(\{\sigma'' : \sigma'' \text{ a été interrompu par } \sigma'\}) \tag{4.15}$$

En sommant cette inégalité pour tous les intervalles de $f_\sigma(l-1)$, on obtient :

$$\sum_{\sigma' \in f_\sigma(l-1)} \mathsf{w_p}(\sigma') \;\geq\; (\alpha - \beta) \cdot w\left(\bigcup_{\sigma' \in f_\sigma(l-1)} \{\sigma'' : \sigma'' \text{ a été interrompu par } \sigma'\}\right)$$

$$\Rightarrow \quad \sum_{\sigma' \in f_\sigma(l-1)} \mathsf{w_p}(\sigma') \;\geq\; (\alpha - \beta) \cdot w(f_\sigma(l)) \quad \Rightarrow \quad \mathsf{w_p}(f_\sigma(l-1)) \;\geq\; (\alpha - \beta) \cdot w(f_\sigma(l))$$

(par définition de $f_\sigma(l)$)

En sommant toutes ces inégalités pour l, $1 \leq l \leq p(\sigma)$, on obtient :

$$\begin{aligned}
& \sum_{l=0}^{p(\sigma)-1} \mathsf{w_p}(f_\sigma(l)) \;\geq\; (\alpha-\beta)\sum_{l=1}^{p(\sigma)} w(f_\sigma(l)) \\
\Rightarrow\quad & \mathsf{w_p}(\sigma) + \sum_{l=1}^{p(\sigma)} \mathsf{w_p}(f_\sigma(l)) \;\geq\; (\alpha-\beta)\sum_{l=1}^{p(\sigma)} w(f_\sigma(l)) \\
& (\text{car } f_\sigma(0) = \{\sigma\}) \\
\Rightarrow\quad & \mathsf{w_p}(\sigma) \;\geq\; (\alpha-\beta-1)\sum_{l=1}^{p(\sigma)} w(f_\sigma(l)) \\
& (\text{car } \forall\sigma',\ w(\sigma') \geq \mathsf{w_p}(\sigma')) \\
\Rightarrow\quad & \frac{\mathsf{w_p}(\sigma)}{\alpha-\beta-1} + w(\sigma) \;\geq\; \sum_{l=0}^{p(\sigma)} w(f_\sigma(l)) \\
& (\text{car } f_\sigma(0) = \{\sigma\}) \\
\Rightarrow\quad & \frac{\mathsf{w_p}(\sigma)}{\alpha-\beta-1} + \mathsf{w_p}(\sigma) + \beta\cdot w(\{\sigma' : \sigma' \text{ a été interrompu par } \sigma\}) \;\geq\; \sum_{l=0}^{p(\sigma)} w(f_\sigma(l)) \\
& (\text{d'après (4.14)}) \\
\Rightarrow\quad & \frac{\mathsf{w_p}(\sigma)}{\alpha-\beta-1} + \mathsf{w_p}(\sigma) + \beta\cdot \frac{\mathsf{w_p}(\sigma)}{\alpha-\beta} \;\geq\; \sum_{l=0}^{p(\sigma)} w(f_\sigma(l)) \\
& (\text{d'après (4.15)})
\end{aligned}$$

En sommant cette inégalité pour tous les intervalles de S_j, et comme $T_j = \bigcup_{\sigma\in S_j}\left(\bigcup_{0\leq l\leq p(\sigma)} f_\sigma(l)\right)$, on obtient :

$$\begin{aligned}
& \left(1+\frac{\beta}{\alpha-\beta}+\frac{1}{\alpha-\beta-1}\right)\sum_{\sigma\in S_j}\mathsf{w_p}(\sigma) \geq \sum_{\sigma\in S_j}\sum_{l=0}^{p(\sigma)} w(f_\sigma(l)) \\
\Rightarrow\quad & \left(1+\frac{\beta}{\alpha-\beta}+\frac{1}{\alpha-\beta-1}\right)\mathsf{w_p}(S_j) \geq w(T_j)
\end{aligned}$$

□

Nous pouvons maintenant donner une preuve du théorème 27.

Preuve du théorème 27.

$$\begin{aligned}
& (2\alpha+3)\textstyle\sum_{j=1}^{k} w(T_j) \;\geq\; w(S^*) && (\text{d'après le lemme 21}) \\
\Rightarrow\quad & (2\alpha+3)\textstyle\sum_{j=1}^{k}\left(1+\frac{\beta}{\alpha-\beta}+\frac{1}{\alpha-\beta-1}\right)\mathsf{w_p}(S_j) \;\geq\; w(S^*) && (\text{d'après le lemme 22}) \\
\Rightarrow\quad & (2\alpha+3)\left(1+\tfrac{\beta}{\alpha-\beta}+\tfrac{1}{\alpha-\beta-1}\right)\mathsf{w_p}(S) \;\geq\; w(S^*)
\end{aligned}$$

□

4.3 Bornes inférieures sur les rapports de compétitivité

Bornes inférieures sur une machine. En ce qui concerne les bornes inférieures, il est important de distinguer le cas d'un système à une seule machine ($k = 1$) de celui à plusieurs machines ($k \geq 2$). En effet, s'il existe plusieurs travaux ayant déjà proposé des bornes inférieures sur le rapport de compétitivité de tout algorithme on-line (aussi bien pour la *taille* que pour le *poids*) pour le cas d'un système à $k = 1$ machine, à notre connaissance, il n'en existe pas pour celui d'un système à plusieurs machines. La raison de ce manque est simple : il est plus aisé de piéger n'importe quel algorithme on-line sur $k = 1$ machine que sur $k \geq 2$ avec un adversaire adaptatif (voir section 1.1 pour un rappel de la définition d'adversaire adaptatif) pour la raison suivante : l'idée est que lorsque l'on ne dispose que d'une machine, l'ordonnancement d'une tâche sur un intervalle occupe la *totalité* des ressources du système pendant cet intervalle. La seule possibilité d'ordonnancer une autre tâche intersectant une tâche déjà ordonnancée consiste donc à interrompre la tâche déjà ordonnancée, décision qui peut se révéler lourde de conséquence en termes de *taille* et *poids* de l'ordonnancement. À l'inverse, lorsque $k \geq 2$, il existe k alternatives différentes pour placer une tâche sur le même intervalle. Dans ces conditions, il est beaucoup plus difficile pour un adversaire de forcer tout algorithme à prendre une décision d'interruption lourde de conséquence, puisque l'interruption d'une tâche déjà ordonnancée pour permettre le placement sur le même intervalle d'une nouvelle tâche ne sera nécessaire que si les k machines sont toutes occupées pendant cet intervalle.

La meilleure borne inférieure existante sur le rapport de compétitivité de tout algorithme pour la *taille* provient de [27]. Dans cet article, J. Garay et al. montrent que tout algorithme on-line pour l'ordonnancement d'intervalles (révélés dans un ordre quelconque) sur $k = 1$ machine a un rapport de compétitivité pour la *taille* en $\Omega(\log \gamma)$, où γ est le rapport entre la longueur de la plus grande tâche et la longueur de la plus petite tâche. Le problème de l'ordonnancement d'intervalles révélés dans un ordre quelconque étant un sous-problème de celui de l'ordonnancement de tâches révélées dans un ordre quelconque, cette borne inférieure s'applique à notre problème (lorsque $k = 1$). Ce résultat montre que l'algorithme MT $(4 \min(\lambda, \lfloor \log_2(\gamma) \rfloor + 1)) -$ compétitif pour la *taille* (avec λ le nombre de longueurs de tâches différentes et γ le rapport entre la longueur de la plus grande tâche et la longueur de la plus petite tâche) est optimal en ordre de grandeur lorsque $k = 1$.

La meilleure borne inférieure existante sur le rapport de compétitivité de tout algorithme pour le *poids* provient de [67]. Dans cet article, G. Woeginger montre que tout algorithme on-line pour l'ordonnancement d'intervalles révélés dans l'ordre croissant de leur bord gauche sur $k = 1$ machine a un rapport de compétitivité pour le *poids* d'au moins 4. Le problème de l'ordonnancement d'intervalles révélés dans l'ordre croissant de leur bord gauche sur $k = 1$ machine étant un sous problème de celui de l'ordonnancement de tâches révélées dans un ordre quelconque sur $k = 1$ machine, cette borne inférieure s'applique à notre problème lorsque $k = 1$.

Bornes inférieures sur k machines. Nous montrons maintenant que tout algorithme on-line a un rapport de compétitivité d'au moins $\min\left(\frac{4}{3}, \frac{2k}{2k-1}\right)$ (resp. $\min\left(\frac{5}{4}, \frac{2k}{2k-1}\right)$) pour la *taille* (resp. le *poids*), où k est le nombre de machines du système. Étant donné que pour tout $k \geq 1$, on a $\frac{2k}{2k-1} > 1$, ce résultat montre qu'il n'existe pas (pour chacun des deux critères) d'algorithme on-line optimal.

Théorème 28 *Soit $k \geq 1$ le nombre de machines du système. Pour tout algorithme on-line A, il existe une séquence on-line de tâches révélées dans l'ordre croissant de leur bord gauche tel que A*

est au moins $\left(\min\left(\frac{4}{3}, \frac{2k}{2k-1}\right)\right)$ *– compétitif pour la* taille *et* $\left(\min\left(\frac{5}{4}, \frac{2k}{2k-1}\right)\right)$ *– compétitif pour le* poids.

Preuve. Nous utilisons pour cette preuve un adversaire adaptatif (voir section 1.1). L'adversaire va soumettre des tâches à ordonnancer parmi les $5k$ tâches suivantes (voir figure 4.2) :

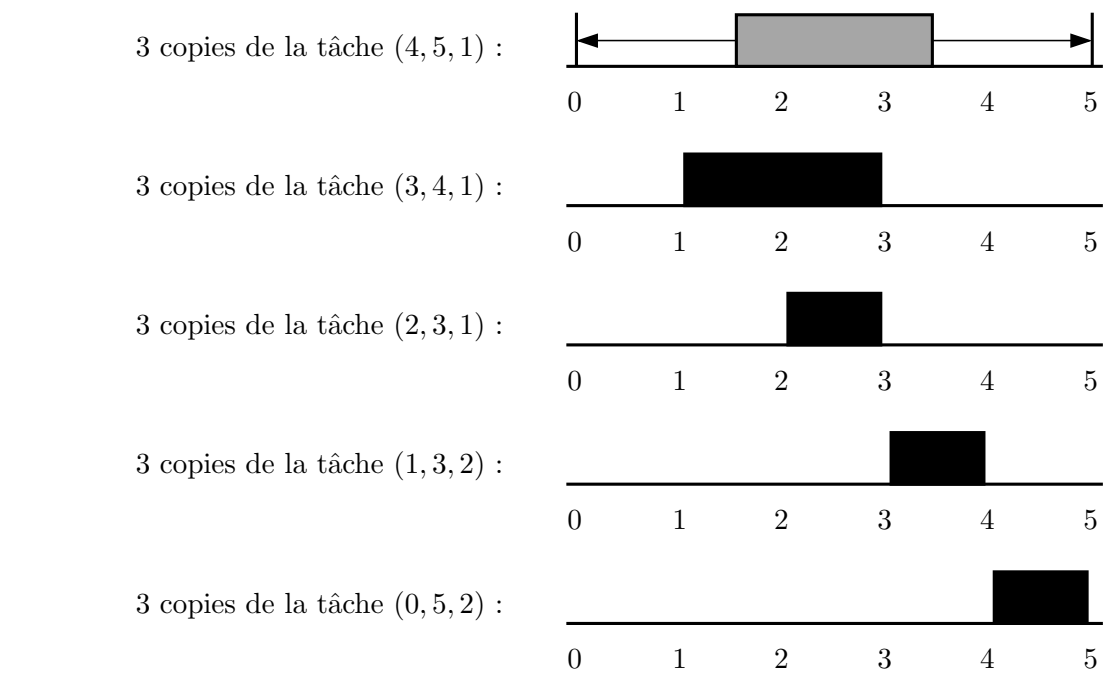
- k copies de la tâche $(0, 5, 2)$
- k copies de la tâche $(1, 3, 2)$ (donc ne pouvant s'exécuter que comme l'intervalle $[1, 3[$)
- k copies de la tâche $(2, 3, 1)$ (donc ne pouvant s'exécuter que comme l'intervalle $[2, 3[$)
- k copies de la tâche $(3, 4, 1)$ (donc ne pouvant s'exécuter que comme l'intervalle $[3, 4[$)
- k copies de la tâche $(4, 5, 1)$ (donc ne pouvant s'exécuter que comme l'intervalle $[4, 5[$).

Nous décrivons maintenant le comportement de l'adversaire. Soit A un algorithme on-line quelconque et soit S l'ordonnancement courant construit par A. L'adversaire soumet d'abord les k copies de la tâche $(0, 5, 2)$ puis les k copies de la tâche $(1, 3, 2)$. Deux comportements de l'algorithme A sont à distinguer :

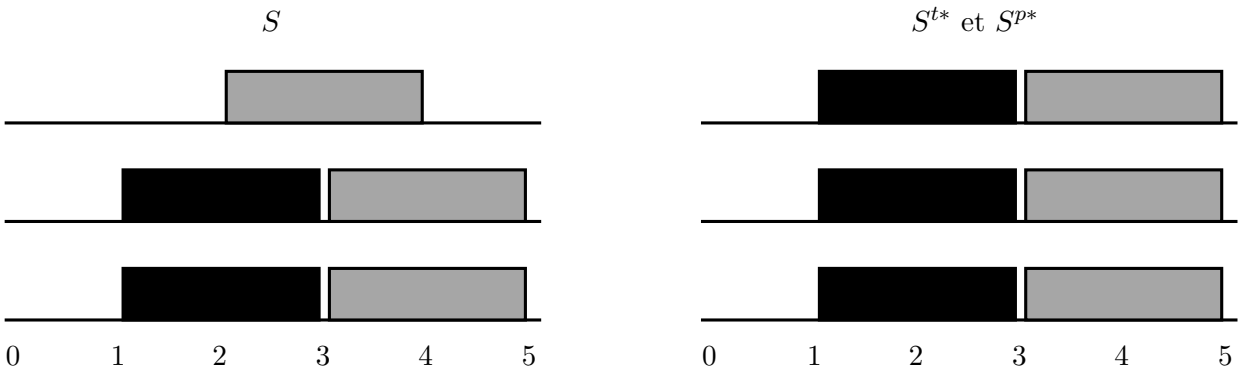
- Si A n'ordonnance pas la totalité de ces $2k$ tâches, alors l'adversaire ne soumet plus d'intervalle et on a $|S| \leq 2k - 1$ et $w(S) \leq 4k - 2$. Or l'ordonnancement optimal S^{t*} pour la *taille* (resp. S^{p*} pour le *poids*) consiste ici à placer les k copies de $(1, 3, 2)$ (une sur chaque machine, s'exécutant comme l'intervalle $[1, 3[$) et les k copies de $(0, 5, 2)$ (une sur chaque machine, s'exécutant comme l'intervalle $[3, 5[$). On a donc $|S^{t*}| = 2k$ et $w(S^{p*}) = 4k$ (voir le scénario n°1 de la figure 4.2). A a donc un rapport de compétitivité pour la *taille* (resp. le poids) d'au moins :
$$\frac{|S^{t*}|}{|S|} = \frac{w(S^{p*})}{w(S)} = \frac{2k}{2k-1}$$
Notons que si A n'ordonnance aucune tâche, alors le rapport de compétitivité est infini.
- Sinon, A a ordonnancé la totalité de ces $2k$ tâches et les exécute dans la configuration optimale décrite ci-dessus, c'est-à-dire entre autres avec les k copies de la tâche $(0, 5, 2)$ exécutées comme l'intervalle $[3, 5[$ (il s'agit de la seule configuration possible pour placer la totalité des $2k$ tâches). L'adversaire soumet alors les k copies de la tâche $(2, 3, 1)$, puis les k copies de la tâche $(3, 4, 1)$, puis les k copies de la tâche $(4, 5, 1)$. L'ordonnancement optimal pour la *taille* (resp. le *poids*) est alors composé de $4k$ tâches (les k copies de $(0, 5, 2)$, chacune ordonnancée sur une machine comme l'intervalle $[0, 2[$, les k copies de la tâche $(2, 3, 1)$, chacune ordonnancée sur une machine comme l'intervalle $[2, 3[$, les k copies de la tâche $(3, 4, 1)$, chacune ordonnancée sur une machine comme l'intervalle $[3, 4[$ et les k copies de la tâche $(4, 5, 1)$, chacune ordonnancée sur une machine comme l'intervalle $[4, 5[$). Or à partir de l'instant où la première des k copies de la tâche $(2, 3, 1)$ est révélée, A ne pourra plus ordonnancer qu'un maximum de 3 tâches par machine (voir le scénario n°2 de la figure 4.2). On a donc $|S| \leq 3k$ et $w(S) \leq 4k$ alors que $|S^{t*}| = 4k$ et $w(S^{p*}) = 5k$. A a donc un rapport de compétitivité pour la *taille* (resp. le *poids*) d'au moins :
$$\frac{|S^{t*}|}{|S|} = \frac{4k}{3k} = \frac{4}{3} \qquad \text{et} \qquad \frac{w(S^{p*})}{w(S)} = \frac{5k}{4k} = \frac{5}{4}$$

La figure 4.2 illustre chacun des deux scénarios que l'adversaire adaptatif peut proposer en fonction des choix de l'algorithme A (avec $k = 3$ machines). □

Nous soulignons que le théorème 28 reste vrai même si nous imposons que les tâches soient révélées dans l'ordre croissant de leur bord gauche et même si nous autorisons la reprise de l'exécution



Scénario n°1 :



Scénario n°2 :

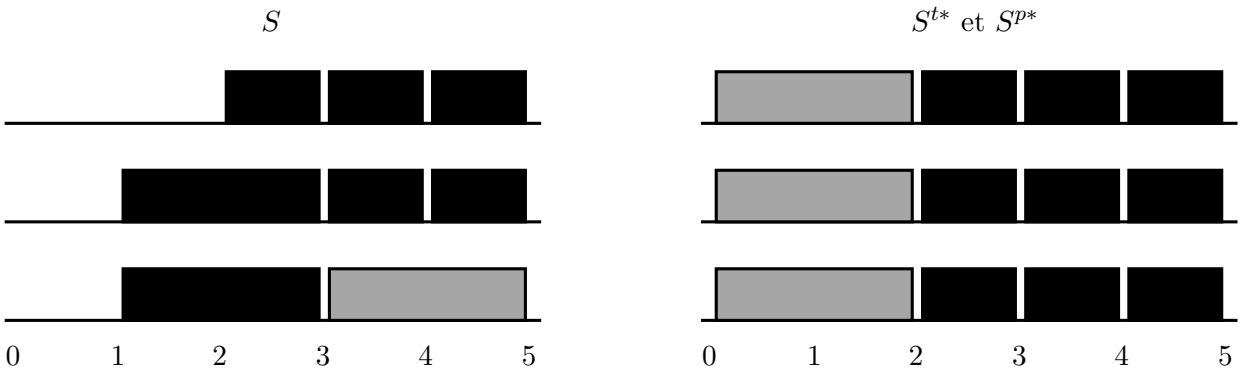


FIG. 4.2 – Illustration de l'adversaire adaptatif avec $k = 3$

d'une tâche interrompue (la reprise de l'exécution doit toutefois se faire depuis le début de la tâche, le morcellement d'une tâche n'étant pas autorisé). En effet, la preuve du théorème 28 reste la même : la séquence proposée par l'adversaire adaptatif (qui soumet effectivement les tâches dans l'ordre croissant de leur bord gauche) est telle qu'un algorithme ne pourra pas bénéficier de l'assouplissement du modèle concernant la possibilité de reprise de l'exécution d'une tâche.

Nous ajoutons que le théorème 28 montre également que tout algorithme on-line est au moins $\left(\min\left(\frac{5}{4}, \frac{2k}{2k-1}\right)\right)$ – compétitif pour le *poids avec pénalité*, celui-ci étant une généralisation du *poids*.

Bilan du chapitre 4. Dans la section 4.1, nous avons proposé un algorithme on-line $(4\min(\lambda, \lfloor \log_2(\gamma) \rfloor + 1))$ – compétitif pour la *taille* pour l'ordonnancement de tâches sur $k \geq 1$ machines (appelé MT), où λ est le nombre de longueurs de tâches différentes parmi les tâches révélées et γ le rapport entre la longueur de la plus grande tâche et la longueur de la plus petite tâche parmi les tâches révélées (remarquons que l'algorithme MT n'a besoin de connaître ni λ ni γ à l'avance). D'après [27], le rapport de compétitivité de MT est optimal en ordre de grandeur pour le cas particulier d'un système à $k = 1$ machine. Nous soulignons le fait que dans de nombreuses situations (correspondant aux séquences de tâches "homogènes"), ce rapport de compétitivité est constant. Par exemple, lorsque $\lambda = c$ (avec c une constante), l'algorithme MT est $4c$ – compétitif. Lorsque $\gamma = 2^{c'}$ (avec c' une constante), l'algorithme MT est $(4c' + 4)$ – compétitif. Dans la section 4.2, nous avons proposé un algorithme on-line $\left((2\alpha + 3)\left(1 + \frac{\beta}{\alpha-\beta} + \frac{1}{\alpha-\beta-1}\right)\right)$ – compétitif pour le *poids avec pénalité* pour l'ordonnancement de tâches sur k machines (appelé MP), où β est la constante de pénalité représentant la perte de profit pour l'opérateur associée à l'interruption d'une tâche déjà ordonnancée, et α une constante à choisir en fonction de β pour que le rapport de compétitivité de MP soit le plus faible possible. Nous soulignons que β et α étant constants, le rapport de compétitivité de MP est constant. Enfin, nous avons montré dans 4.3 qu'il n'existe d'algorithme optimal ni pour le problème de la maximisation de la *taille*, ni pour le problème de la maximisation du *poids* (et donc du *poids avec pénalité*), même si nous relâchons au maximum les contraintes de notre modèle on-line (en révélant les tâches par ordre croissant de leur bord gauche et en autorisant la reprise d'une tâche interrompue depuis le début de son exécution).

Chapitre 5

Ordonnancements bicritères d'intervalles

L'objet de ce chapitre est la maximisation *simultanée* de la *taille* et du *poids* pour la construction d'ordonnancements on-line. Nous proposons un algorithme pour l'ordonnancement d'intervalles sur $k \geq 4$ machines dont les rapports de compétitivité sont simultanément constants pour chacun des deux critères. Les travaux présentés dans ce chapitre ont été réalisés en collaboration avec Fabien Baille (voir [10, 13, 12]).

5.1 Introduction et notations

Le problème que nous traitons dans ce chapitre est moins général que celui traité dans le chapitre 7 du point de vue du modèle on-line considéré : nous traitons ici l'ordonnancement d'*intervalles* (c'est-à-dire de tâches (l, r, p) telles que $p = r - l$) révélés dons l'ordre croissant de leur bord gauche. En revanche, cette perte de généralité nous permet d'obtenir en termes de garanties un résultat beaucoup plus fort, puisque nous montrons que l'algorithme que nous proposons dans ce chapitre est *simultanément* compétitif pour la *taille* et le *poids* (avec un rapport de compétitivité constant pour chacun des critères). Le problème que nous proposons de résoudre est donc un problème *bicritère*.

Notations. Pour simplifier la manipulation des algorithmes que nous allons utiliser et proposer, pour tout algorithme A, on notera A^k la version de l'algorithme A s'exécutant sur $k \geq 1$ machine(s). Nous précisons dans ce chapitre le nombre de machines utilisées par chaque algorithme pour éviter toute confusion car l'algorithme bicritère que nous proposons utilise deux algorithmes comme sous-programme, chacun étant exécuté sur l (resp. $k - l$) machines.

Dans la suite du chapitre, un algorithme bicritère A est dit $(\rho, \mu) -$ compétitif lorsque A est simultanément $\rho -$ compétitif pour la *taille* et $\mu -$ compétitif pour le *poids*.

Travaux précédents. La version off-line du problème bicritère que nous présentons (pour l'ordonnancement d'intervalles) a été traitée dans [11], où un algorithme $(\frac{k}{l}, \frac{k}{k-l}) -$approché $(1 \leq l < k)$ a été proposé. Chacun des deux problèmes monocritères (dans leur version off-line) est polynomial (voir [23] pour la *taille* et [7, 19] pour le *poids*). Dans le contexte on-line, l'algorithme GOL proposé dans [23] est optimal pour la *taille* et des algorithmes avec rapport de compétitivité constant pour le *poids* ont été proposés dans [15, 67]. Enfin, nous soulignons qu'à notre connaissance, le seul article traitant un problème d'ordonnancement de manière on-line *et* bicritère est [17]. Le problème considéré est l'ordonnancement sur k machines identiques et indépendantes de tâches définies par

un temps d'exécution et une taille mémoire. Le but est alors de minimiser simultanément le makespan pour le temps (c'est-à-dire la date de complétude de la dernière tâche ordonnancée) et le makespan pour la mémoire (c'est-à-dire la valeur maximum des sommes des tailles mémoires sur chaque machine). Les auteurs proposent alors (entre autres) un algorithme résolvant le problème ainsi que son analyse en termes de rapport de compétitivité pour chacun des critères.

Plan du chapitre. Dans la section 5.2, nous présentons et analysons un algorithme on-line bicritère pour la maximisation simultanée de la *taille* et du *poids*. Enfin nous présentons en section 5.3 des bornes inférieures sur les rapports de compétitivité atteignables simultanément.

5.2 Un algorithme bicritère pour la *taille* et le *poids*

Dans cette section, nous présentons un algorithme on-line bicritère maximisant simultanément la *taille* et le *poids* pour l'ordonnancement d'intervalles révélés dans l'ordre croissant de leur bord gauche. Pour cela, nous avons besoin des algorithmes on-line monocritères suivants : l'algorithme GOL^k (pour Greedy On-line Algorithm sur k machines), proposé par Faigle et Nawijn dans [23], pour la maximisation de la *taille* et l'algorithme LR^k (pour Left Right sur $k \geq 3$ machines), proposé par Bar-Noy et al. dans [15], pour la maximisation du *poids*. Ceux deux algorithmes vont être utilisés comme sous-programmes de notre algorithme bicritère.

Étant donné que nous n'avons pas besoin de connaître le contenu technique de ces deux algorithmes pour les utiliser dans notre algorithme bicritère (seul le résultat donné par chacun des deux algorithmes importe), nous ne présentons GOL^k et LR^k qu'en annexe (voir annexe 3). Afin de faciliter l'utilisation de ces deux algorithmes comme sous-programmes de notre algorithme bicritère, nous soulignons que dans le modèle que nous considérons, tout algorithme on-line peut être décrit en deux phases, de la manière suivante.

Forme générique d'un algorithme on-line A^k

1 Soit k le nombre de machines du système.
2 Soit σ le nouvel intervalle révélé
3 **Phase d'interruption :**
4 Décider quel(s) intervalle(s) interrompre.
5 **Phase d'ordonnancement :**
6 Décider si σ est rejeté ou ordonnancé,
7 Le cas échéant, décider sur quelle machine ordonnancer σ.

Nous donnons maintenant l'énoncé des deux lemmes prouvant la compétitivité de GOL^k et LR^k pour chacun des critères étudiés. Ces lemmes seront utilisés dans la suite de cette section pour l'analyse de la compétitivité de notre algorithme bicritère. Le lemme 23 (resp. 24) provient de [23] (resp. [13]).

Lemme 23 *Pour tout $k \geq 1$, l'algorithme GOL^k est $1-$ compétitif (c'est-à-dire optimal) pour la* taille.

Lemme 24 *Pour tout $k \geq 3$, l'algorithme LR^k est $(\frac{2}{1-\frac{2}{k}})-$ compétitif pour le* poids.

L'algorithme bicritère MTP^k**.** L'idée principale de l'algorithme MTP^k (pour Maximisation de la Taille et du Poids) est la suivante. MTP^k est exécuté sur les k machines *réelles* du système. Ces machines sont dites réelles car c'est sur celles-ci que l'ordonnancement final sera effectivement construit. MTP^k utilise les algorithmes GOL^k et LR^k (voir annexe pour une définition précise de GOL et de LR) comme sous-programmes. Plus précisément, pour chaque nouvel intervalle σ révélé, l'algorithme MTP^k (avec $k \geq 4$) simule l'exécution de GOL^l (avec $1 \leq l \leq k-3$) sur l machines *virtuelles* et l'exécution de LR^{k-l} sur $k-l$ autres machines *virtuelles*, dans le but de contrôler la *taille* et le *poids* de l'ordonnancement. Ces deux simulations sont faites sur des machines dites *virtuelles* car elles n'ont lieu que dans le but de déterminer l'ensemble (potentiellement vide) d'intervalles que MTP^k doit interrompre ainsi que pour décider si MTP^k doit ordonnancer ou rejeter le nouvel intervalle σ révélé. En effet, MTP^k choisit d'ordonnancer σ sur une machine *réelle* si et seulement si celui-ci est ordonnancé par GOL^l sur une de ses l machines *virtuelles* et par LR^{k-l} sur une de ses $k-l$ machines *virtuelles*.

Pour définir l'algorithme MTP^k, nous avons besoin des définitions suivantes.

Définition 38 (Ordonnancement après la phase d'interruption (resp. ordonnancement)) *Pour tout algorithme* A*, on note* $S_{i_1}(\mathsf{A})$ *(resp.* $S_{i_2}(\mathsf{A})$*) l'ordonnancement construit par* A *après la phase d'interruption (resp. ordonnancement) de l'étape* i*.*

Définition 39 (Ensemble d'intervalles $R_{i_1}(\mathsf{MTP}^k)$ **(resp.** $R_{i_2}(\mathsf{MTP}^k)$**))** *On note* $R_{i_1}(\mathsf{MTP}^k)$ *(resp.* $R_{i_2}(\mathsf{MTP}^k)$*) l'ensemble des intervalles ordonnancés (et non interrompus) par l'algorithme* MTP^k *après la phase d'interruption (resp. ordonnancement) de l'étape* i *sur les* k *machines associées à* MTP^k*, appelées machines* réelles*.*

Définition 40 (Ensembles d'intervalles $V_{i_1}(\mathsf{GOL}^l)$**,** $V_{i_2}(\mathsf{GOL}^l)$**,** $V_{i_1}(\mathsf{LR}^{k-l})$ **et** $V_{i_2}(\mathsf{LR}^{k-l})$**)** *On note* $V_{i_1}(\mathsf{GOL}^l)$ *(resp.* $V_{i_2}(\mathsf{GOL}^l)$*) l'ensemble des intervalles ordonnancés (et non interrompus) par l'algorithme* GOL^l *après la phase d'interruption (resp. d'ordonnancement) de l'étape* i *sur les* l *machines associées à* GOL^l*, appelées machines* virtuelles*.*

De la même façon, on note $V_{i_1}(\mathsf{LR}^{k-l})$ *(resp.* $V_{i_2}(\mathsf{LR}^{k-l})$*) l'ensemble des intervalles ordonnancés (et non interrompus) par l'algorithme* LR^{k-l} *après la phase d'interruption (resp. ordonnancement) de l'étape* i *sur les* $k-l$ *machines associées à* LR^{k-l}*, appelées machines* virtuelles*.*

Algorithme Maximisation de la Taille et du Poids – MTP^k

Soient $k \geq 4$ le nombre de machines système et l un entier tel que $1 \leq l \leq k-3$.
À l'étape 0, on a $V_{0_2}(\mathsf{GOL}^l) = V_{0_2}(\mathsf{LR}^{k-l}) = R_{0_2}(\mathsf{MTP}^k) = \emptyset$.
À l'étape i, soit σ_i le nouvel intervalle révélé.
Phase d'interruption :
    Exécuter la phase d'interruption de GOL^l (resp. LR^{k-l}) sur les l (resp. $k-l$) machines virtuelles associées à GOL^l (resp. LR^{k-l}) en soumettant le nouvel intervalle σ_i à GOL^l (resp. LR^{k-l}). On obtient alors :

$$V_{i_1}(\mathsf{GOL}^l) \text{ (resp. } V_{i_1}(\mathsf{LR}^{k-l}))$$

    Exécuter la phase d'interruption de MTP^k sur les k machines réelles associées à MTP^k en interrompant (si nécessaire) les intervalles de $R_{(i-1)_2}(\mathsf{MTP}^k)$ de sorte que l'on obtienne :

$$R_{i_1}(\mathsf{MTP}^k) \;=\; V_{i_1}(\mathsf{GOL}^l) \cup V_{i_1}(\mathsf{LR}^{k-l})$$

Phase d'ordonnancement :
    Exécuter la phase d'ordonnancement de GOL^l (resp. LR^{k-l}) sur les l (resp. $k-l$) machines virtuelles associées à GOL^l (resp. LR^{k-l}) en ordonnançant ou rejetant le nouvel intervalle σ_i.
    Si GOL^l et LR^{k-l} rejettent σ_i
    Alors MTP^k rejette σ_i. On obtient alors :

$$R_{i_2}(\mathsf{MTP}^k) \;=\; R_{i_1}(\mathsf{MTP}^k)$$

    Si GOL^l ou LR^{k-l} ordonnance(nt) σ_i
    Alors MTP^k ordonnance σ_i sur une machine libre parmi les k machines réelles associées à MTP^k. On obtient alors :

$$R_{i_2}(\mathsf{MTP}^k) \;=\; R_{i_1}(\mathsf{MTP}^k) \cup \{\sigma_i\}$$

Analyse de l'algorithme MTP^k. Le théorème 29 montre que l'algorithme MTP^k est $\left(\frac{k}{l}, \frac{2k}{k-l-2}\right)-$compétitif, c'est-à-dire que MTP^k est simultanément $\left(\frac{k}{l}\right)-$compétitif pour la *taille* et $\left(\frac{2k}{k-l-2}\right)-$compétitif pour le *poids*. Par exemple, si on pose $l = \frac{k-2}{3}$, MTP^k est $\left(\frac{3}{1-\frac{2}{k}}, \frac{3}{1-\frac{2}{k}}\right)-$compétitif. Dans ce cas, on a $\frac{3}{1-\frac{2}{k}} \leq 6$ et $\lim_{k\to\infty} \frac{3}{1-\frac{2}{k}} = 3$. Le couple de rapports de compétitivité obtenu est donc paramétrable en fonction de l (le nombre de machines dédiées à la maximisation de la *taille*) ; plus l est grand (resp. petit), plus le rapport de compétitivité de MTP^k pour la *taille* est petit (resp. grand) et plus le rapport de compétitivité de MTP^k pour le *poids* est grand (resp. petit). Le tableau 5.1 récapitule les couples de rapport de compétitivité atteignables pour $k = 20$.

Théorème 29 *Pour tout $k \geq 4$, pour tout l, $1 \leq l \leq k-3$, l'algorithme MTP^k est $\left(\frac{k}{l}, \frac{2k}{k-l-2}\right)-$compétitif pour la* taille *et le* poids.

l	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
ρ	20	10	6.66	5	4	3.33	2.86	2.5	2.22	2	1.82	1.66	1.54	1.43	1.33	1.25	1.18
μ	2.35	2.5	2.66	2.86	3.08	3.33	3.64	4	4.44	5	5.71	6.66	8	10	13.33	20	40

TAB. 5.1 – Les couples de rapport de compétitivité (ρ, μ) de MTP^k pour $k = 20$ en fonction de l

Pour prouver le théorème 29, nous avons besoin du lemme 25 et du corollaire 4. Le lemme 25 montre que l'algorithme MTP^k ordonnance le même ensemble d'intervalles que l'union des ensembles d'intervalles ordonnancés par les algorithmes GOL^l et LR^{k-l}.

Lemme 25 *À chaque étape d'exécution i de l'algorithme MTP^k, l'ordonnancement $S_{i_2}(\mathsf{MTP}^k)$ est valide et on a :*

$$R_{i_2}(\mathsf{MTP}^k) \;=\; V_{i_2}(\mathsf{GOL}^l) \cup V_{i_2}(\mathsf{LR}^{k-l})$$

Preuve. Nous prouvons le lemme 25 par récurrence sur les étapes d'exécution successives i de l'algorithme MTP^k.

Cas de base :
Par définition de MTP^k, on a $V_{0_2}(\mathsf{GOL}^l) = V_{0_2}(\mathsf{LR}^{k-l}) = R_{0_2}(\mathsf{MTP}^k) = \emptyset$. $S_{0_2}(\mathsf{MTP}^k)$ est donc valide et on a :

$$R_{0_2}(\mathsf{MTP}^k) \;=\; V_{0_2}(\mathsf{GOL}^l) \cup V_{0_2}(\mathsf{LR}^{k-l})$$

Cas général :
On suppose que $S_{(i-1)_2}(\mathsf{MTP}^k)$ est valide et que $R_{(i-1)_2}(\mathsf{MTP}^k) = V_{(i-1)_2}(\mathsf{GOL}^l) \cup V_{(i-1)_2}(\mathsf{LR}^{k-l})$ (hypothèses de récurrence).

1. **Phase d'interruption :**
   Nous devons d'abord montrer que $R_{i_1}(\mathsf{MTP}^k) = V_{i_1}(\mathsf{GOL}^l) \cup V_{i_1}(\mathsf{LR}^{k-l})$ et que $S_{i_1}(\mathsf{MTP}^k)$ est valide.
   (a) Par définition, MTP^k interrompt le sous-ensemble d'intervalles de $R_{(i-1)_2}(\mathsf{MTP}^k)$ tel que :

   $$R_{i_1}(\mathsf{MTP}^k) = V_{i_1}(\mathsf{GOL}^l) \cup V_{i_1}(\mathsf{LR}^{k-l}) \qquad \text{(UNION)}$$

   Nous devons montrer qu'il existe toujours un sous-ensemble d'intervalles de $R_{(i-1)_2}(\mathsf{MTP}^k)$ pouvant être interrompus de sorte que (UNION) soit possible.
   Comme $V_{i_1}(\mathsf{GOL}^l) \subseteq V_{(i-1)_2}(\mathsf{GOL}^l)$, $V_{i_1}(\mathsf{LR}^{k-l}) \subseteq V_{(i-1)_2}(\mathsf{LR}^{k-l})$, et $R_{(i-1)_2}(\mathsf{MTP}^k) = V_{(i-1)_2}(\mathsf{GOL}^l) \cup V_{(i-1)_2}(\mathsf{LR}^{k-l})$ (par hypothèse de récurrence), on a bien :

   $$V_{i_1}(\mathsf{GOL}^l) \cup V_{i_1}(\mathsf{LR}^{k-l}) \subseteq R_{(i-1)_2}(\mathsf{MTP}^k)$$

   (b) Par définition, durant sa phase d'interruption, MTP^k ne fait qu'interrompre des intervalles ordonnancés dans $S_{(i-1)_2}(\mathsf{MTP}^k)$ (valide par hypothèse de récurrence). On a donc :

$$S_{i_1}(\mathsf{MTP}^k) \text{ est valide} \qquad \text{(VALIDE)}$$

2. **Phase d'ordonnancement :**
Nous devons maintenant montrer que $R_{i_2}(\mathsf{MTP}^k) = V_{i_2}(\mathsf{GOL}^l) \cup V_{i_2}(\mathsf{LR}^{k-l})$ et que $S_{i_2}(\mathsf{MTP}^k)$ est valide. Par définition de MTP^k, on a :

   (a) Si GOL^l et LR^{k-l} rejettent σ_i, alors MTP^k rejette σ_i et on a :

   i.
   $$\begin{aligned} R_{i_2}(\mathsf{MTP}^k) &= R_{i_1}(\mathsf{MTP}^k) = V_{i_1}(\mathsf{GOL}^l) \cup V_{i_1}(\mathsf{LR}^{k-l}) \\ &\quad \text{(par définition de } \mathsf{MTP}^k \text{ et d'après (UNION))} \\ &= V_{i_2}(\mathsf{GOL}^l) \cup V_{i_2}(\mathsf{LR}^{k-l}) \\ &\quad \text{(puisque } \mathsf{GOL}^l \text{ et } \mathsf{LR}^{k-l} \text{ rejettent } \sigma_i, \text{ on a} \\ &\quad V_{i_1}(\mathsf{GOL}^l) = V_{i_2}(\mathsf{GOL}^l) \text{ et } V_{i_1}(\mathsf{LR}^{k-l}) = V_{i_2}(\mathsf{LR}^{k-l})) \end{aligned}$$

   ii. $S_{i_2}(\mathsf{MTP}^k) = S_{i_1}(\mathsf{MTP}^k)$. Comme $S_{i_1}(\mathsf{MTP}^k)$ est valide (d'après (VALIDE)), $S_{i_2}(\mathsf{MTP}^k)$ l'est aussi.

   (b) Si GOL^l (resp. LR^{k-l}) ordonnance σ_i et LR^{k-l} (resp. GOL^l) rejette σ_i, alors MTP^k ordonnance σ_i sur une machine libre et on a :

   i.
   $$\begin{aligned} R_{i_2}(\mathsf{MTP}^k) &= R_{i_1}(\mathsf{MTP}^k) \cup \{\sigma_i\} = V_{i_1}(\mathsf{GOL}^l) \cup V_{i_1}(\mathsf{LR}^{k-l}) \cup \{\sigma_i\} \\ &\quad \text{(par définition de } \mathsf{MTP}^k \text{ et d'après (UNION))} \\ &= V_{i_2}(\mathsf{GOL}^l) \cup V_{i_2}(\mathsf{LR}^{k-l}) \\ &\quad \text{(comme } \mathsf{GOL}^l \text{ (resp. } \mathsf{LR}^{k-l}) \text{ ordonnance } \sigma_i \text{ et } \mathsf{LR}^{k-l} \\ &\quad \text{(resp. } \mathsf{GOL}^l) \text{ rejette } \sigma_i, \text{ on a } V_{i_2}(\mathsf{GOL}^l) = V_{i_1}(\mathsf{GOL}^l) \cup \{\sigma_i\} \\ &\quad \text{(resp. } V_{i_2}(\mathsf{LR}^{k-l}) = V_{i_1}(\mathsf{LR}^{k-l}) \cup \{\sigma_i\}) \text{ et } V_{i_2}(\mathsf{LR}^{k-l}) = \\ &\quad V_{i_1}(\mathsf{LR}^{k-l}) \text{ (resp. } V_{i_2}(\mathsf{GOL}^l) = V_{i_1}(\mathsf{GOL}^l)) \end{aligned}$$

   ii. Comme $S_{i_1}(\mathsf{MTP}^k)$ est valide (d'après (VALIDE)) et comme $S_{i_2}(\mathsf{MTP}^k)$ est construit par MTP^k en ajoutant σ_i à $S_{i_1}(\mathsf{MTP}^k)$ seulement une fois, l'unique raison pour laquelle $S_{i_2}(\mathsf{MTP}^k)$ pourrait ne pas être valide serait parce que $\sigma_i = [l_i, r_i[$ est ordonnancé par MTP^k à l'instant l_i alors qu'il n'existe pas de machine libre à cet instant, c'est-à-dire parce qu'il existe au moins $k+1$ intervalles de $R_{i_2}(\mathsf{MTP}^k)$ ordonnancés à l'instant l_i par MTP^k. Nous allons montrer que cette situation est impossible. En effet, comme GOL^l et LR^{k-l} construisent à chaque étape un ordonnancement valide, il y a au plus $l + k - l = k$ intervalles de $V_{i_2}(\mathsf{GOL}^l) \cup V_{i_2}(\mathsf{LR}^{k-l})$ ordonnancés à l'instant l_i par GOL^l et LR^{k-l}. On en déduit qu'il y a au plus k intervalles de $R_{i_2}(\mathsf{MTP}^k)$ ordonnancés à l'instant l_i par MTP^k (car nous venons juste de prouver ci-dessus que $R_{i_2}(\mathsf{MTP}^k) = V_{i_2}(\mathsf{GOL}^l) \cup V_{i_2}(\mathsf{LR}^{k-l})$). L'ordonnancement $S_{i_2}(\mathsf{MTP}^k)$ est donc valide.

   (c) Si GOL^l et LR^{k-l} ordonnancent σ_i, alors MTP^k ordonnance σ_i sur une machine libre et on a :

i.
$$\begin{aligned} R_{i_2}(\mathsf{MTP}^k) &= R_{i_1}(\mathsf{MTP}^k) \cup \{\sigma_i\} = V_{i_1}(\mathsf{GOL}^l) \cup V_{i_1}(\mathsf{LR}^{k-l}) \cup \{\sigma_i\} \\ &\quad \text{(par définition de } \mathsf{MTP}^k \text{ et d'après (UNION))} \\ &= V_{i_2}(\mathsf{GOL}^l) \cup V_{i_2}(\mathsf{LR}^{k-l}) \\ &\quad \text{(comme } \mathsf{GOL}^l \text{ et } \mathsf{LR}^{k-l} \text{ ordonnancent } \sigma_i \text{ on a } V_{i_2}(\mathsf{GOL}^l) = \\ &\quad V_{i_1}(\mathsf{GOL}^l) \cup \{\sigma_i\} \text{ et } V_{i_2}(\mathsf{LR}^{k-l}) = V_{i_1}(\mathsf{LR}^{k-l}) \cup \{\sigma_i\}) \end{aligned}$$

ii. Nous prouvons que $S_{i_2}(\mathsf{MTP}^k)$ est valide de la même façon qu'en 2(b)ii.

□

Corollaire 4 *À chaque étape d'exécution i de l'algorithme MTP^k, on a :*

$$|V_{i_2}(\mathsf{GOL}^l)| \leq |R_{i_2}(\mathsf{MTP}^k)| \quad \textit{et} \quad w(V_{i_2}(\mathsf{LR}^{k-l})) \leq w(R_{i_2}(\mathsf{MTP}^k))$$

Preuve. Le corollaire 4 est une conséquence immédiate du lemme 25, puisqu'à chaque étape d'exécution i de l'algorithme MTP^k, on a $R_{i_2}(\mathsf{MTP}^k) = V_{i_2}(\mathsf{GOL}^l) \cup V_{i_2}(\mathsf{LR}^{k-l})$. □

Nous pouvons maintenant donner une preuve du théorème 29. L'idée de ce théorème est d'évaluer la dégradation du rapport de compétitivité, pour le *poids* et la *taille*, due à la *limitation* du nombre de machines allouées à chaque critère : l machines pour la *taille* (resp. $k-l$ machines pour le *poids*) au lieu de la totalité des k machines. Cette idée peut être mise en parallèle avec certains travaux sur l'*augmentation de ressources* pour les ordonnancements on-line (voir [42, 45, 53]), où le procédé inverse du nôtre est envisagé. En effet, l'idée de l'augmentation de ressources est de se donner plus de ressources que la quantité mise à disposition pour le calcul de la solution off-line optimale. Cette augmentation peut se traduire par l'ajout de machines supplémentaires ou le remplacement des machines par d'autres plus rapides. L'objectif est alors l'analyse de l'impact de cette augmentation sur le rapport de compétitivité d'un algorithme.

Preuve du théorème 29. Soit $\sigma_1, \cdots, \sigma_i$ une séquence on-line d'intervalles et soit S_x^{t*} (resp. S_x^{p*}) un ordonnancement optimal de $\{\sigma_1, \cdots, \sigma_i\}$ pour la *taille* (resp. pour le *poids*) sur $x \leq k$ machines. Soit S_l^{GOL} (resp. S_{k-l}^{LR}) l'ordonnancement construit par GOL^l (resp. LR^{k-l}) pour la séquence $\sigma_1, \cdots, \sigma_i$ sur $l \leq k-3$ (resp. $k-l \geq 3$) machines. Comme GOL^l est $1-$compétitif (d'après le lemme 23) et LR^{k-l} est $\left(\frac{2}{1-\frac{2}{k-l}}\right)-$compétitif (d'après le lemme 24), on a :

$$|S_l^{t*}| \leq |S_l^{\mathsf{GOL}}| \qquad (\text{resp. } w(S_{k-l}^{p*}) \leq \left(\frac{2}{1-\frac{2}{k-l}}\right) w(S_{k-l}^{\mathsf{LR}}) \ ) \tag{5.1}$$

Soit S'^t (resp. S'^p) le sous-ordonnancement sur l (resp. $k-l$) machines de S_k^{t*} (resp. S_k^{p*}) exécutant tous les intervalles ordonnancés sur les l (resp. $k-l$) machines de S_k^{t*} (resp. S_k^{p*}) générant la plus grande *taille* (resp. le plus grand *poids*). Puisque S'^t (resp. S'^p) est un ordonnancement sur l (resp. $k-l$) machines, on a :

$$|S'^t| \leq |S_l^{t*}| \qquad (\text{resp. } w(S'^p) \leq w(S_{k-l}^{p*}) \ )$$

En combinant cette inégalité avec (5.1), on obtient :

$$|S'^t| \leq |S_l^{\mathsf{GOL}}| \qquad (\text{resp. } w(S'^p) \leq \left(\frac{2}{1-\frac{2}{k-l}}\right) w(S_{k-l}^{\mathsf{LR}}) \ ) \tag{5.2}$$

Comme S'^t (resp. S'^p) est le sous-ordonnancement de S_k^{t*} (resp. S_k^{p*}) sur l (resp. $k-l$) machines générant la plus grande *taille* (resp. le plus grand *poids*), la *taille* moyenne par machine de S'^t est plus grande que la *taille* moyenne par machine de S_k^{t*} (resp. le *poids* moyen par machine de S'^p est plus grand que le *poids* moyen par machine de S_k^{p*}). On a donc :

$$\frac{|S_k^{t*}|}{k} \leq \frac{|S'^t|}{l} \Rightarrow |S_k^{t*}| \leq \frac{k}{l}|S'^t| \qquad (\text{resp. } \frac{w(S_k^{p*})}{k} \leq \frac{w(S'^p)}{k-l} \Rightarrow w(S_k^{p*}) \leq \left(\frac{k}{k-l}\right) w(S'^p) \ )$$

En combinant cette inégalité avec (5.2), on obtient :

$$|S_k^{t*}| \leq \frac{k}{l}|S_l^{\mathsf{GOL}}| \qquad (\text{resp. } w(S_k^{p*}) \leq \left(\frac{2k}{k-l-2}\right) w(S_{k-l}^{\mathsf{LR}}) \ ) \tag{5.3}$$

Comme $|S_l^{\mathsf{GOL}}| = |V_{i_2}(\mathsf{GOL}^l)|$ (resp. $w(S_{k-l}^{\mathsf{LR}}) = w(V_{i_2}(\mathsf{LR}^{k-l}))$), en appliquant le corollaire 4 à (5.3), on obtient :

$$|S_k^{t*}| \leq \frac{k}{l}|R_{i_2}(\mathsf{MTP}^k)| \qquad (\text{resp. } w(S_k^{p*}) \leq \left(\frac{2k}{k-l-2}\right) w(R_{i_2}(\mathsf{MTP}^k)) \ )$$

Par définition de $R_{i_2}(\mathsf{MTP}^k)$, l'algorithme MTP^k est donc $\left(\frac{k}{l}, \frac{2k}{k-l-2}\right)$ – compétitif. □

5.3 Bornes inférieures simultanées

Lorsque le système est composé d'une seule machine ($k = 1$), il n'existe pas d'algorithme (on-line ou off-line) dont les rapports de compétitivité pour la *taille* et le *poids* sont simultanément constants. En effet, considérons l'ensemble d'intervalles à ordonnancer (de manière on-line ou off-line) suivant : $\{\sigma_0, \ldots, \sigma_n\}$ avec $\sigma_0 = [0, K[$ et pour tout i, $1 \leq i \leq n$, $\sigma_i = [(i-1)\epsilon, i \cdot \epsilon[$. Si K est tel que $K \geq n \cdot \epsilon > 0$, un ordonnancement valide ne pourra pas ordonnancer simultanément σ_0 et des intervalles parmi $\{\sigma_1, \ldots, \sigma_n\}$. Or un algorithme qui rejette l'intervalle σ_0 pour ordonnancer des intervalles de $\{\sigma_1, \ldots, \sigma_n\}$ a un rapport de compétitivité pour le *poids* d'au moins :

$$\frac{w(\sigma_0)}{\sum_{1 \leq i \leq n} \sigma_i} = \frac{K}{n \cdot \epsilon} \qquad (\text{avec } \lim_{\epsilon \to 0} \frac{K}{n \cdot \epsilon} = \infty)$$

À l'inverse, un algorithme qui rejette tous les intervalles de $\{\sigma_1, \ldots, \sigma_n\}$ pour ordonnancer σ_0 a un rapport de compétitivité pour la *taille* de :

$$\frac{|\{\sigma_1, \ldots, \sigma_n\}|}{|\{\sigma_0\}|} = n \qquad (\text{avec } \lim_{n \to \infty} n = \infty)$$

Nous montrons maintenant que pour tout $k \geq 2$ (où k est le nombre de machines du système), il n'existe pas d'algorithme bicritère (on-line ou off-line) dont les rapports de compétitivité sont simultanément strictement inférieurs à 2 pour la *taille* et le *poids* (nous rappelons que lorsque $k = 1$, il n'existe pas d'algorithme dont les rapports de compétitivité pour la *taille* et le *poids* sont simultanément constants).

Théorème 30 *Pour tout $k \geq 2$, tout algorithme (on-line ou off-line) bicritère (ρ, μ) – compétitif pour l'ordonnancement d'intervalles sur k machines est tel que :*

- *Si $\rho \leq 2$, alors $\mu \geq 2$*
- *Si $\mu \leq 2$, alors $\rho \geq 2$.*

Preuve. Soit l'ensemble d'intervalles à ordonnancer (de manière on-line ou off-line) composé des intervalles suivant :

- k copies de l'intervalle $[0, K[$ et
- pour tout i, $1 \leq i \leq n$, k copies de l'intervalle $[(i-1)\epsilon, i \cdot \epsilon[$.

On pose K tel que $K \geq n \cdot \epsilon > 0$. Un ordonnancement valide ne pourra alors pas ordonnancer simultanément sur la même machine un intervalle $[0, K[$ et des intervalles parmi $\{[0, \epsilon[, [\epsilon, 2\epsilon[, \ldots, [(n-1)\epsilon, n \cdot \epsilon[\}$. On pose $n > k$.

- Si un algorithme A a un rapport de compétitivité pour la *taille* de $\rho \leq 2$, cela signifie qu'au moins la moitié des machines sont occupées par des intervalles de la forme $[(i-1)\epsilon, i \cdot \epsilon[$. A ne peut donc ordonnancer au plus que $\lfloor \frac{k}{2} \rfloor$ copies de l'intervalle $[0, K[$ (sur l'autre moitié des machines non occupées par des intervalles de la forme $[(i-1)\epsilon, i \cdot \epsilon[$). A a donc un rapport de compétitivité μ pour le *poids* d'au moins :

$$\begin{aligned} \mu &= \frac{k \cdot w([0, K[)}{\lfloor \frac{k}{2} \rfloor \cdot w([0, K[) + \lceil \frac{k}{2} \rceil \cdot \sum_{1 \leq i \leq n} w([(i-1)\epsilon, i \cdot \epsilon[)} \\ &= \frac{k \cdot K}{\lfloor \frac{k}{2} \rfloor \cdot K + \lceil \frac{k}{2} \rceil \cdot n \cdot \epsilon} \end{aligned}$$

  On a donc :

$$\lim_{\epsilon \to 0} \mu = \frac{k}{\lfloor \frac{k}{2} \rfloor} \geq 2$$

- Si un algorithme A a un rapport de compétitivité pour le *poids* de $\mu \leq 2$, cela signifie qu'au moins la moitié des machines sont occupées par une copie de l'intervalle $[0, K[$. A ne peut donc ordonnancer au plus que $n \cdot \lfloor \frac{k}{2} \rfloor$ intervalles de la forme $[(i-1)\epsilon, i \cdot \epsilon[$ (n intervalles sur chacune des machines non occupées par une copie de l'intervalle $[0, K[$). A a donc un rapport de compétitivité ρ pour la *taille* d'au moins :

$$\rho = \frac{n \cdot k}{n \cdot \lfloor \frac{k}{2} \rfloor + \lceil \frac{k}{2} \rceil}$$

  On a donc :

$$\lim_{n \to \infty} \rho = \frac{k}{\lfloor \frac{k}{2} \rfloor} \geq 2$$

□

Bilan du chapitre 5. Dans ce chapitre, nous avons proposé un algorithme on-line bicritère (appelé MTP^k) maximisant *simultanément* la *taille* et le *poids* pour l'ordonnancement d'intervalles présentés dans l'ordre croissant de leur bord gauche sur $k \geq 4$ machines. Nous avons montré que MTP^k est simultanément $\left(\frac{k}{l}\right)-$compétitif pour la *taille* et $\left(\frac{2k}{k-l-2}\right)-$compétitif pour le *poids* (avec l, $1 \leq l \leq k-3$ le nombre de machines allouées à la maximisation de la *taille*). En particulier, MTP^k est $(3,3)-$compétitif lorsque $l = \frac{k-2}{3}$ et k tend vers l'infini. Nous soulignons que la qualité de l'ordonnancement ainsi construit est particulièrement satisfaisante, puisque d'après le théorème 30, il n'existe pas d'algorithme bicritère dont le rapport de compétitivité est meilleur que le couple $(2,2)$.

Il est intéressant de remarquer que l'algorithme MTP^k est généralisable. En effet, nous avons présenté MTP^k dans une version utilisant comme sous-programmes les algorithmes GOL^l et LR^{k-l},

car ceux-ci sont les deux algorithmes on-line les plus performants pour leur critère respectif. Il est néanmoins possible d'utiliser d'autres algorithmes comme sous-programme de MTP^k. En effet, de manière générale, si on utilise un algorithme A^l (ρ – compétitif pour la *taille*) et un algorithme B^{k-l} (μ – compétitif pour le poids), on obtiendra un algorithme $(\rho \cdot \frac{k}{l}, \mu \cdot \frac{k}{k-l})$ – compétitif. De manière encore plus générale, l'algorithme MTP^k peut servir à maximiser tout couple de critère de profits quelconques, à partir du moment où l'on dispose de deux algorithmes on-line compétitifs pour chacun des critères. En particulier, si on utilise les algorithmes GOL^l et MP^{k-l} (voir section 4.2 pour une définition de l'algorithme MP) comme sous-programme de l'algorithme MTP^k, on obtient un algorithme simultanément $\left(\frac{k}{l}\right)$–compétitif pour la *taille* et $\left((2\alpha+3)\left(1+\frac{\beta}{\alpha-\beta}+\frac{1}{\alpha-\beta-1}\right)\left(\frac{k}{k-l}\right)\right)$– compétitif pour le *poids avec pénalité*. En effet, l'algorithme MP étant dédié à l'ordonnancement de tâches révélées dans un ordre quelconque, il peut bien sûr être utilisé pour le cas particulier de l'ordonnancement d'intervalles révélés dans l'ordre croissant de leur bord gauche.

Enfin, ajoutons que nous avons également généralisé les résultats du problème présenté dans ce chapitre au cas des intervalles *dégradables*. Un intervalle dégradable est défini par un triplet (l, r, q) que l'opérateur peut exécuter comme n'importe quel intervalle de la forme $[l, r'[$ tel que $q \leq r' \leq r$ (le profit que tire l'opérateur de cette exécution est alors $w([l, r'[) = r' - l$). L'interprétation intuitive du modèle d'ordonnancement d'intervalles dégradables est la suivante : il s'agit d'offrir à l'opérateur la possibilité d'ordonnancer une version *dégradée* (ou raccourcie) d'un intervalle soumis. Pour répondre à ce problème, nous avons proposé un algorithme on-line bicritère pour l'ordonnancement d'intervalles dégradables présentés dans l'ordre croissant de leur bord gauche sur $k \geq 4$ machines. Nous avons proposé un algorithme simultanément $\left(\frac{k}{l}\right)$ – compétitif pour la *taille* et $\left(\frac{4k}{k-l-2}\right)$ – compétitif pour le *poids*. Il s'agit à nouveau d'un travail réalisé en collaboration avec Fabien Baille (voir [10, 14]). Nous ne présentons pas ici ce résultat car les techniques utilisées pour sa résolution sont très proches de celles décrites dans ce chapitre. Son intégration aurait donc surcharger inutilement ce chapitre.

Synthèse et perspectives de la seconde partie

Ce court chapitre fait la synthèse des résultats obtenus dans la seconde partie de la thèse et propose un certain nombre de perspectives.

Synthèse

Nous récapitulons notre contribution pour les critères *taille*, *poids* et *poids proportionnel* que nous avons obtenus : sous la forme de deux algorithmes on-line monocritères pour l'ordonnancement de tâches et sous la forme d'un algorithme on-line bicritère pour l'ordonnancement d'intervalles. Le tableau 5.2 récapitule pour chaque algorithme proposé le modèle de tâches à ordonnancer traité, l'ordre de révélation on-line requis et le rapport de compétitivité obtenu. Nous rappelons que :

- $k \leq 1$ est le nombre de machines du système,
- λ le nombre de longueurs de tâches différentes parmi les tâches révélées,
- γ le rapport entre la longueur de la plus grande tâche et la longueur de la plus petite tâche parmi les tâches révélées,
- $\beta \geq 0$ la constante de pénalité représentant la perte de profit pour l'opérateur associée à l'interruption d'une tâche déjà ordonnancée et $\alpha \geq 1$ une constante à choisir en fonction de β pour minimiser le rapport de compétitivité de l'algorithme MP, et
- l, $1 \leq l \leq k-3$, le nombre de machines dédiées à la maximisation du critère *taille*.

L'analyse de l'algorithme MT (pour Maximisation de la Taille) se trouve dans la section 4.1, celle de l'algorithme MP (pour Maximisation du Poids) se trouve dans la section 4.2, et enfin, celle de l'algorithme MTP^k (pour Maximisation de la Taille et du Poids) se trouve dans la section 5.2. Nous insistons sur le fait que les rapports de compétitivité que nous avons obtenus dans la seconde partie de la thèse sont constants (avec la restriction suivante pour l'algorithme MT : la valeur de λ ou de β doit être bornée par une constante).

Nous soulignons également que les résultats de la section 4.3 montrent que pour chacun des problèmes traités par les algorithmes MT et MP, il n'existe pas d'algorithme on-line optimal, et ce quel que soit le nombre de machines considérées. De plus, les résultats de la section 5.3 montrent qu'il n'existe pas d'algorithme bicritère (on-line ou off-line) dont les rapports de compétitivité sont simultanément strictement inférieurs à 2 pour la *taille* et le *poids*.

	Algorithmes d'ordonnancement on-line sur k machines		
	MT	MP	MTP^k
Modèle	tâches	tâches	intervalles
Ordre de révélation	quelconque	quelconque	bords gauches croissants
Rapport(s) de compétitivité	$4\min(\lambda, \lfloor\log_2(\gamma)\rfloor + 1)$ pour la *taille*	$(2\alpha+3)\left(1+\frac{\beta}{\alpha-\beta}+\frac{1}{\alpha-\beta-1}\right)$ pour le *poids avec pénalité*	$\frac{k}{l}$ pour la *taille* et $\frac{2k}{k-l-2}$ pour le *poids*
Exemple avec des valeurs fixées pour chacun des paramètres	$28-$ compétitif pour la *taille* avec λ quelconque et $\gamma = 128$	$14.63-$ compétitif pour le *poids avec pénalité* avec $\beta = 0.2$ et $\alpha \approx 2.88$	$2-$ compétitif pour la *taille* et $5-$ compétitif pour le *poids* avec $k = 20$ et $l = 10$

TAB. 5.2 – Tableau récapitulatif des algorithmes d'ordonnancement on-line proposés

Perspectives

Comparaison des deux modèles envisagés. Le modèle d'ordonnancement que nous proposons pour notre étude bicritère est beaucoup plus faible que celui que nous proposons pour l'étude des critères séparés (monocritère). En effet, l'algorithme MTP^k ne s'applique qu'aux intervalles (nous rappelons qu'un intervalle est une tâche $\Gamma = (l, r, p)$ avec $p = r - l$) révélés dans l'ordre croissant de leur bord gauche, alors que les algorithmes MT et MP s'appliquent à des tâches quelconques (c'est-à-dire des tâches $\Gamma = (l, r, p)$ avec $p \leq r - l$) révélées dans un ordre quelconque. Néanmoins, même si nous n'avons pour l'instant pas de résultats bicritères pour le modèle général des tâches révélées dans un ordre quelconque, l'idée principale de l'algorithme bicritère MTP^k (simuler deux algorithmes on-line monocritères sur un sous-ensemble de machines) nous semble générale et nous espérons pouvoir l'appliquer au modèle des tâches révélées dans un ordre quelconque. La principale difficulté de cette généralisation est la suivante : une tâche (contrairement à un intervalle) peut être exécutée comme plusieurs intervalles temporels (et même une infinité). Cela signifie que les deux algorithmes monocritères simulés peuvent ordonnancer la même tâche comme deux intervalles temporels différents. Or, pour construire à chaque étape un ordonnancement valide, un algorithme

bicritère devra choisir quelle version de la tâche ordonnancer. Dans un contexte on-line (c'est-à-dire sans connaissance du futur), le choix d'une version plutôt que l'autre pourra éventuellement se révéler mauvaise par la suite. C'est cette difficulté du choix entre deux alternatives d'une tâche que nous n'arrivons pas à maîtriser pour l'instant.

Raffinement des algorithmes proposés. Les algorithmes on-line monocritères MT et MP proposés dans le chapitre 4 sont particulièrement simples. En effet, ils consistent à remplacer sur une machine quelconque une (ou plusieurs) tâche(s) déjà ordonnancée(s) si un critère de remplacement est satisfait. Néanmoins, si nous devions implémenter ces algorithmes, il serait intéressant de les raffiner en choisissant à chaque étape la meilleure machine possible. Pour MT, la meilleure machine signifie une machine libre s'il en existe une, sinon, la machine sur laquelle la condition de remplacement est satisfaite et où la tâche interrompue est la plus longue possible. Pour MP, la meilleure machine signifie à nouveau une machine libre s'il en existe une, sinon, la machine sur laquelle la condition de remplacement est satisfaite et où la somme des tâches interrompues est la plus faible. La raison pour laquelle nous n'avons pas inclus ces raffinements dans les descriptions des algorithmes MT et MP est qu'ils ne modifient pas le pire cas. Autrement dit, ils ne nous permettent pas d'améliorer l'analyse des rapports de compétitivité. Nous avons donc décrit MT et MP de la manière la plus concise possible.

Nous insistons également sur le fait que le choix des constantes de remplacement (égale à 2 pour MT et dépendant de la pénalité β pour MP) ne sont pas nécessairement les meilleures constantes possibles dans l'absolu, mais les meilleures constantes pour obtenir le meilleur rapport de compétitivité avec *notre* analyse. De plus, si on s'intéresse au rapport de compétitivité moyen, il semble que ces constantes ne sont pas appropriées. En effet, nous avons effectué des simulations sur des instances aléatoires d'intervalles révélés dans un ordre quelconque et obtenu les résultats suivants. Aussi bien pour le critère *taille* que *poids*, les meilleurs rapports de compétitivité ont été obtenus pour une constante de remplacement égale à 1, avec des rapports de compétitivité effectifs situés entre 1 et 1.5 (donc très loin des rapports de compétitivité au pire cas analysés dans le chapitre 4). Cela signifie qu'en pratique, remplacer un intervalle par un autre si celui-ci est plus petit (resp. grand) semble la meilleure stratégie pour maximiser la *taille* (resp. *poids*). Nos simulations se sont limités aux cas des intervalles car dans ce cas, le calcul de la solution optimale (et donc du rapport de compétitivité effectif) peut être fait en temps polynomial, alors que le problème est NP-complet lorsqu'il s'agit de tâches. Nous soulignons également que lorsque les constantes de remplacement sont égales à 1, le rapport de compétitivité dans le pire cas n'est pas constant (aussi bien pour la *taille* que pour le *poids*).

Amélioration des bornes inférieures et supérieures pour les rapports de compétitivité. Une perspective à court terme est l'amélioration des bornes inférieures et supérieures pour les rapports de compétitivité, notamment pour la *taille* et le *poids avec pénalité* pour l'ordonnancement sur k machines de tâches révélées dans un ordre quelconque. En effet, il reste pour l'instant un écart important entre le rapport de compétitivité de MT (MT est par exemple $28-$ compétitif pour la *taille* lorsque $\gamma = 128$) et la borne inférieure pour le rapport de compétitivité pour la *taille* de tout algorithme sur $k \geq 2$ machines (égale à $\min\left(\frac{4}{3}, \frac{2k}{2k-1}\right)$). De même, l'écart est également important entre le rapport de compétitivité de MP (MP est $13.32-$ compétitif pour le *poids*) et la borne inférieure pour le rapport de compétitivité pour le *poids* de tout algorithme sur $k \geq 2$ machines (égale à $\min\left(\frac{5}{4}, \frac{2k}{2k-1}\right)$). Au-delà de la réduction de ces écarts, le problème de savoir s'il est possible d'obtenir un rapport de compétitivité constant pour la *taille* (donc indépendant de λ et γ) reste ouvert.

Quelques propositions d'élargissement du modèle. Il serait intéressant d'aller plus loin dans la dissociation de l'axe temporel de l'ordonnancement des tâches et de l'axe temporel de leur révélation (ce qui a été esquissé ici en ne considérant pas uniquement les séquence on-line de tâches révélées par bords gauches croissants). Plus précisément, un modèle intéressant à étudier serait le suivant. Une tâche est maintenant définie par un quadruplet $\Gamma = (l, r, p, d)$, où l représente le bord gauche de la tâche, r son bord droit, $p \leq r - l$ la taille de l'intervalle qu'occupera Γ sur une machine entre l et r si Γ est ordonnancée et d la date d'échéance avant laquelle le client veut pouvoir bénéficier de sa réservation. Cela signifie que la taille p ne correspond plus à une durée mais à un espace situé entre l et r. Une requête $\Gamma = (l, r, p, d)$ s'interprète alors comme la demande de réservation sur une des k machines du système d'un espace de taille p situé entre l et r jusqu'à la date d. L'intérêt de cette proposition est qu'elle modélise la notion de *retrait* d'une tâche (puisqu'à la date d, l'espace occupé par Γ sur une des machines est à nouveau libre), notion que nous n'avons envisagée pour l'instant que dans la première partie de la thèse, lorsque des membres du groupe courant veulent quitter la structure. La figure 5.1 illustre ce modèle en proposant un ordonnancement possible des requêtes suivantes : au temps $t = 0$ est révélée la requête $\Gamma_1 = (1, 3, 2, 10)$, au temps $t = 5$ est révélée la requête $\Gamma_2 = (0, 1, 1, 15)$ puis, au temps $t = 10$ est révélée la requête $\Gamma_3 = (1, 5, 3, 20)$. Nous précisons que dans l'exemple d'exécution de la figure 5.1, aucune tâche n'est interrompue : chaque tâche ordonnancée est simplement retirée au moment de sa date d'échéance.

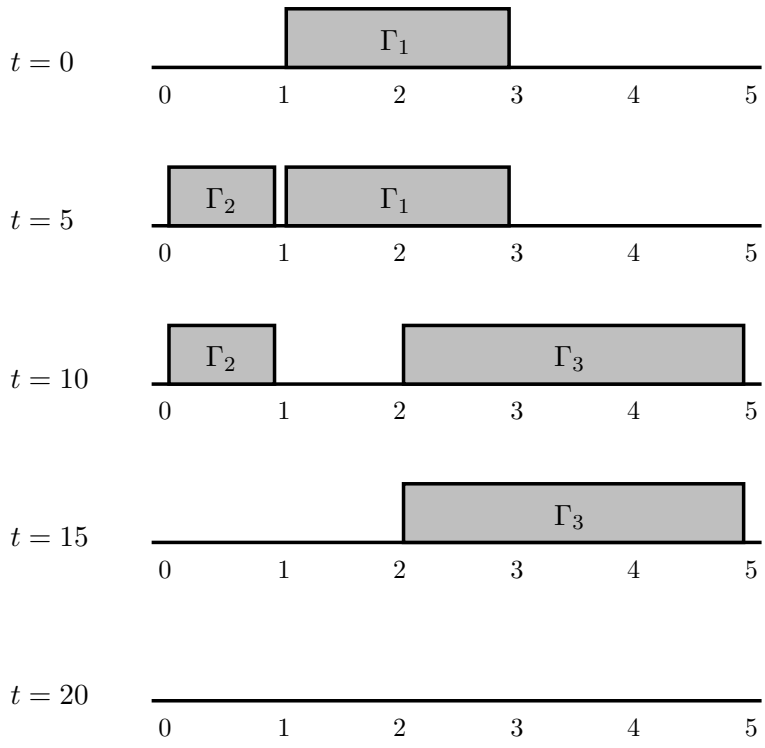


FIG. 5.1 – Illustration de l'ordonnancement courant au temps t

Il serait également intéressant de prendre en compte le phénomène de famine qui peut se produire pour certains clients. En effet, considérons la situation suivante. Un client soumet une tâche

à l'opérateur, mais celui-ci la rejette. La situation la plus probable est que le client soumette ultérieurement une nouvelle tâche de même durée. Mais, celle-ci peut tout à fait être à nouveau rejetée (ou acceptée puis interrompue), et ce indéfiniment. Il serait intéressant d'atténuer ce phénomène en proposant un système de priorité croissante aux clients dont les tâches sont rejetées ou interrompues. Cet indice de priorité par client pourrait par exemple être proportionnel à la somme des poids des tâches dont le client n'a pas bénéficié.

Le défi posé par ces modèles plus complexes consiste à proposer des algorithmes prenant en compte ces nouveaux paramètres tout en restant compétitifs (c'est-à-dire avec des rapports de compétitivité les plus faibles possibles).

Conclusion générale

Dans cette thèse, nous avons résolu un certain nombre de problèmes liés à la réservation de ressources dans un réseau, en proposant des algorithmes on-line pour la construction de structures de connexion dans des graphes et pour la construction d'ordonnancements. Nous avons évalué analytiquement la performance de chacun des algorithmes proposés en fonction d'un (ou parfois plusieurs) critère(s), pour obtenir des garanties au pire cas.

Au-delà de la différence de nature entre les objets étudiés (graphes versus ordonnancements) et au-delà des différences entre les problèmes on-line envisagés (minimisation des distances dans des arbres versus maximisation de la taille et du poids des ordonnancements), la question récurrente suivante s'est posée : malgré l'absence totale de connaissance du futur, peut-on garantir à l'utilisateur/opérateur une certaine qualité de la solution construite ? Nous avons montré que dans un certain nombre de cas, cette "maîtrise du futur" est possible. Bien sûr, tous les résultats que nous avons obtenus ne sont pas pleinement satisfaisants, notamment du fait de l'importance de la valeur des rapports de compétitivité obtenus (néanmoins constants pour la plupart). Si le défi premier était d'obtenir des rapports de compétitivité constants, un travail de raffinement de ces constantes reste à faire (par le raffinement des algorithmes proposés et/ou de leur analyse).

Pour chacun des problèmes traités, lorsque cela s'est révélé nécessaire, nous avons proposé un principe de *remise en cause maîtrisée* de la solution courante. Pour la première partie (sur les groupes dynamiques dans un graphe), cela s'est traduit par l'autorisation de remettre en cause la solution courante en maîtrisant le nombre maximum d'étapes critiques et de changements élémentaires. Dans la seconde partie (sur les ordonnancements on-line), cela s'est traduit par l'autorisation d'interruption de tâches compensée (dans certains cas) par la mise en place d'un principe de pénalité pour chaque interruption. Ces remises en cause se sont en effet révélées nécessaires pour l'obtention d'une qualité suffisante des objets construits. En effet, chaque fois que nous l'avons pu, nous avons donné des bornes inférieures universelles (valables pour tout algorithme) prouvant cette nécessité de relâchement d'un certain nombre de contraintes.

Une perspective à long terme concerne la démarche utilisée pour l'évaluation d'un algorithme on-line. Nous aimerions en effet pouvoir nous affranchir de l'étude du pire cas, et faire évoluer nos travaux vers des résultats plus fins, qui rendraient compte analytiquement du *comportement moyen* d'un algorithme. Bien sûr, il s'agit là d'un vrai défi, puisqu'il est plus facile d'évaluer analytiquement un algorithme dans le pire cas que dans le cas moyen. Il serait également intéressant d'étudier chaque problème incrémental (resp. décrémental) indépendamment de la non connaissance du futur. En effet, si le modèle (intermédiaire entre les modèles off-line et on-line classiques) où les données sont *disponibles* au fur et à mesure, mais *connues* à l'avance, est très peu utilisé, il présente pourtant un intérêt certain, aussi bien d'un point de vue pratique que théorique (notamment pour dissocier les difficultés liées à l'incrémentalité des solutions construites de celles liées à la non connaissance du futur).

Annexe 1 : preuve du théorème 11

Pour prouver le théorème 11, nous décrivons d'abord le graphe G_p et la séquence particulière d'ajouts de l'adversaire adaptatif que nous allons utiliser.

Description du graphe G_p. Pour tout $p \geq 2$, on définit le graphe $G_p = (V_p, E_p, w_p)$ de la manière suivante (voir figure 5.2 pour une illustration du graphe G_2). Pour toute arête $e \in E_p$, $w_p(e) = 1$ (on ne précisera donc plus w_p par la suite). G_p est le cycle $C = (V_C, E_C)$ de longueur 2^p augmenté des éléments suivants : chaque sommet $r \in V_C$ est la racine d'une étoile $S_r = (V_{S_r}, E_{S_r})$ à 2^p feuilles. C'est-à-dire que l'on a :

- Pour tout sommet $r \in V_C$, chaque sommet $u \in V_{S_r} \backslash \{r\}$ est relié à r par une arête.
- Pour tout sommet $r \in V_C$, on a $|V_{S_r}| = 2^p + 1$.

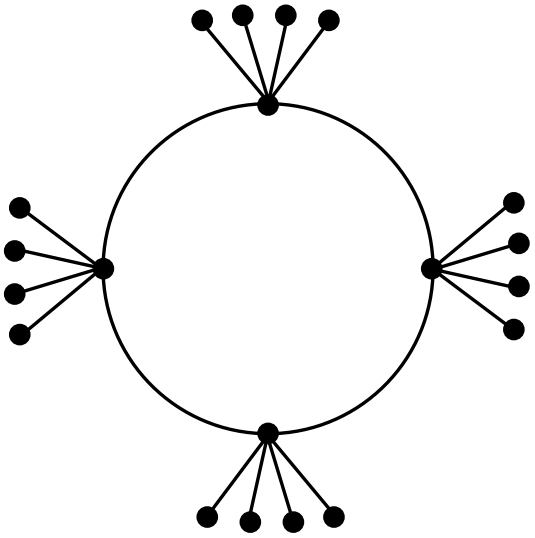


FIG. 5.2 – Le graphe G_2

Définition de la séquence d'ajouts $M_0 \subset \cdots \subset M_i$. Soit A un algorithme *quelconque* respectant les contraintes *arbre* et *emboîtement*. Nous utilisons un adversaire adaptatif pour définir la séquence d'ajouts dans le graphe G_p défini ci-dessus, avec p tel que pour toute étape i, on a $i \leq 2^{p+1} + p$ (où i est le nombre de sommets ajoutés). Pour tout i ($0 \leq i \leq 2^{p+1} + p$), soit T_i l'arbre

élagué couvrant M_i construit par l'algorithme A à l'étape i. La séquence d'ajouts est alors définie de la manière suivante.

- À l'étape $i = 0$. Soit $u_0 \in V_C$ un sommet quelconque du cycle C inclus dans G_p. On a :

  $$M_0 = \{u_0\}$$

  $T_0 = (V_{T_0}, E_{T_0})$ est donc composé uniquement du sommet u_0 (car T_0 couvrant M_0 est élagué). L'adversaire adaptatif ajoute maintenant le sommet $u_1 \in V_C$ tel que $d_{G_p}(u_0, u_1) = 2^{p-1}$ pour obtenir M_1 (u_1 est donc le sommet du cycle C diamétralement opposé au sommet u_0).
- À l'étape $i = 1$, on a :

  $$M_1 = \{u_0, u_1\}$$

  Soit $T_1 = (V_{T_1}, E_{T_1})$ l'arbre couvrant M_1 construit par l'algorithme A à l'étape 1. Comme u_0 et u_1 appartiennent au cycle C, T_1 est un chemin d'extrémités u_0 et u_1 (car T_1 couvrant M_1 est élagué).
- À l'étape i, $2 \leq i \leq p$, on a :

  $$M_i = \bigcup_{0 \leq j \leq i} \{u_j\} \subseteq V_C$$

  Soit $T_{i-1} = (V_{T_{i-1}}, E_{T_{i-1}})$ l'arbre couvrant M_{i-1} construit par l'algorithme A à l'étape $i-1$. Comme tous les sommets de M_{i-1} appartiennent au cycle C, T_{i-1} est un chemin. Soient u_{i-1} et u_j ($0 \leq j \leq i-2$) les deux sommets extrémités du chemin T_{i-1}. L'adversaire adaptatif ajoute maintenant le sommet $u_i \in V_C \backslash V_{T_{i-1}}$ tel que $d_{G_p}(u_{i-1}, u_i) = 2^{p-i}$ pour obtenir M_i (u_i est donc le sommet à égale distance des sommets u_{i-1} et u_j dans la partie du cycle C non couverte par T_{i-1}). Pour obtenir T_i, un algorithme A quelconque (respectant les contraintes *arbre* et *emboîtement* doit donc incrémenter T_{i-1} du chemin joignant les sommets u_i et u_{i-1} ou bien du chemin joignant les sommets u_i et u_j.
- À l'étape $i = p$, l'arbre T_p couvrant M_p obtenu est composé de toutes les arêtes du cycle sauf une (car T_p est un arbre). Soit e cette arête et soient r_1 et r_2 les deux sommets connectés par e (un de ces deux sommets est u_p, le dernier sommet ajouté par l'adversaire. Sans perte de généralité, on pose $r_1 = u_p$). Soient $S_{r_1} = (V_{S_{r_1}}, E_{S_{r_1}})$ et $S_{r_2} = (V_{S_{r_2}}, E_{S_{r_2}})$ les deux étoiles de G_p de racine respective r_1 et r_2. L'adversaire ajoute maintenant (un par un et dans n'importe quel ordre) tous les sommets de $\left(V_{S_{r_1}} \backslash \{r_1\}\right) \cup \left(V_{S_{r_2}} \backslash \{r_2\}\right)$ pour obtenir $M_{2^{p+1}+p}$.
- À l'étape $i = 2^{p+1} + p$, on a :

  $$M_{2^{p+1}+p} = \bigcup_{0 \leq j \leq p} \{u_j\} \cup \left(V_{S_{r_1}} \backslash \{r_1\}\right) \cup \left(V_{S_{r_2}} \backslash \{r_2\}\right)$$

La figure 5.3 illustre l'adversaire adaptatif avec $p = 3$.

Tout algorithme a un rapport de compétitivité au moins linéaire en i. Pour démontrer le théorème 11, nous avons besoin des deux lemmes préliminaires suivants.

Lemme 26 *Soient G_p le graphe et u_i, r_1 les sommets définis ci-dessus. On a alors :*

$$\sum_{i=0}^{p} d_{G_p}(r_1, u_i) \leq 2^{p+1} + 2^{p-1} - 2$$

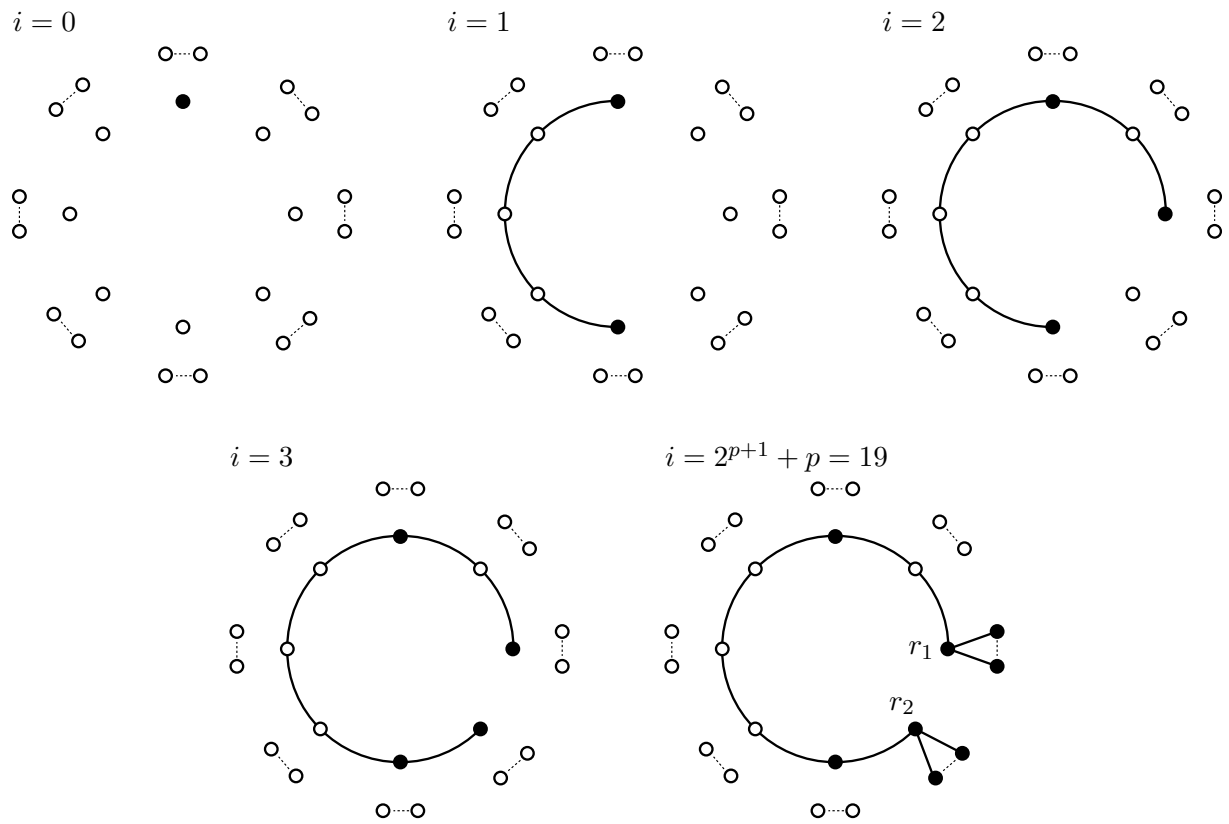


FIG. 5.3 – Illustration de la séquence d'ajouts sur le graphe G_3

Preuve.

$$\begin{aligned}
\sum_{i=0}^{p} d_{G_p}(r_1, u_i) &= d_{G_p}(u_p, u_0) + \sum_{i=1}^{p} d_{G_p}(u_p, u_i) \;\leq\; d_{G_p}(u_0, u_1) + \sum_{i=1}^{p} d_{G_p}(u_p, u_i) \\
&\quad (\text{car } d_{G_p}(r_1, u_0) = d_{G_p}(u_p, u_0) \leq 2^{p-1} = d_{G_p}(u_0, u_1)) \\
&\leq d_{G_p}(u_0, u_1) + \sum_{i=1}^{p} \sum_{j=i}^{p} d_{G_p}(u_j, u_{j-1}) \\
&\quad (\text{d'après l'inégalité triangulaire}) \\
&\leq 2^{p-1} + \sum_{i=1}^{p} \sum_{j=i}^{p} 2^{p-j} \;\leq\; 2^{p-1} + \sum_{i=1}^{p} 2.2^{p-i} \;=\; 2^{p+1} + 2^{p-1} - 2 \\
&\quad (\text{car } d_{G_p}(u_{j-1}, u_j) = 2^{p-j})
\end{aligned}$$

□

Lemme 27 *Soient G_p le graphe et u_i, r_2 les sommets définis ci-dessus. On a alors :*

$$\sum_{i=0}^{p} d_{G_p}(r_2, u_i) \leq 2^{p+1} + 2^{p-1} + p - 1$$

Preuve.

$$\begin{aligned}
\sum_{i=0}^{p} d_{G_p}(r_2, u_i) &\leq \sum_{i=0}^{p} d_{G_p}(r_2, r_1) + \sum_{i=0}^{p} d_{G_p}(r_1, u_i) \\
&\quad \text{(d'après l'inégalité triangulaire)} \\
&\leq \sum_{i=0}^{p} d_{G_p}(r_2, r_1) + 2^{p+1} + 2^{p-1} - 2 \\
&\quad \text{(d'après le lemme 26)} \\
&\leq 2^{p+1} + 2^{p-1} + p - 1 \\
&\quad (\text{car } d_{G_p}(r_2, r_1) = 1)
\end{aligned}$$

□

Nous rappelons maintenant l'énoncé du théorème 11.

Théoreme 11 *Pour tout algorithme* A *respectant les contraintes* on-line, arbre *et* emboîtement, *pour tout i suffisamment grand, il existe un graphe G_p et une séquence de i ajouts tels que* A *a un rapport de compétitivité pour la somme des distances en $\Omega(i)$, c'est-à-dire que l'on a :*

$$\frac{C_{T_i}(M_i)}{C_{T_i^*}(M_i)} \in \Omega(i)$$

Preuve. On pose $p \geq 7$. Nous allons dans un premier temps majorer $C_{T_i^*}(M_i)$ et minorer $C_{T_i}(M_i)$. Pour cela, on partitionne M_i en trois sous-ensembles :

- $M_i^1 = V_{S_{r_1}} \backslash \{r_1\}$ ($|M_i^1| = 2^p$),
- $M_i^2 = V_{S_{r_2}} \backslash \{r_2\}$ ($|M_i^2| = 2^p$) et
- $M_i^3 = M_i \cap V_C$ ($|M_i^3| = p + 1$).

Afin de simplifier les notations, pour tous groupes U et V disjoints, on pose :

$$C_{G_p}(U \leftrightarrow V) = 2 \sum_{u \in U} \sum_{v \in V} d_{G_p}(u, v)$$

Minoration de $C_{T_i}(M_i)$. Pour chaque sommet de M_i^1, les 2^p sommets de M_i^2 sont à une distance $2^p + 1$ dans T_i, et vice versa. On a donc :

$$C_{T_i}(M_i) \;\geq\; C_{T_i}(M_i^1 \leftrightarrow M_i^2) \;=\; 2 \cdot 2^p \cdot 2^p \cdot (2^p + 1) \;\geq\; 2^{3p+1} \tag{5.4}$$

Majoration de $C_{T_i^*}(M_i)$. Pour cela, on commence par majorer $C_{G_p}(M_i)$ On a :

$$C_{G_p}(M_i) = C_{G_p}(M_i^1) + C_{G_p}(M_i^2) + C_{G_p}(M_i^3) + C_{G_p}(M_i^1 \leftrightarrow M_i^2) + C_{G_p}(M_i^1 \leftrightarrow M_i^3) + C_{G_p}(M_i^2 \leftrightarrow M_i^3)$$

On majore chaque terme de $C_{G_p}(M_i)$:

- Majoration de $C_{G_p}(M_i^1)$ et $C_{G_p}(M_i^2)$. Pour tout $u \in M_i^1$ (resp. $u \in M_i^2$), pour tout $v \in M_i^1 \backslash \{u\}$ (resp. $v \in M_i^2 \backslash \{u\}$), on a $d_{G_p}(u, v) = 2$. Comme $|M_i^1| = 2^p$ (resp. $|M_i^2| = 2^p$), on en déduit :

$$C_{G_p}(M_i^1) = 2^p(2^p - 1) \cdot 2 \leq 2^{2p+1} \qquad (\text{resp. } C_{G_p}(M_i^2) = 2^p(2^p - 1) \cdot 2 \leq 2^{2p+1})$$

– Majoration de $C_{G_p}(M_i^3)$. Pour tout $u \in M_i^3$, pour tout $v \in M_i^3 \backslash \{u\}$, on a $d_{G_p}(u,v) \leq 2^{p-1}$ (car C est un cycle de longueur 2^p). Comme $|M_i^3| = p+1$, on en déduit :

$$C_{G_p}(M_i^3) \leq (p+1)p \cdot 2^{p-1} = (p^2+p) \cdot 2^{p-1} \leq (2^{p+1}+2^{p+1}) \cdot 2^{p-1} \leq 2^{2p+1}$$

– Majoration de $C_{G_p}(M_i^1 \leftrightarrow M_i^2)$. Pour tout $u \in M_i^1$, pour tout $v \in M_i^2$, on a $d_{G_p}(u,v) = 3$. Comme $|M_i^1| = |M_i^2| = 2^p$, on en déduit :

$$C_{G_p}(M_i^1 \leftrightarrow M_i^2) = 2 \cdot 2^p \cdot 2^p \cdot 3 \leq 2^{2p+3}$$

– Majoration de $C_{G_p}(M_i^1 \leftrightarrow M_i^3)$. On a :

$$\begin{aligned} C_{G_p}(M_i^1 \leftrightarrow M_i^3) &= 2 \sum_{u \in M_i^1} \sum_{v \in M_i^3} d_{G_p}(u,v) \leq 2 \sum_{u \in M_i^1} \left(\sum_{v \in M_i^3} d_{G_p}(u,r_1) + \sum_{v \in M_i^3} d_{G_p}(r_1,v) \right) \\ &\quad \text{(d'après l'inégalité triangulaire)} \\ &= 2 \cdot 2^p \left(p+1 + \sum_{v \in M_i^3} d_{G_p}(r_1,v) \right) \leq 2^{p+1}(p+2^{p+1}+2^{p-1}-1) \\ &\quad \text{(d'après le lemme 26 et car } p \geq 7) \\ &\leq 2^{2p+3} \end{aligned}$$

– Majoration de $C_{G_p}(M_i^2 \leftrightarrow M_i^3)$. On a :

$$\begin{aligned} C_{G_p}(M_i^2 \leftrightarrow M_i^3) &= 2 \sum_{u \in M_i^2} \sum_{v \in M_i^3} d_{G_p}(u,v) \leq 2 \sum_{u \in M_i^2} \left(\sum_{v \in M_i^3} d_{G_p}(u,r_2) + \sum_{v \in M_i^3} d_{G_p}(r_2,v) \right) \\ &\quad \text{(d'après l'inégalité triangulaire)} \\ &\leq 2 \cdot 2^p \left(p+1 + \sum_{v \in M_i^3} d_{G_p}(r_2,v) \right) \leq 2^{p+1}(2^{p+1}+2^{p-1}+2p) \\ &\quad \text{(d'après le lemme 27 et car } p \geq 7) \\ &\leq 2^{2p+3} \end{aligned}$$

On en déduit la majoration de $C_{G_p}(M_i)$ suivante :

$$C_{G_p}(M_i) \leq 2^{2p+1}+2^{2p+1}+2^{2p+1}+2^{2p+3}+2^{2p+3}+2^{2p+3} \leq 2^{2p+5}$$

De plus, d'après [46], il existe un arbre T_i^{off} couvrant M_i tel que $C_{T_i^{\text{off}}}(M_i) \leq 2C_G(M_i)$. Comme T_i^* est un arbre couvrant M_i optimal pour la somme des distances, on en déduit :

$$C_{T_i^*}(M_i) \leq C_{T_i^{\text{off}}}(M_i) \leq 2C_{G_p}(M_i) \leq 2^{2p+6} \tag{5.5}$$

D'après (5.4) et (5.5), on obtient :

$$\frac{C_{T_i}(M_i)}{C_{T_i^*}(M_i)} \geq \frac{2^{3p+1}}{2^{2p+6}} = \frac{2^{p+2}}{128}$$

Comme $i = |M_i| - 1 = 2^{p+1} + p \leq 2^{p+2}$, on obtient :

$$\frac{C_{T_i}(M_i)}{C_{G_p}(M_i)} \geq \frac{i}{128} \in \Omega(i)$$

□

Annexe 2 : l'algorithme AR

Nous proposons ici un algorithme on-line très simple qui permet de construire un arbre couvrant le groupe courant pour des séquences on-line autorisant ajouts et retraits mêlés. Nous prouvons que cette algorithme est simulanément $(2D_{\max})$-compétitif pour le diamètre et $\left(\left(2+\frac{2}{m_i-1}\right)D_{\max}\right)-$ compétitif pour la somme des distances, où $D_{\max}$ est le diamètre maximum calculé à partir de tous les sommets des groupes successifs. Nous définissons l'algorithme AR (pour Ajouts et Retraits) de la manière suivante.

Algorithme Ajouts et Retraits – AR

1 Soit $G=(V,E,w)$ un graphe.
2 Soit $M_0 \subseteq V$ le groupe initial.
3 Construire un arbre T_0 de plus courts chemins enraciné en r_0
4 ($r_0 \in M_0$ est quelconque)
5 Soit T_i l'arbre couvrant M_i à l'étape i.
6 Soit rq_{i+1} la $i+1^{\text{ième}}$ requête on-line
7 Si $rq_{i+1}=(a,u)$ est une requête d'ajout
8 Alors construire T_{i+1} couvrant $M_{i+1}=M_i \cup \{u\}$ en incrémentant T_i
9 d'un plus court chemin de la u à r_0 sans créer de cycle
10 Sinon $rq_{i+1}=(r,u)$ est une requête de retrait et $M_{i+1}=M_i \backslash \{u\}$.
11 Construire T_{i+1} en élaguant (si nécessaire) T_i pour obtenir
12 l'arbre élagué couvrant $M_{i+1} \cup \{r_0\}$.

Remarques : L'algorithme AR respecte bien la contrainte *emboîtement*, puisqu'à chaque étape d'ajout, on incrémente l'arbre courant d'au plus un chemin (voir lignes 8 et 9 de l'algorithme AR), et à chaque étape de retrait, on élague (si nécessaire) l'arbre courant pour obtenir un nouvel arbre entièrement contenu dans le précédent. Nous soulignons également que la contrainte *arbre* est respectée, puisqu'à chaque étape (d'ajout ou de retrait), l'algorithme maintient l'absence de cycle et la connexité de la structure (voir lignes 8, 9, 11 et 12 de l'algorithme AR).

Construire T_0 un arbre de plus courts chemins enraciné en r_0 (voir lignes 3 et 4 de l'algorithme AR) peut être fait en temps polynomial, en utilisant l'algorithme de Dijkstra. De plus, incrémenter l'arbre ne nécessite que l'ajout d'un plus court chemin. Cela peut être fait en temps polynomial, à nouveau en utilisant l'algorithme de Dijkstra. Enfin, chaque étape de retrait nécessite l'élagage des branches mortes de l'arbre courant, ce qui peut être à nouveau fait en temps polynomial, en

utilisant un parcours quelconque.

Analyse de l'algorithme AR. Nous prouvons maintenant que l'algorithme AR est $(2D_{\max})$-compétitif pour tout graphe dont les arêtes ont un poids d'au moins 1.

Théorème 31 *Soit $G = (V, E, w)$ un graphe tel que $\min\{w(e) : e \in E\} = 1$. Soient i le nombre de requêtes on-line et $D_{\max} = \max\left\{d_G(u, v) : u, v \in \bigcup_{0 \leq j \leq i} M_j\right\}$. Pour toute séquence de requêtes on-line $M_0, \ldots, M_i$, l'algorithme AR est $2D_{\max}$-compétitif, c'est-à-dire que l'on a :*

$$D_{T_i}(M_i) \leq 2D_{\max} \cdot D_G(M_i)$$

Preuve. Par définition du diamètre, on a :

$$\begin{aligned}
D_{T_i}(M_i) &= \max\{d_{T_i}(u, v) : u, v \in M_i\} \\
&\leq \max\{d_{T_i}(u, r_0) : u \in M_i\} + \max\{d_{T_i}(r_0, v) : v \in M_i\} \\
&\quad \text{(d'après l'inégalité triangulaire)} \\
&\leq \max\{d_G(u, r_0) : u \in M_i\} + \max\{d_G(r_0, v) : v \in M_i\} \\
&\quad \text{(car } T_i \text{ est un arbre de plus courts chemins enraciné en } r_0 \text{ couvrant } M_i \cup \{r_0\}) \\
&\leq 2D_{\max} \\
&\quad \text{(car } r_0 \in M_0 \text{, et d'après la définition de } D_{\max}) \\
&\leq 2D_{\max} \cdot D_G(M_i) \\
&\quad \text{(comme } \min\{w(e) : e \in E\} = 1 \text{, on a } D_G(M_i) \geq 1)
\end{aligned}$$

□

Nous montrons maintenant que l'algorithme AR est $\left(\left(2 + \frac{2}{m_i - 1}\right) D_{\max}\right)$ − compétitif pour la somme des distances pour tout graphe dont les arêtes ont un poids d'au moins 1.

Théorème 32 *Soit $G = (V, E, w)$ un graphe tel que $\min\{w(e) : e \in E\} = 1$. Soient i le nombre de requêtes on-line, M_i le $i^{ème}$ groupe et $D_{\max} = \max\left\{d_G(u, v) : u, v \in \bigcup_{0 \leq j \leq i} M_j\right\}$. Pour toute séquence de requêtes on-line $M_0, \ldots, M_i$ telle que $m_j \geq 2$ $(0 \leq j \leq i)$, l'algorithme AR est $\left(\left(2 + \frac{2}{m_i - 1}\right) D_{\max}\right)$ − compétitif, c'est-à-dire que l'on a :*

$$C_{T_i}(M_i) \leq \left(2 + \frac{2}{m_i - 1}\right) D_{\max} \cdot C_G(M_i)$$

Preuve. Par définition de $C_G(M_i)$, on a :

$$\begin{aligned}
& C_G(M_i) = \sum_{u \in M_i} \sum_{v \in M_i \setminus \{u\}} d_G(u, v) \\
\Rightarrow\ & C_G(M_i) \geq m_i(m_i - 1) \\
& \quad \text{(car } \forall u, v \in V \text{ tel que } u \neq v,\ d_G(u, v) \geq 1) \\
\Rightarrow\ & C_G(M_i) + m_i = m_i^2 \\
\Rightarrow\ & C_G(M_i) + \frac{C_G(M_i)}{m_i - 1} = m_i^2
\end{aligned}$$

On en déduit :

$$\left(1 + \frac{1}{m_i - 1}\right) C_G(M_i) \;\geq\; m_i^2 \tag{5.6}$$

Or, on a :

$$\begin{aligned}
C_{T_i}(M_i) &= \sum_{u \in M_i} \sum_{u \in M_i} d_{T_i}(u, v) \\
&\leq \sum_{u \in M_i} \sum_{u \in M_i} (d_{T_i}(u, r) + d_{T_i}(r, v)) \\
&\quad \text{(d'après l'inégalité triangulaire)} \\
&\leq \sum_{u \in M_i} \sum_{u \in M_i} d_G(u, r) + \sum_{u \in M_i} \sum_{u \in M_i} d_G(r, v) \\
&\quad \text{(car } T_i \text{ est un arbre de plus courts chemins enraciné en } r\text{)} \\
&= 2m_i \sum_{u \in M_i} d_G(u, r) \\
&\leq 2m_i \sum_{u \in M_i} D_{\max} \\
&\quad \text{(d'après la définition de } D_{\max}\text{)} \\
&= 2m_i^2 \cdot D_{\max} \\
&\leq \left(2 + \frac{2}{m_i - 1}\right) D_{\max}.C_G(M_i) \\
&\quad \text{(d'après (5.6))}
\end{aligned}$$

□

Le théorème 31 (resp. 32) montre que si le groupe courant évolue dans une zone du graphe dont le diamètre $D_{\max}$ est borné par une constante, alors l'algorithme AR a un rapport de compétitivité constant pour le diamètre (resp. la somme des distances). Bien sûr, ce résultat n'est intéressant que dans certains cas très particuliers, lorsque la valeur de $D_{\max}$ est bornée.

Annexe 3 : les algorithmes GOL^k et LR^k

Nous présentons ici les deux algorithmes on-lines monocritères pour l'ordonnancement d'intervalles révélés par bords gauches croissants, proposés par Faigle et Nawijn (dans [23]) et Bar-Noy et al. (dans [15]). Nous rappelons que ces deux algorithmes sont utilisés comme sous-programmes l'algorithme bicritère MTP^k dans le chapitre 8.

Présentation de l'algorithme GOL^k. L'algorithme GOL^k (pour Greedy On-line Algorithm sur k machines) provient de [23].

Algorithme GOL^k

Soit k le nombre de machines du système.
Soit $\sigma = [l, r[$ le nouvel intervalle révélé
Phase d'interruption :
    S'il existe k intervalles déjà ordonnancés intersectant l
    ALORS soit $\sigma_{\max} = [l_{\max}, r_{\max}[$ l'intervalle dont le bord droit est maximum
        SI $r_{max} \geq r$
        ALORS interrompre $\sigma_{\max}$
        SINON n'interrompre aucun intervalle
    SINON n'interrompre aucun intervalle (il existe une machine libre).
Phase d'ordonnancement :
    S'il existe une machine libre
    ALORS ordonnancer σ sur cette machine
    SINON rejeter σ.

Présentation de l'algorithme LR^k. L'algorithme LR^k (pour Left Right sur $k \geq 3$ machines) est une adaptation de l'algorithme LR de [15].

Algorithme LR^k

Soit $k \geq 3$ le nombre de machines parallèles identiques du système.
Soit σ le nouvel intervalle révélé
Soit F_t l'ensemble des intervalles déjà ordonnancés contenant le réel (l'instant) t.
Phase d'interruption :
    Si $|F_r| < k$
    Alors n'interrompre aucun intervalle
    Sinon on a $|F_r| = k$.
    − Trier les $k+1$ intervalles de $F_r \cup \{\sigma\}$ par ordre croissant de leur bord gauche.
      Si plusieurs intervalles ont le même bord gauche, les trier par ordre décroissant
      de leur bord droit et soit L l'ensemble des $\lceil \frac{k}{2} \rceil$ premiers intervalles.
    − Trier les $k+1$ intervalles de $F_r \cup \{\sigma\}$ par ordre décroissant de leur
      bord droit et soit R l'ensemble des $\lfloor \frac{k}{2} \rfloor$ premiers intervalles.
        Si $\sigma \in L \cup R$
        Alors interrompre un intervalle $\sigma' \in F_r \backslash (L \cup R)$ quelconque
        Sinon n'interrompre aucun intervalle.
Phase d'ordonnancement :
    Si $|F_r| < k$
    Alors ordonnancer σ sur n'importe quelle machine libre
    Si $|F_r| = k$ et $\sigma \in L \cup R$
    Alors ordonnancer σ sur la machine où σ' a été interrompu
    Si $|F_r| = k$ et $\sigma \notin L \cup R$
    Alors Rejeter σ.

Bibliographie

[1] F. Adelstein, G. G. Richard III, and L. Scwiebert, *Distributed multicast tree generation with dynamic group membership*, Computer communications, 26 (2003), pp. 1105–1128.

[2] R. Adler and Y. Azar, *Beating the logarithmic lower bound : Randomized preemptive disjoint paths and call control algorithms*, Journal of Scheduling, 6 (2003), pp. 113–129.

[3] N. Alon and Y. Azar, *On-line steiner trees in the euclidean plane*, in Symposium on Computational Geometry, 1992, pp. 337–343.

[4] C. Ambuhl and M. Mastrolilli, *On-line scheduling to minimize max flow time : an optimal preemptive algorithm*, Operations Research Letters, 33 (2005), pp. 597–602.

[5] E. Angelelli, M. G. Speranza, and Z. Tuza, *Semi on-line scheduling on two parallel processors with upper bound on the items*, Algorithmica, 37 (2003), pp. 243–262.

[6] ——, *New bounds and algorithms for on-line scheduling : two identical processors, known sum and upper bound on the tasks*, Discrete Mathematics and Theoretical Computer Science, 8 (2006), pp. 1–16.

[7] E. Arkin and B. Silverberg, *Scheduling jobs with fixed start and end times*, Discrete Applied Mathematics, 18 (1987), pp. 1–8.

[8] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti Spaccamela, and M. Protasi, *Complexity and approximation*, Springer, 1999.

[9] B. Awerbuch, Y. Azar, and Y. Bartal, *On-line generalized steiner problem*, in ACM-SIAM Symposium on Discrete Algorithms (SODA), 1996, pp. 68–74.

[10] F. Baille, *Algorithmes d'approximation pour des problèmes d'ordonnancement bicritères : application à un problème d'accès au réseau*, in Manuscrit de thèse, 2005.

[11] F. Baille, E. Bampis, and C. Laforest, *A note on bicriteria schedules with optimal approximation ratios*, Parallel Processing Letters, 14 (2004), pp. 315–323.

[12] F. Baille, E. Bampis, C. Laforest, and N. Thibault, *Algorithmes d'ordonnancements bicritères en-lignes (résumé)*, in AlgoTel, 2005, pp. 71–74.

[13] ——, *On-line bicriteria interval scheduling*, in Euro-Par, LNCS 3648, Springer, 2005, pp. 312–322.

[14] ——, *On-line simultaneous maximization of the size and the weight for degradable intervals schedules*, in COCOON, LNCS 3595, Springer, 2005, pp. 308–317.

[15] A. Bar-Noy, R. Canetti, S. Kutten, Y. Mansour, and B. Schieber, *Bandwidth allocation with preemption*, SIAM Journal on Computing, 28 (1999), pp. 1806–1828.

[16] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber, *Approximating the throughput of multiple machines in real-time scheduling*, SIAM Journal on Computing, 31 (2001), pp. 331–352.

[17] B. Bilò, M. Flammini, and L. Moscardelli, *Pareto approximations for the bicriteria scheduling problem*, 18th International Parallel and Distributed Processing Symposium (IPDPS), (2004).

[18] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*, Cambridge University press, 1998.

[19] M. C. Carlisle and E. Lloyd, *On the k-coloring of intervals*, Discrete Applied Mathemetics, 59 (1995), pp. 225–235.

[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction à l'algorithmique, 2^e édition*, DUNOD, 2002.

[21] T. Erlebach and F. C. Spieksma, *Interval selection : Applications, algorithms, and lower bounds*, Journal of Algorithms, 46 (2003), pp. 27–53.

[22] U. Faigle, W. Kern, and G. Turan, *On the performance of on-line algorithms for particular problems*, Acta Cybernetica, 9 (1989), pp. 107–119.

[23] U. Faigle and W. Nawijn, *Note on scheduling intervals on-line*, Discrete Applied Mathematics, 58 (1994), pp. 13–17.

[24] A. Feldmann, M. Y. Kao, J. Sgall, and S. H. Teng, *Optimal online scheduling of parallel jobs with dependencies*, Journal of Combinatorial Optimization, 1 (1998), pp. 393–411.

[25] A. Fiat and G. J. Woeginger, *Online algorithmes : The state of the art*, LNCS no. 1442, Springer, 1998.

[26] S. P. Y. Fung, C. F. Y. L., and H. Shen, *Online scheduling of unit jobs with bounded importance ratio*, International Journal of Foundations of Computer Science, 16 (2005), pp. 581–598.

[27] J. Garay, I. S. Gopal, S. Kutten, Y. Mansour, and M. Yung, *Efficient on-line call control algorithms*, Journal of Algorithms, 23 (1997), pp. 180–194.

[28] J. Garay, J. Naor, B. Yener, and P. Zhao, *On-line admission control and packet scheduling with interleaving*, in Proceedings of INFOCOM, 2002, pp. 94–103.

[29] M. Garey and D. Johnson, *Computers and intractability*, in Freeman and compagny, 1979.

[30] A. Goel and K. Munagala, *Extending greedy multicast routing to delay sensitive applications*, Algorithmica, 33 (2002), pp. 335–352.

[31] S. A. Goldman, J. Parwatikar, and S. Suri, *On-line scheduling with hard deadlines*, in 5th Workshop on Algorithms and Data Structures (WADS), 1997, pp. 258–271.

[32] M. H. Goldwasser, *Patience is a virtue : the effect of slack on competitiveness for admission control*, in 10th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1999, pp. 396–405.

[33] M. H. Goldwasser and B. Kerbikov, *Admission control with immediate notification*, Journal of Scheduling, 6 (2003), pp. 269–285.

[34] R. L. Graham, *Bounds on certain multiprocessing anomalies*, Bell System Technical Journal, 45 (1966), pp. 1563–1581.

[35] ——, *Bounds on certain multiprocessing timing anomalies*, SIAM Journal of Applied Mathematics, 17 (1969), pp. 416–429.

[36] D. Hochbaum, *Approximation algorithms for NP-hard problems*, PWS publishing compagny, 1997.

[37] J. Hromkovic, *Algorithmics for Hard Problems - Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*, Texts in Theoretical Computer Science, An EATCS Series, 2nd edition, Corr. 2nd printing, Springer, 2004.

[38] T. C. Hu, *Optimum communication spanning trees*, SIAM Journal on Computing, 3 (1974), pp. 188–195.

[39] M. Imase and B. Waxman, *Dynamic steiner tree problem*, SIAM J. Discr. Math., 4 (1991), pp. 369–384.

[40] Y. Jiang and Y. He, *Preemptive online algorihms for scheduling with machine cost*, Acta Informatica, 41 (2005), pp. 315–340.

[41] D. S. Johnson, J. K. Lenstra, and A. Rinnooy Kan, *The complexity of the network design problem*, Networks, 3 (1978), pp. 279–285.

[42] B. Kalyanasunaram and K. Pruhs, *Speed is as powerful as clairvoyance*, Journal of ACM, 47 (2000), pp. 617–643.

[43] J. H. Kim and C. K. Y., *Scheduling broadcasts with deadlines*, Theorical Computer Science, 325 (2004), pp. 479–488.

[44] R. Klasing, C. Laforest, J. Peters, and N. Thibault, *Constructing incremental sequences in graphs*, Algorithmic Operations Research, (2006).

[45] C. F. Y. L. and S. P. Y. Fung, *Improved competitive algorithms for online scheduling with partial job values*, Theorical Computer Science, 325 (2004), pp. 467–478.

[46] C. Laforest, *A good balance between weight and distances for multipoint trees*, in International Conference On Principles Of DIstributed Systems, 2002, pp. 195–204.

[47] L. Layuan and L. Chunlin, *A qos multicast routing protocol for dynamic group topology*, in Euro-Par, LNCS 2790, Springer, 2003, pp. 980–988.

[48] R. J. Lipton and A. Tomkins, *Online interval scheduling*, in 5th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1994, pp. 1806–1828.

[49] Z. Lui and T. Cheng, *Scheduling with job release dates, delivery times and preemption penalties*, Information Process. Lett., 82 (2002), pp. 107–111.

[50] ——, *Minimizing total completion time subject to job release dates and preemption penalties*, Journal of Scheduling, 7 (2004), pp. 313–327.

[51] J. Naor, A. Rosen, and G. Scalosub, *Online time-constrained scheduling in linear networks*, in Proceedings of INFOCOM, vol. 2, 2005, pp. 855–865.

[52] E. Naroska and U. Schwiegelshohn, *On an on-line scheduling problem for parallel jobs*, Information Processing Letters, 81 (2002), pp. 297–304.

[53] C. A. Philips, C. Stein, E. Torng, and J. Wein, *Optimal time-critical scheduling via resource augmentation*, in 29th ACM Symposium on Theory of Computing, 1997, pp. 140–149.

[54] R. Sriram, G. Manimaran, and C. Murthy, *A rearrangeable algorithm for the construction of delay-constrained dynamic multicast trees*, IEEE-ACM Transactions on Networking, 7 (1999), pp. 514–529.

[55] N. Thibault, *Médian-ajout : un algorithme incrémental pour le maintien d'un arbre de connexion*, in Mémoire de DEA Informatique de l'université d'Évry Val d'Essonne, 2003.

[56] N. Thibault and C. Laforest, *Algorithme incrémental pour le maintien d'un arbre de connexion*, in RenPar'15, 2003, pp. 27–34.

[57] ——, *Deux méthodes incrémentales pour le maintien d'un arbre de connexion*, in AlgoTel, 2004, pp. 63–67.

[58] ——, *An optimal online strategy to increment connection trees*, in IEEE Workshop on Adaptive Wireless Networks, (in conjuction with GLOBECOM), 2004.

[59] ——, *On-line time-constrained scheduling problem for the size on k machines*, in ISPAN, IEEE Computer Society, 2005, pp. 20–24.

[60] ——, *Ajouts et retraits dans un arbre de connexion.*, in AlgoTel, 2006, pp. 33–36.

[61] ——, *Algorithmes d'ordonnancement on-line sur k machines : maximisation du poids et de la taille (résumé)*, in ROADEF, 2006.

[62] ——, *An optimal rebuilding strategy for a decremental tree problem*, in SIROCCO, LNCS 4056, Springer, 2006, pp. 157–170.

[63] ——, *An optimal rebuilding strategy for an incremental tree problem*, in Journal of Interconnexion Networks, accepté en 2006.

[64] V. V. Vazirani, *Approximation algorithms*, 1st edition, Corr. 2nd printing, Springer, 2002.

[65] B. Waxman, *Routing of multipoint connections*, IEEE Journal on Selected Areas in Communications, 6 (1988), pp. 1617–1622.

[66] J. Westbrook and D. Yan, *Linear bounds for on-line steiner problems*, Information Processing Letters, (1995), pp. 59–63.

[67] G. Woeginger, *On-line scheduling of jobs with fixed start and end times*, Theoritical Computer Science, 130 (1994), pp. 5–16.

[68] R. Wong, *Worst-case analysis of network design problem heuristics*, SIAM J. Algebraic Discrete Math., 1 (1980), pp. 51–63.

[69] B. Y. Wu, K. M. Chao, and C. Y. Tang, *Approximation algorithms for the shortest total path length spanning tree problem*, Discrete Applied Mathematics, 105 (2000), pp. 273–289.

[70] D. Ye and G. Zhang, *On-line scheduling of parallel jobs*, in 11th Colloquium on Structural Information and Communication Complexity (SIROCCO), LNCS 3104, Springer, 2004, pp. 279–290.

Index

Remerciements

Je remercie l'ensemble des membres du jury, Marc Demange, Vassilis Zissimopoulos, Pierre Fraigniaud, Ralf Klasing et Evripidis Bampis pour leur intérêt et leur disponibilité. Enfin, je tiens à remercier particulièrement Christian Laforest pour avoir été à tous points de vue un directeur de thèse exceptionnel.