Agrégation de ressources avec contrainte de distance : applications aux plateformes de grande échelle.

Hubert Larchevêque

N° d'ordre : 4067

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ DE BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Par **Hubert Larchevêque**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Agrégation de ressources avec contrainte de distance : applications aux plateformes de grande échelle.

Soutenue le : 27 Septembre 2010

Après avis des rapporteurs :
Christian Laforest
Christian Lavault

Devant la commission d'examen composée de :

Laurent Viennot .	Président du Jury
Christian Laforest	Rapporteur
Christian Lavault	Rapporteur
Anne Benoit	Examinateur
Olivier Beaumont	Directeur de Thèse
Nicolas Bonichon	Directeur de Thèse
Philippe Duchon .	Directeur de Thèse

– 2010 –

Remerciements

Mes premiers remerciements iront à l'attention de mes directeurs de thèse : Olivier Beaumont, Nicolas Bonichon et Philippe Duchon (l'ordre ne compte pas !). J'ai eu beaucoup de chance de pouvoir bénéficier d'une telle complémentarité, à la fois scientifique mais aussi humaine. Ils ont chacun su m'apporter un soutien différent, des ouvertures et des idées différentes, et des façons diverses et variées de démontrer l'inexactitude des preuves que je leur proposais. Je doute que nombre d'étudiants ait la chance de bénéficier d'un trio aussi efficace dans leur encadrement. Je les remercie pour cette chance qu'ils m'ont offert.

Je souhaite ensuite remercier sincèrement Christian Laforest et Christian Lavault, pour leur relecture attentive de mon manuscrit. Je souhaite également adresser mes remerciements à Anne Benoit pour avoir accepté de faire partie de mon jury de thèse, à Laurent Viennot pour avoir accepté d'en assurer la présidence, et globalement à l'ensemble des membres du jury de ma thèse pour leurs questions qui ont révélé l'intérêt qu'ils ont porté à mon travail.

Je tiens à remercier l'ensemble des membres de l'équipe Cépage, très disponible et avec qui il est très facile d'avoir des discussions scientifiques intéressantes, et, en particulier, Lionel Eyraud-Dubois, avec qui j'ai travaillé au cours de ma thèse.

Je remercie toutes les personnes avec qui j'ai pu travailler dans le cadre de mes enseignements à l'Université de Bordeaux, Nicolas le premier, mais aussi, entre autres, Frédérique Carrère, Patrick Chervet et Arthur Sarthou avec qui j'ai pris beaucoup de plaisir à travailler.

Enfin je remercie toutes les personnes qui ont croisé ma route au LaBRI au cours de ces trois ans, mes anciens co-bureaux exilés dans le bâtiment d'à côté et toute leur équipe, en particulier Damien qui s'est attardé un peu plus que les autres avec moi. Allez Robin, c'est à toi maintenant ! Je remercie également les membres de l'AFoDIB, Anne-Laure en particulier, mais aussi Cathy Roubineau et toutes les personnes qui m'ont aidé d'une façon ou d'une autre durant cette thèse. Un clin d'oeil à Alice Rivière, qui a toujours eu la patience de me rééxpliquer les mêmes démarches, sans que je parvienne jamais à les assimiler.

Merci à ma soeur, mon frère, mes parents et plus généralement ma famille, de ne pas trop chercher à savoir ce que je fais mais de savoir que je le fais et de penser que je le fais bien. Merci à mes parents d'être venus et d'avoir cohabité, merci à mes grands parents d'avoir fait le déplacement également.

Par ailleurs, j'ai une pensée émue pour toutes les personnes avec lesquelles j'ai vécu au cours de ces trois années à l'Auberge du Chat Kipu, qui ont eu à supporter mes théories fumeuses et mes expériences foireuses : Johan, Pierre-Laurent, Gregoire, Céline, Julie, Goulven, Laetitia, Baturan, Romain, Julien, Emmanuelle, Florian, Stephen, Leïla (oui tout ça !), mais également tous les couchsurfers et squatteurs de passage qui auront contribué à animer notre habitat durant ces trois années.

Enfin, pour finir en musique, je serai infiniment reconnaissant à la Grasse Bande de m'avoir fait découvrir un monde parallèle qui permet de laisser s'exprimer des idées qui n'ont pas vocation à faire avancer la recherche ! Elle me fut nécessaire, salutaire et très enrichissante, et ses membres hauts en couleurs sont indéniablement ancrés dans mon coeur. Au sein de cet univers de la fanfare, je tiens en particulier à remercier Adam, qui m'y a amené et dont j'ai suivi la voie sinueuse tracée à travers ces deux mondes, scientifiquement, musicalement et en trottinette (zeugmatiquement, même !), et Cécile, qui accepte ma folie et en fait une force pour aller de l'avant.

Agrégation de ressources avec contrainte de distance : applications aux plateformes de grande échelle.

Résumé :
Durant cette thèse, nous avons introduit les problèmes de Bin Covering avec Contrainte de Distance (BCCD) et de Bin Packing avec Contrainte de Distance (BPCD), qui trouvent leur application dans les réseaux de grande échelle, tel Internet.

L'étude de ces problèmes que nous effectuons dans des espaces métriques quelconques montre qu'il est impossible de travailler dans un tel cadre sans avoir recours à de l'augmentation de ressources, un procédé qui permet d'élaborer des algorithmes construisant des solutions moins contraintes que la solution optimale à laquelle elles sont comparées. En plus de résultats d'approximation intéressants, nous prouvons la difficulté de ces problèmes si ce procédé n'est pas utilisé.

Par ailleurs, de nombreux outils ont pour objectif de plonger les grands réseaux qui nous intéressent dans des espaces métriques bien décrits. Nous avons alors étudié nos problèmes dans les espaces métriques générés par certains de ces outils, comme Vivaldi et Sequoia.

Mots clés :
Bin Packing, Bin Covering, algorithmes d'approximation, augmentation de ressources, algorithmes distribués, structure de données probabiliste, réseaux de grande échelle, plongement d'Internet.

Resource clustering with distance constraint: applications to large scale platforms.

Abstract :
During this Ph. D. we introduced two problems, respectively Bin Covering under Distance Constraints (BCCD in French) and Bin Packing under Distance Constraints (BPCD in French). Both problems find their applications in large scale networks, like the Internet.

The study of those problems in general metric spaces revealed that it is impossible to work in such a general framework with using resource augmentation, that allows solutions built by one of our algorithms to be less constrained than the optimal solution they are compared to. We prove both interesting approximation results, and that without using resource augmentation those problem are hard to tackle.

In this work we were also interested in tools aiming at embedding large scale networks aimed by our applications into well described metric spaces. Thus we studied our problems in some specific metric spaces, generated by some of those tools, namely Vivaldi and Sequoia.

Keywords :
Bin Packing, Bin Covering, approximation algorithms, resource augmentation, distributed algorithms, probabilistic data structures, large scale networks, embedding of the Internet. plongement d'Internet.

Plan

Chapitre 1

Introduction

1.1 Contexte et Motivations

Les travaux présentés dans ce manuscrit sont liés à l'émergence depuis quelques années d'un nouveau type de plateformes de calcul comme BOINC [And04], Folding@home [1] ou WCG [2]. Ces plateformes largement distribuées sont caractérisées par leur capacité de calcul totale élevée, l'hétérogénéité des performances de leurs ressources (aussi bien des ressources de communication que de calcul) et le dynamisme de leur topologie, dû aux fréquents arrivées et départs des participants. Jusqu'à maintenant, toutes les applications utilisant ces plateformes (Seti@home [ACK+02], Folding@home,...) sont constituées d'un grand nombre de tâches indépendantes, et toutes les données nécessaires au traitement d'une tâche doivent être stockées localement sur le noeud traitant la tâche. Les seuls échanges de données ont lieu entre le noeud maître (serveur central distribuant les tâches) et les esclaves (participants), ce qui restreint fortement le champ d'applications de ce type de plateformes.

Deux sortes d'applications fonctionnent de cette façon. La première contient celles qui, comme Seti@home, sont constituées d'un gros volume de données découpable en petits morceaux qui peuvent être traités indépendamment sur les noeuds participants. La seconde correspond aux simulations de type Monte-Carlo. Dans ce cas, tous les esclaves travaillent sur les mêmes données, exception faite d'un petit ensemble de paramètres qui dirigent la simulation. C'est par exemple le cas de Folding@home.

Notre idée première est d'étendre ce dernier ensemble d'applications. Plus précisément, nous nous intéressons au cas où le volume de données nécessaire au traitement d'une tâche est trop grand pour être stocké sur un seul noeud. Cette situation peut vraisemblablement apparaître sur ce type de plateformes de grande échelle s'appuyant sur l'agrégation de ressources très hétérogènes. Dans ce cas, les besoins en stockage et en ressources de calcul doivent être répartis sur un ensemble de noeuds qui va collaborer pour traiter une même tâche. Les noeuds présents au sein d'un même groupe de travail doivent alors avoir une capacité agrégée (en terme de mémoire, de puissance de calcul, etc.) suffisante, plus grande qu'une certaine valeur seuil, et on souhaite qu'ils soient proches les uns des autres de façon à éviter des latences importantes durant les communications au sein d'un même groupe de travail.

[1] http ://folding.stanford.edu/
[2] http ://www.worldcommunitygrid.org

Pour travailler d'une manière plus théorique dans ce contexte, nous introduisons le problème de *Bin Covering avec Contrainte de Distance* (BCCD), inspiré de Bin Covering classique [AJKL84]. Nous décrivons ce problème dans le Chapitre 2.

Dans ce manuscrit nous nous intéressons également à de nombreux problèmes classiques de placement de serveurs et de réplicats dans les réseaux, dont de nombreuses variantes font partie de problèmes d'optimisation classiques (cf. [PTW01, KPR00, STA97, CW99, KPR00, KLL09, Chu06, KM07, RS97] pour des études de problèmes de ce type). Ces problèmes considèrent que l'ensemble des utilisateurs d'un réseau souhaite accéder à un service particulier. L'objectif est alors de distinguer un ensemble de fournisseurs de services qui sont susceptibles de mettre à disposition une quantité suffisante de ressources pour pouvoir répondre aux demandes des clients de façon à assurer une qualité de service répondant à certaines exigences, comme le temps de réponse, le temps de transmission des éventuelles données liées au service considéré, etc. Par exemple, s'il s'agit d'un service de vidéo à la demande, chaque serveur doit avoir une bande passante suffisante pour gérer un certain nombre de connexions, suffisamment de serveurs doivent être disponibles simultanément pour qu'un grand nombre de clients puissent utiliser le service sans perte d'efficacité (liée au nombre de requêtes des clients). Il peut également être souhaitable que les serveurs soient suffisamment bien répartis dans le réseau, soit pour minimiser les temps de latence client-serveur, soit pour obtenir une plus grande tolérance aux pannes. Toujours dans le cadre de l'utilisation de plateformes à grande échelle, et notamment dans de grands réseaux de type Internet, il peut être intéressant, et surtout réaliste, d'ajouter à ce genre de problèmes des contraintes fortes d'hétérogénéité sur les capacités (bande passante, capacités de stockage, etc.) des serveurs et sur les demandes des clients.

Enfin nos travaux ont aussi été motivés par un autre problème qui se rapproche d'un problème présent en astrophysique et étudié dans [LMY98], où Lupton *et al.* visent à construire une carte bidimensionnelle du ciel composée de quasars et de galaxies. Pour récupérer les informations nécessaires sur ces galaxies, ils procèdent à de multiples captures de données avec un télescope, chacune englobant les informations sur une portion circulaire du ciel visible à travers le télescope. Ils introduisent alors le problème de "Couverture Euclidienne par des Disques, avec Capacité" (ECCD), dont l'objectif est de couvrir un ensemble de points dans un plan euclidien avec un nombre minimal de disques de diamètre fixé, sans violer la capacité maximale des disques (correspondant au nombre maximal de galaxies pour lesquelles les données peuvent être récupérées en une seule capture). Des heuristiques pour ce problème ont été proposées dans [LMY98], et ont été validées par des simulations.

Nous introduisons également un nouveau problème théorique pour travailler sur ces problématiques, que nous nommons le *Bin Packing avec Contrainte de Distance* (BPCD), inspiré de Bin Packing classique [EGCGJ97]. Nous présentons également ce problème dans le Chapitre 2.

Les domaines d'applications que visent les problèmes abordés dans ce manuscrit sont généralement liés à des réseaux de grande échelle, contenant de grands nombres de noeuds qui peuvent communiquer les uns avec les autres selon certaines règles. Dans ce genre de réseaux, il est souvent impossible de centraliser toute l'information concernant sa topologie, à cause du dynamisme fort intrinsèque à ce type de réseaux, couplé à la très grande taille de ceux-ci, ce qui limite l'utilisation d'algorithmes centralisés. Il est alors naturel d'essayer d'élaborer des stratégies qui fonctionnent dans des environnements distribués. Ainsi, en plus de résultats théoriques, souvent centralisés, nous essayons à plusieurs reprises de proposer des solutions distribuées à nos problèmes.

Dans un premier temps, nous étudions ces problèmes en modélisant les réseaux par des espaces

métriques quelconques (*i.e.* des graphes valués). Les résultats que nous obtenons montrent qu'il est impossible de travailler dans un tel cadre sans avoir recours à de l'augmentation de ressources. L'augmentation de ressources est un procédé qui peut être utilisé pour travailler sur des problèmes d'optimisation. Ce procédé permet d'élaborer des algorithmes construisant des solutions à un problème donné en le dotant de plus de puissance (de ressources), ou en lui imposant moins de contraintes que la solution optimale à laquelle il est comparé. Ce procédé nous permet d'obtenir des résultats théoriques d'approximation intéressants, mais également de prouver la difficulté des problèmes auxquels nous nous intéressons, en mettant en exergue le fait que sans ce procédé il est impossible de trouver des solutions convenables (garanties) en un temps raisonnable (polynomial).

Nous nous sommes ensuite intéressés à un autre moyen de contourner la difficulté des problèmes théoriques posés : plutôt que de considérer des espaces métriques généraux, nous avons utilisé une modélisation plus réaliste des réseaux auxquels nous nous intéressons. Il est connu que les réseaux à grande échelle intéressants pour notre étude ont une structure bien particulière. De nombreux outils ont pour objectif de plonger ce type de grands réseaux, Internet notamment, dans des espaces métriques bien décrits, dans lesquels il peut être plus simple de travailler. C'est le cas en particulier de Sequoia [RMK+09, ABK+07], qui plonge Internet dans un arbre dont les feuilles sont les noeuds du réseau, et de Vivaldi [DCKM04, CDK+04], qui plonge un réseau dans une généralisation du plan euclidien. Nous nous sommes donc intéressés à l'étude de nos problèmes dans ce type d'espaces métriques, plus simples et semblant correspondre plus exactement aux réseaux qui nous intéressent.

1.2 Quelques définitions d'ordre général

1.2.1 NP-Difficulté-Facteur d'approximation

Nous rappellons dans ce paragraphe les notions de NP-complétude, NP-difficulté et de facteurs d'approximation que nous utilisons dans tout le reste de ce manuscrit.

Un problème de décision est un problème dont la solution est "vrai" ou "faux". La classe de problèmes P est composée de l'ensemble des problèmes de décision décidables par un algorithme déterministe en temps polynomial [GJ+79]. La classe de problèmes NP est composée de l'ensemble des problèmes de décision décidables par un algorithme non déterministe en temps polynomial.

L'assertion $P \subseteq NP$ est vraie, par contre l'inclusion $NP \subseteq P$, qui impliquerait que $P = NP$ est à ce jour une des conjectures les plus importantes dans le domaine des mathématiques et de l'informatique (en pratique, en fait, la conjecture porte plutôt sur l'inégalité $P \neq NP$). Que le lecteur soit rassuré, nous ne nous y attaquons pas dans ce manuscrit. De manière générale, il est reconnu que si un problème de décision est connu pour être dans NP mais pas connu comme étant dans P, alors à moins d'une avancée révolutionnaire dans le monde de la complexité on peut considérer qu'il n'existe pas d'algorithme s'exécutant en temps polynomial permettant de le décider.

La classe de problèmes NP-complets est composée des problèmes "les plus durs" de NP, dans le sens où s'il existe un algorithme s'exécutant en temps polynomial permettant de décider de façon déterministe un problème NP-complet, alors tous les problèmes de la classe NP peuvent également être décidés en temps polynomial. Dans l'autre sens, si on prouve qu'il n'existe pas d'algorithme déterministe s'exécutant en temps polynomial permettant de décider un problème

de NP, alors c'est le cas pour tous les problèmes NP-complets. De même, si un problème NP-complet est décidable de façon déterministe en temps polynomial, alors $P = NP$.

Un des problèmes de décision fondamentaux en théorie de la complexité est le problème de logique booléenne SAT utilisé par Cook [Coo71] dans ses travaux de référence concernant la théorie de la complexité :

Définition 1.1 (*SAT*)

Etant donnés un ensemble $U = \{u_1, \ldots, u_n\}$ de variables booléennes , et un ensemble $C = \{c_1, \ldots, c_m\}$ de clauses composées de littéraux de U (pour $u \in U$, u et $\overline{u}$ sont des littéraux de U), trouver une affectation de valeurs aux variables de U de façon à ce que chaque clause possède au moins un littéral de valeur "vrai".

Par exemple le couple (U, C) tel que $U = \{u_1, u_2\}$, et $C = \{\{u_1, \overline{u_2}\}, \{\overline{u_1}, u_2\}\}$ est une instance de SAT pour laquelle la réponse est "vrai". Une affectation de valeurs assurant que chaque clause possède au moins un littéral "vrai" est $u_1 = u_2 =$ "*vrai*".

Ce problème a en effet été le premier problème prouvé comme appartenant à la classe de problème NP-complet par Cook [Coo71]. Pour cela, Cook a utilisé des réductions en temps polynomial pour comprendre les relations liant les problèmes de NP et le problème SAT : étant donnés deux problèmes Π_1 et Π_2, on dit qu'il existe une réduction (en temps polynomial, qu'on omet de préciser par la suite car on ne considère que des réductions de ce type) depuis Π_1 vers Π_2 si il existe un algorithme déterministe s'exécutant en temps polynomial qui, pour toute instance $I(\Pi_1)$ de Π_1, la transforme en une instance $I(\Pi_2)$ de Π_2 et tel que la réponse à Π_1 sur $I(\Pi_1)$ est "vrai" si et seulement si la réponse à Π_2 sur $I(\Pi_2)$ est "vrai".

Cette relation entre deux problèmes est très forte puisqu'elle permet souvent d'étendre les résultats d'appartenance à une classe de l'un des problèmes à l'autre. Par exemple, en l'occurence, si Π_1 est NP-complet, alors Π_2 l'est aussi. C'est de cette manière que sont désormais démontrées les résultats de NP-complétude : on prouve d'abord que le problème auquel on s'intéresse est dans NP, puis on prouve qu'il existe une réduction depuis un problème dont la NP-complétude a d'ores et déjà été prouvée.

Cook n'a pu avoir recours à cette méthode. Pour prouver que SAT est NP-complet, il a prouvé qu'il était plus dur que tous les problèmes dans NP, c'est à dire que pour tout problème dans NP il existait une réduction vers SAT, impliquant que si SAT était décidable en temps polynomial, alors tous les problèmes de NP le seraient (et donc $P = NP$).

La dernière classe de problèmes qui nous intéresse ici est celle des problèmes NP-difficiles. Elle se compose de tous les problèmes plus durs que tous les problèmes NP-complets, *i.e.* pour tout problème NP-complet il existe une réduction vers chacun des problèmes de cette classe.

La notion de NP-difficulté cependant s'étend à des problèmes qui ne sont pas nécessairement des problèmes de décision, mais qui peuvent être par exemple des problèmes d'optimisation. Un problème d'optimisation Π consiste en :

- un ensemble d'instances $\mathcal{I}(\Pi)$,
- pour chaque instance $I \in \mathcal{I}(\Pi)$, un ensemble de solutions $S_\Pi(I)$.

Un algorithme résout le problème Π si, étant donnée une instance $I \in \mathcal{I}(\Pi)$ il renvoie une solution s de $S_\Pi(I)$, ou "faux" si $S_\Pi(I) = \emptyset$.

Tout problème de décision peut être vu comme un problème d'optimisation pour lequel l'ensemble des solutions est {"*vrai*"} quand la réponse est "vrai" pour une instance donnée, et l'ensemble vide sinon. Cette généralisation de la classe NP-difficile aux problèmes d'optimisation est possible car la notion de réduction depuis un problème de décision vers un autre peut être généralisée pour définir cette relation entre deux problèmes d'optimisation. Comme tout problème de décision peut être vu comme un problème d'optimisation, on peut prouver que ces problèmes peuvent également être "au moins aussi durs" que tous les problèmes de décision NP-complets correspondants.

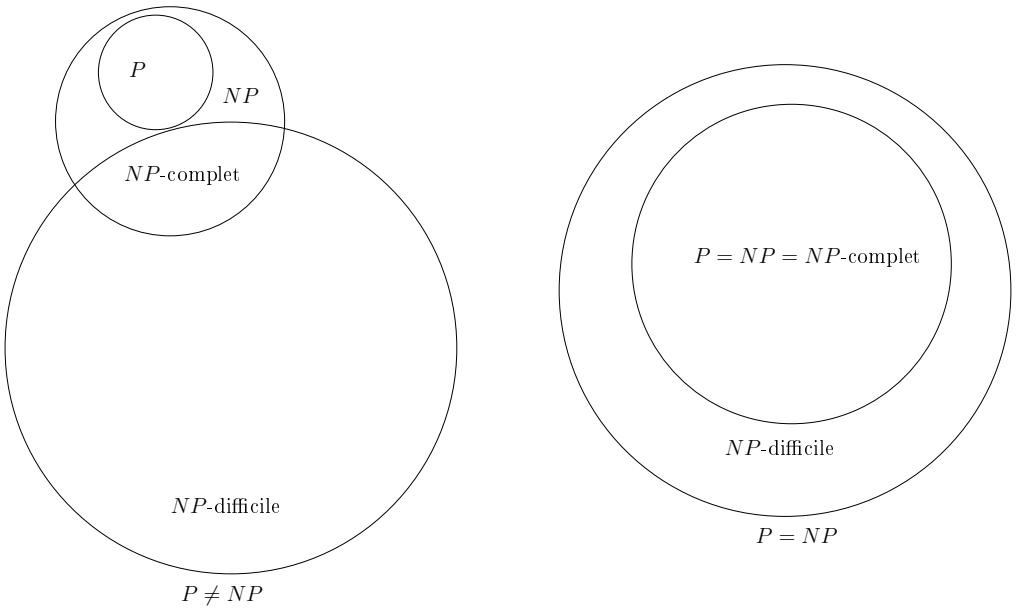


FIG. 1.1 – Représentation des inclusions des classes de problèmes décrites dans ce paragraphe, en fonction de la justesse de la conjecture $P \neq NP$.

La Figure 1.1 illustre les relations d'inclusion entre les différentes classes présentées. Notons qu'il existe certains problèmes, comme le problème de décider s'il existe un isomorphisme entre deux graphes, qu'on sait être dans NP mais pour lesquels on ne sait ni dire s'ils sont dans P, ni dire s'ils sont NP-complets [KST93].

Un problème d'optimisation Π est soit un problème de minimisation soit un problème de maximisation et peut également être défini par la donnée de :
- un ensemble d'instances $\mathcal{I}(\Pi)$,
- pour chaque instance $I \in \mathcal{I}(\Pi)$, un ensemble de solutions candidates ou valides $S_\Pi(I)$,
- une fonction m_Π qui affecte à chaque instance $I \in \mathcal{I}(\Pi)$ et chaque solution $s \in S_\Pi(I)$ un rationnel positif $m_\Pi(I, s)$ appelé valeur de s.

Si Π est un problème de maximisation (resp. de minimisation) alors une solution optimale pour une instance $I \in \mathcal{I}(\Pi)$ est une solution valide $s \in S_\Pi(I)$ telle que, pour toute autre solution valide $s' \in S_\Pi(I)$, $m_\Pi(I, s) \geq m_\Pi(I, s')$ (resp. $m_\Pi(I, s) \leq m_\Pi(I, s')$). On note $\text{OPT}_\Pi(I)$ cette solution et $|\text{OPT}_\Pi(I)|$ la valeur $m_\Pi(I, \text{OPT}_\Pi(I))$.

Pour étudier un problème d'optimisation on peut se ramener au problème de décision associé aux valeurs que peut prendre une solution à ce problème. Dans notre exemple, supposons que Π soit un problème de maximisation, fixons une valeur K rationnelle positive, le problème de décision concerné est "étant donnée une instance $I \in \mathcal{I}(\Pi)$, existe-il une solution valide $s \in S_\Pi(I)$

telle que $m_\Pi(I,s) \geq K$?". La donnée d'un algorithme qui permet, pour tout K et pour toute instance $I \in \mathcal{I}(\Pi)$, de décider ce problème, permet de résoudre le problème d'optimisation Π en temps polynomial (via le test en temps polynomial de plusieurs valeurs de K). Ainsi dans la suite de ce manuscrit pour prouver qu'un problème Π d'optimisation est NP-difficile nous prouvons que le problème de décision correspondant est NP-difficile, voire NP-complet.

Etant donné un problème d'optimisation NP-difficile, il n'existe pas d'algorithme permettant de trouver une de ses solutions optimales en temps polynomial (à moins, donc, que $P = NP$). Comme nous souhaitons travailler uniquement sur des algorithmes ayant des temps d'exécution polynomiaux, on est alors forcé de relâcher la contrainte d'optimalité, et de se contenter de trouver une "bonne" solution à défaut de la meilleure.

On utilise alors des algorithmes d'approximation : étant donné un problème d'optimisation Π, $\mathcal{A}$ est un algorithme d'approximation pour Π si, pour n'importe quelle instance $I \in \mathcal{I}(\Pi)$, $\mathcal{A}$ trouve une solution $s \in S_\Pi(I)$. On note $|\mathcal{A}(I)|$ la valeur $m_\Pi(I,s)$ obtenue par $\mathcal{A}$.

Etant donné un algorithme d'approximation $\mathcal{A}$ pour Π, on définit $R_\mathcal{A}^N$ comme :

- $R_\mathcal{A}^N = \min\left\{\frac{|\mathcal{A}(I)|}{|\textsc{Opt}_\Pi(I)|} \text{ pour } I \in \mathcal{I}(\Pi), |\textsc{Opt}_\Pi(I)| = N\right\}$ si Π est un problème de maximisation,
- $R_\mathcal{A}^N = \max\left\{\frac{|\mathcal{A}(I)|}{|\textsc{Opt}_\Pi(I)|} \text{ pour } I \in \mathcal{I}(\Pi), |\textsc{Opt}_\Pi(I)| = N\right\}$ si Π est un problème de minimisation.

Nous appelons $R_\mathcal{A}^N$ le facteur d'approximation de $\mathcal{A}$ pour le problème Π. $R_\mathcal{A}^N$ est toujours inférieur à 1 dans un problème de maximisation, respectivement toujours supérieur à 1 dans un problème de minimisation.

On peut également définir le facteur d'approximation asymptotique $R_\mathcal{A}^\infty = \limsup_{N\to\infty} R_\mathcal{A}^N$ pour un problème de maximisation, et $R_\mathcal{A}^\infty = \liminf_{N\to\infty} R_\mathcal{A}^N$ pour un problème de minimisation.

Un *APTAS* (Schéma asymptotique d'approximation en temps polynomial) pour un problème Π est une famille d'algorithmes $\{\mathcal{A}_\varepsilon\}_{\varepsilon>0}$ qui, quand la taille d'une instance tend vers l'infini (*i.e.* quand la valeur $|\textsc{Opt}_\Pi(I)|$ de la solution optimale sur une instance I donnée tend vers l'infini), assure un facteur d'approximation de $1-\varepsilon$.

On peut à présent parler d'inapproximabilité d'un problème. Etant donné un problème de maximisation Π (resp. de minimisation), on dit que Π est inapproximable à un facteur $R \leq 1$ (resp. $R \geq 1$) si il n'existe pas d'algorithme polynomial assurant un facteur d'approximation $R' \geq R$ (resp. $R \leq R$), à moins que $P = NP$. Pour démontrer un résultat d'inapproximabilité, nous utilisons à nouveau des réductions d'un problème à un autre. Si un problème Π_1 est inapproximable à un facteur R connu, et qu'on prouve qu'il existe une réduction depuis Π_1 vers un second problème Π_2, alors Π_2 est également inapproximable à un facteur R.

1.2.2 Espaces Métriques

Les problèmes que nous étudions considèrent des ensembles d'éléments plongés dans un espace métrique.

Un *espace métrique* est un ensemble S au sein duquel une notion de distance $d : S \times S \to \mathbb{R}^+$ est définie. Etant donné un espace métrique (S,d), rappelons qu'une distance vérifie les propriétés suivantes :

- symétrie : $\forall x, y \in S^2, d(x,y) = d(y,x)$,
- séparation : $\forall x, y \in S^2, d(x,y) = 0 \Leftrightarrow x = y$,
- inégalité triangulaire : $\forall x, y, z \in S^3, d(x,z) \leq d(x,y) + d(y,z)$.

Une fois qu'un espace métrique est bien défini, il est possible de définir des *boules* dans cet espace. Etant donnés un espace métrique (S,d), un élément $s \in S$ et un nombre $r \in \mathbb{R}$, la boule (fermée) centrée en s et de rayon r, notée $B(s,r)$, est définie comme suit :
$B(s,r) = \{x \in S, d(s,x) \leq r\}$.

Etant donnés l'espace métrique $(\mathbb{R}^D, d)$ et deux éléments $x, y \in \mathbb{R}^{D^2}$ de coordonnées respectivement $x = (x_1, \ldots, x_D)$ et $y = (y_1, \ldots, y_D)$, on définit la norme n comme la distance entre les vecteurs de coordonnées x et y. Une telle norme définit ainsi une distance entre les éléments x et y de $(\mathbb{R}^D, d)$. On s'intéresse à cette notion car dans la suite nous utilisons tour à tour les *normes* L_∞ et L_2 pour définir les distances sur les espaces métriques considérés. La distance en utilisant la norme L_∞ entre x et y est $d(x,y) = \max(|x_1 - y_1|, \ldots, |x_D - y_D|)$. La distance entre x et y en utilisant la norme L_2 est $d(x,y) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_D - y_D)^2}$.

Ces deux normes sont équivalentes, *i.e.* pour toute paire (x,y) d'éléments de $\mathbb{R}^D$,
$d_\infty(x,y) \leq d_2(x,y) \leq \sqrt{D}.d_2(x,y)$, où $d_N(x,y)$ est la distance entre x et y telle que définie par la norme L_N. Comme ces deux normes sont équivalentes, travailler avec l'une ou l'autre nous fournit des résultats équivalents à un facteur près dans chacun des cas étudiés (mais ne nous permet pas d'attendre des facteurs d'approximation proches de 1).

Par la suite, nous utilisons quelques espaces métriques particuliers :
- L'espace dans lequel les éléments ont des coordonnées dans $\mathbb{R}^D$, et où la norme L_∞ est utilisée pour définir les distances. Les boules dans cet espace métrique sont des hypercubes (D-dimensionnels) pleins, comme dans la Figure 1.2.
- Le plan euclidien, dans lequel les éléments sont placés dans le plan $\mathbb{R}^2$. Les boules dans cet espace métrique sont des disques, comme dans la Figure 1.3.
- L'espace défini par un graphe, muni de la distance dans ce graphe. Etant donné un graphe $G = (V,E)$, dans lequel les arêtes sont étiquetées avec des valeurs positives, on peut définir la longueur d'un chemin comme la somme des longueurs des arêtes parcourues par ce chemin. La distance entre deux points x et y de V est la longueur du plus court chemin pondéré dans G. Notons qu'une même distance peut être engendrée par plusieurs graphes différents. L'ensemble des éléments de G muni de cette distance induite forme un espace métrique. Les boules dans un tel espace dépendent du graphe, mais si l'on considère un graphe dans lequel toutes les arêtes sont de poids 1, étant donné un sommet $v \in V$, $B(v,1)$ est composée exactement de v et de ses voisins dans G, comme présenté dans la Figure 1.4.

1.2.3 Modèle d'algorithmique distribuée

Dans ce manuscrit nous élaborons à de multiples reprises des algorithmes distribués. Pour cela il est nécessaire de définir le modèle d'environnement distribué que nous utilisons, ce que nous faisons dans ce paragraphe, en nous inspirant de plusieurs travaux [Lav95, Lyn96, Pel00].

Tout d'abord repenchons nous sur les problèmes qui nous intéressent et leur contexte : leur cadre d'application considère des réseaux à grande échelle, fortement dynamiques, dont les caractéristiques sont très difficilement centralisables, impossible à collecter en un point.

Pour cela on cherche à maintenir dans ce type de réseau un réseau de communication restreint,

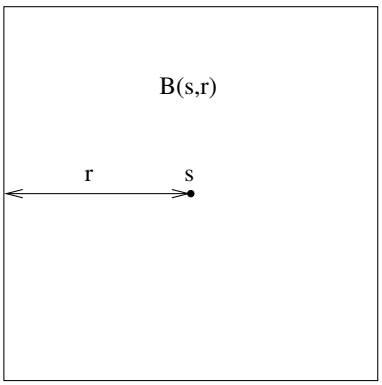


FIG. 1.2 – Représentation d'une boule centrée en un élément s et de rayon r dans le plan muni de la norme L_∞.

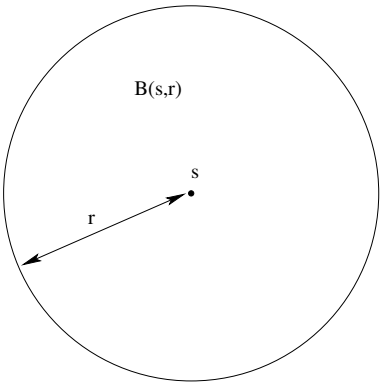


FIG. 1.3 – Représentation d'une boule centrée en un élément s et de rayon r dans le plan euclidien.

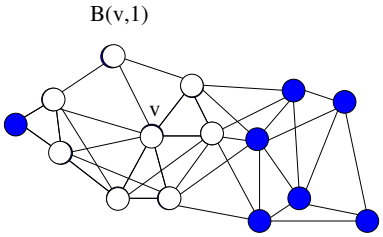


FIG. 1.4 – Représentation de la boule $B(v,1)$ centrée en v et de rayon 1 dans l'espace défini par ce graphe muni de la distance dans ce graphe. Cette boule contient exactement v et tous ses voisins.

autorisant chaque noeud à communiquer à un nombre de préférence petit de voisins, ce qui économise la quantité d'informations que chaque noeud doit conserver localement pour pouvoir communiquer avec le monde extérieur. Ce genre de réseau de communications doit cependant assurer certaines propriétés, en fonction des besoins des applications qu'on souhaite y porter. Les propriétés les plus courantes sont entre autres que dans un tel réseau les temps de recherche d'un noeud à un autre restent courts, que la localité dans le réseau originel soit conservée (que les voisins d'un noeud dans le réseau originel puissent être identifiés facilement), et que le réseau de communication permette un bon équilibrage de charge et une forte tolérance aux pannes. On peut citer comme réseau de communication de ce type Chord [SMK+01] et CAN [RFH+01].

Dans nos travaux, nous utilisons des réseaux de communications particuliers, que nous présentons dans les paragraphes idoines. Nous utilisons dans le Chapitre 4 respectivement des *skip-lists* combinées à des *Z-ordres*, des *skip-graphs* en lieu et place de ces skip-lists, puis des arbres dans le Chapitre 5.

Muni de nos réseaux de communication pour travailler dans un environnement distribué, nous pouvons à présent définir un *système distribué* : c'est un ensemble d'entités ayant leurs propres moyens de calculs, communiquant entre eux via des liens de communications. Pour nous ces entités sont les noeuds des réseaux à grande échelle qui nous intéressent. Nous appliquons les restrictions suivantes à nos systèmes distribués :

- Nous ne faisons aucun cas des problèmes de tolérance aux pannes. Bien que les réseaux à grande échelle soient doublement hautement dynamiques, au sens où d'une part des noeuds y apparaissent et disparaissent souvent, et où d'autre part la latence liant deux noeuds varie en fonction du traffic dans le réseau, notre étude ne s'est pas intéressée à ces problématiques. C'est une piste d'étude intéressante et nécessaire à la mise en application de nos résultats.
- Nous considérons que chaque noeud des réseaux considérés possède un identifiant unique de taille raisonnable (par exemple, son adresse IP si on travaille dans le réseau d'Internet), ce qui semble une considération tout à fait raisonnable dans le cadre d'application qui est le nôtre.

– Nous considérons que les calculs locaux en chaque noeud ne nécessitent aucune ressource (en particulier ni temporelle, ni de calcul, ni de stockage). Pour limiter l'abus de cette relaxation nous faisons en sorte de minimiser les calculs à effectuer par chaque noeud de manière centralisée, et précisons le cas échéant le temps d'exécution requis pour un calcul local important.

Aux seins des réseaux de communication que nous utilisons, chaque sommet est un des noeuds du réseau, et chaque arête représente un canal de communication bidirectionnel entre les noeuds qu'elle relie. Les échanges d'informations entre les noeuds se font par passage de messages au travers de ces liens.

Un poids unitaire est affecté à chacune des arêtes du réseau de communication. Ainsi on considère que pour toute paire de noeuds, le temps de transmission d'un message d'un noeud à l'autre ne dépend pas de la distance les séparant. Dans les réseaux qui nous intéressent, la latence est malgré tout un facteur très important qui concerne le temps de transmission des messages entre deux noeuds. Nous choisissons, plutôt que d'essayer de quantifier le temps que peut prendre un échange de messages entre deux noeuds, de travailler dans un environnement asynchrone. Dans un tel environnement, chaque message est transmis en un temps fini mais indéterminé.

Dans ce cadre nous définissons la complexité en temps d'un algorithme asynchrone comme le fait Peleg dans [Pel00], *i.e.* comme le nombre d'unités de temps depuis le début de cet algorithme jusqu'à sa fin dans le pire des cas (*i.e.* pour toute instance en entrée de l'algorithme, et pour tout scénario d'asynchronie durant l'exécution de l'algorithme), en considérant que chaque message est transmis en un temps unitaire. Cette hypothèse n'impose pas une borne sur le temps de transmission d'un message, mais est uniquement utilisée pour pouvoir quantifier le temps d'exécution d'un tel algorithme.

Par ailleurs certains algorithmes que nous présentons sont des algorithmes probabilistes. Les résultats que nous obtenons pour ces algorithmes sont des résultats vrais *avec forte probabilité*. Un événement $E(n)$ dépendant d'un paramètre n est vrai avec forte probabilité si et seulement si $\mathbb{P}(E = "vrai") = 1 - o(\frac{1}{n^k})$ pour tout k.

1.3 Contributions de cette thèse

Dans le Chapitre 2, nous introduisons les problèmes de Bin Covering *avec Contrainte de Distance* (BCCD) et de Bin Packing *avec Contrainte de Distance* (BPCD), qui correspondent à la formalisation théorique des problèmes auxquels nous nous sommes intéressés et que nous avons présentés dans le Paragraphe 1.1. Succinctement, étant donnés un ensemble de noeuds dans un réseau, BCCD modélise le problème de construire un maximum de groupes de noeuds de capacité suffisante (fixée) et de diamètre borné (représentant la latence maximale entre deux noeuds d'un même groupe), et BPCD modélise le problème d'identifier un minimum d'ensembles de noeuds de telle sorte que chaque noeud soit dans un de ces ensembles, que chaque ensemble ne dépasse pas une certaine capacité (correspondant à la capacité maximale que peut gérer un serveur auquel on associerait chaque ensemble, par exemple), et de diamètre borné également (pour qu'un serveur n'ait pas à gérer des noeuds situés trop loin les uns des autres).

Dans le Chapitre 2 nous présentons également un survol de différents problèmes gravitant autour de BCCD et BPCD, comme le problème du K-centre avec capacités en particulier. Nous

détaillons alors les liens étroits et subtils qui existent entre différentes variantes de problèmes plus généraux de placement de serveurs et BPCD.

Dans le Chapitre 3, nous menons notre étude en modélisant les réseaux visés par nos applications par des espaces métriques quelconques. Pour travailler dans ce cadre, nous introduisons le procédé d'augmentation de ressources également évoqué précédemment. Plus précisément nous autorisons nos algorithmes à construire des groupes de diamètre supérieur d'un facteur $\beta \geq 1$ aux groupes des solutions optimales auxquelles leurs performances sont comparées.

Munis de ce procédé, nous démontrons que même avec une augmentation de ressources d'un facteur $(2-\varepsilon)$ les deux problèmes, BCCD et BPCD, sont inapproximables à facteur constant. Par ailleurs, nous démontrons également qu'il n'existe pas d'algorithme d'approximation pour BCCD assurant un facteur d'approximation strictement supérieur à $1/2$, quelle que soit l'augmentation de ressources utilisée, à moins que $P = NP$. De même pour BPCD, il n'existe d'algorithme d'approximation assurant un facteur d'approximation strictement inférieur à $3/2$ quelle que soit l'augmentation de ressources utilisée, et ce dans n'importe quel espace métrique, à moins que $P = NP$.

Dans le même cadre de travail, *i.e.* en travaillant dans des espaces métriques généraux et avec de l'augmentation de ressources, nous présentons ensuite un algorithme d'approximation centralisé pour BCCD assurant un facteur d'approximation de $\frac{2}{5}$ en ayant recours à une augmentation de ressources d'un facteur 4. Nous examinons également l'apport que peut fournir un relâchement de la contrainte de capacité de chaque groupe sur le facteur d'approximation, en autorisant les groupes construits à être de capacité inférieure à la borne fixée, et en comparant la solution ainsi produite par notre algorithme à une solution optimale contrainte à créer des groupes de la capacité fixée, non relachée.

Ensuite nous présentons un algorithme d'approximation centralisé pour BPCD assurant un facteur d'approximation de $\frac{7}{3}$ en ayant recours à une augmentation de ressources de 2, optimale dans le sens où l'utilisation d'une augmentation de ressources inférieure ne permet pas d'obtenir d'approximation à facteur constant de BPCD. Nous examinons également sa version uniforme.

Nous revenons alors sur le problème de K-centre avec capacités et détaillons sa correspondance avec BPCD, en observant que nos algorithmes pour BPCD fournissent des résultats intéressants sur les variantes uniformes et non uniformes de ce problème.

Dans le Chapitre 4 nous restreignons notre étude de BCCD et BPCD aux cas où l'on modélise les réseaux à grande échelle par des espaces métriques particuliers, qui sont utilisés par différents outils pour modéliser l'espace des latences sur Internet. Plus précisément, nous travaillons sur :
- $\mathbb{R}^D$ muni de la norme L_∞ pour définir les distances,
- le plan muni de la norme L_2 ou de la norme L_∞,
- une généralisation du plan euclidien,
- les espaces métriques induits par les arbres.

Ainsi en premier lieu nous proposons plusieurs résultats, utilisant ou non une augmentation de ressources, lorsque les éléments d'une instance de BCCD sont plongés dans un espace bidimensionnel muni de la norme L_∞.

Ensuite nous présentons un algorithme distribué d'approximation pour BCCD, inspiré de l'algorithme centralisé présenté précédemment, qui assure un facteur d'approximation de $\frac{1}{3}$ en ayant recours à une augmentation de ressources d'un facteur 4. Cet algorithme fonctionne sur les instances de BCCD dans lequel l'espace métrique contenant les éléments est $\mathbb{R}^D$, pour une

dimension D fixée (faible), muni de la norme L_∞ pour définir les distances.

Il se base sur une structure de skip-graph, dont les performances sont étudiées dans le Paragraphe 4.4, nous assurant que notre algorithme distribué, dans le cadre du modèle de calculs distribués que nous avons défini, fonctionne en $O(\log n)$ unités de temps avec forte probabilité.

Enfin nous étudions BPCD lorsque les instances utilisent comme espace métrique le plan euclidien, en présentant dans ce cadre une famille d'algorithmes d'approximation centralisés assurant un facteur d'approximation de $\frac{5}{2} + \varepsilon$ pour tout $\varepsilon > 0$.

Dans le Chapitre 5 nous étudions divers outils permettant de modéliser la structure du réseau d'Internet, qui fait partie des domaines d'applications de l'étude de BCCD et de BPCD. En particulier, nous nous intéressons à l'utilisation de Vivaldi et de Sequoia, que nous décrivons dans ce chapitre.

Dans Vivaldi [DCKM04, CDK+04], mis au point par Dabek *et al.*, les hôtes se voient attribuer des coordonnées dans le plan euclidien, en plus d'une hauteur, tandis que Sequoia [ABK+07, RMK+09] permet de plonger un espace vérifiant certaines propriétés dans un arbre dont les feuilles seraient les noeuds du réseau, et dans lequel les noeuds internes seraient virtuels. Ramasubramanian *et al.* ont prouvé que le réseau composé des noeuds d'Internet et des latences les reliant vérifiait ces propriétés particulières et ont donc présenté un mécanisme de plongement d'Internet dans un tel arbre.

Nous nous intéressons alors dans la suite du Chapitre 5 à l'étude de BCCD et de BPCD dans les espaces métriques générés par ces outils. Nous présentons des simulations pour chacun des algorithmes élaborés pour BCCD comme pour BPCD dans les espaces métriques générés par ces outils, de façon à comparer leurs performances appliquées au cadre de nos travaux.

Dans le cadre de l'utilisation de Vivaldi, et en considérant les noeuds d'un réseau placés dans un espace métrique tel que ceux générés par cet outil, nous présentons un algorithme pour BCCD assurant un facteur d'approximation de $\frac{2}{5}$ en utilisant une augmentation de ressources d'un facteur 3.

En ce qui concerne l'utilisation de Sequoia, nous présentons un algorithme d'approximation pour BCCD assurant un facteur d'approximation de $\frac{1}{3}$ en ayant recours à une augmentation de ressources d'un facteur 2, et un algorithme d'approximation pour BPCD assurant un facteur d'approximation de $\frac{5}{2}$ en ayant aussi recours à une augmentation de ressources d'un facteur 2.

Première partie

Introduction de contrainte de distance à deux problèmes classiques, le Bin Covering et le Bin Packing

Chapitre 2

Bin Covering/Bin Packing avec Contrainte de Distance

Le problème de *Bin Covering* [AJKL84] peut se rapprocher d'un problème de poires au sirop : lors de la mise en boîte de conserves des poires au sirop, chaque boîte doit contenir une masse de poires au moins égale à celle affichée sur la boîte. Disposant d'un ensemble, disons, de demi-poires au sirop, chacune ayant sa masse propre, l'objectif est alors de remplir un maximum de boîtes de conserves sous la contrainte que chaque boîte contienne une masse suffisante de poires au sirop (nous ne prenons pas ici en compte la masse de sirop, ni la contrainte spécifiant un éventuel nombre constant de demi-poires par boîte).

Le problème de *Bin Packing* [EGCGJ97] est souvent évoqué quand il s'agit de partir en vacances : au moment de faire nos valises, on dispose d'un ensemble important de bagages dans lesquels empaqueter nos affaires. En l'occurence, ces bagages ont tous la même contenance. Par contre, chacune des affaires que nous souhaiterions emporter a son propre volume. Notre objectif est alors d'empaqueter toutes nos affaires dans un minimum de valises, c'est à dire en perdant le moins de place possible dans chacune de celles que nous emportons.

De nombreuses variantes des versions plus formelles de chacun de ces problèmes ont été étudiées. Ici nous étudions une généralisation de ces problèmes dans laquelle les éléments sont placés dans un espace métrique et la distance maximale entre deux éléments appartenant à une même *bin* (groupe d'éléments, groupe dans la suite) doit être inférieure à une certaine valeur limite, $d_{\max}$. Ces deux généralisations que nous allons introduire dans la suite sont nommées respectivement *Bin Covering avec Contrainte de Distance* (BCCD) et *Bin Packing avec Contrainte de Distance* (BPCD).

2.1 De Bin Covering à BCCD

2.1.1 Définition du problème et travaux liés

Comme dit précédemment, Bin Covering est un problème NP-difficile bien étudié (cf. [AJKL84, CJK01] pour les principaux travaux sur ce problème). Dans la variante que nous nous proposons d'étudier, l'objectif est de maximiser le nombre de groupes de diamètre (distance

maximale entre deux éléments d'un même groupe) inférieur à $d_{\max}$ et dont le poids dépasse une certaine valeur fixée, W.

Il est à noter qu'une solution plaçant tous les éléments dans un groupe n'est pas nécessairement une solution valide pour ce problème, *i.e.* respectant les contraintes de distance et de poids. Sur certaines instances il n'existe pas de solution plaçant tous les éléments dans un groupe en respectant ces contraintes.

Un *APTAS* (Schéma asymptotique d'approximation en temps polynomial) est connu pour le problème de BIN COVERING [CJK01]. Un algorithme pour BIN COVERING assurant un facteur d'approximation asymptotique de $\frac{3}{4}$ est connu [AJKL84], mais les algorithmes gloutons ne permettent pas d'obtenir des facteurs d'approximation supérieurs à $\frac{1}{2}$. Parmi eux, l'algorithme *Next-Fit* [AJKL84] est une stratégie simple que nous utilisons par la suite : elle consiste à prendre les éléments un par un et à les placer dans le groupe courant, jusqu'à ce que le poids total de celui-ci dépasse la valeur seuil W. Quand un tel poids est atteint, le groupe est fermé et la construction d'un nouveau groupe commence. Cette stratégie assure un facteur d'approximation de $\frac{1}{2}$.

Par ailleurs, bien qu'il existe un schéma d'approximation *asymptotique* pour le problème de BIN COVERING, il n'existe pas d'algorithme assurant un facteur d'approximation supérieur à $1/2$ pour ce problème en temps polynomial, à moins que $P = NP$, quand la taille de l'instance est bornée (donc il n'existe pas de tel algorithme fonctionnant quelle que soit l'instance). Ceci se montre via une réduction depuis le problème 2-partition [CJK01].

Clairement, BCCD est NP-difficile, puisque si on supprime la contrainte de distance (ou quand $d_{\max}$ est plus grand que la plus grande des distances entre deux éléments), on se ramène au problème de BIN COVERING classique. Dans le Paragraphe 3.2.1 nous présentons des résultats précis concernant l'inapproximabilité de BCCD. On peut cependant déjà évoquer qu'il existe une réduction simple au problème de savoir s'il existe une clique de cardinalité fixée dans un graphe, que nous détaillons plus tard, impliquant que BCCD est inapproximable à un facteur constant (*i.e.* ne dépendant pas du nombre d'éléments de l'instance).

2.1.2 Notations

Une instance $\mathcal{I}$ de BCCD peut être décrite par un 5−uplet $\mathcal{I} = (S, d, w, W, d_{\max})$ où S est un ensemble d'éléments $S = \{e_1, \ldots, e_n\}$, (S, d) est un espace métrique fini, w une fonction de poids, W une valeur de poids limite, et $d_{\max}$ un diamètre limite.

Pour des raisons de simplicité, on fixe dans la suite $W = 1$ et on normalise les poids des éléments (*i.e.* on les divise par W). Par ailleurs, on ne considère pas les éléments de poids supérieur à 1 (après normalisation), puisque de tels éléments peuvent former un groupe à eux seuls. Une instance de BCCD peut finalement être décrite par un 4−uplet $\mathcal{I} = (S, d, w, d_{\max})$, où $w : S \to [0; 1[$, d'où la définition suivante :

Définition 2.1 (*BCCD :* BIN COVERING *avec Contrainte de Distance*)

Etant donnée une instance $\mathcal{I} = (S, d, w, d_{\max})$*, trouver une collection de sous-ensembles disjoints* $S_1, \ldots S_K$ *de* S *de cardinalité maximale* K *telle que* $\forall i \leq K, \quad \sum_{e \in S_i} w(e) \geq 1$, $\forall (e_u, e_v) \in S_i, \quad d(e_u, e_v) \leq d_{max}$.

On note $\text{OPT}_{BCCD}(\mathcal{I})$ (ou simplement OPT_{BCCD}) une solution où la valeur de K est maximale

pour une instance $\mathcal{I}$ donnée, et $|\text{Opt}_{BCCD}(\mathcal{I})|$ cette valeur K maximale.

Une instance de BCCD est proposée dans les Figures 2.1(a) et 2.1(b) : dans ces figures le graphe de compatibilité (nous le définissons plus précisément dans le Paragraphe 2.3) connectant deux éléments à moins de $d_{\max}$ l'un de l'autre est représenté. Les points sont placés dans le plan euclidien. La valeur de $d_{\max}$ est également indiquée dans chacune de ces figures. Le poids de chaque élément est indiqué à proximité de celui-ci. Dans la Figure 2.1(a) est proposée une solution quelconque de BCCD sur cette instance, dans la Figure 2.1(b), une solution optimale. Remarquons que les solutions présentées dans ces figures, bien que valides, ne placent pas tous les éléments dans un groupe. En l'occurence c'est impossible en respectant à la fois les contraintes de poids et de diamètre.

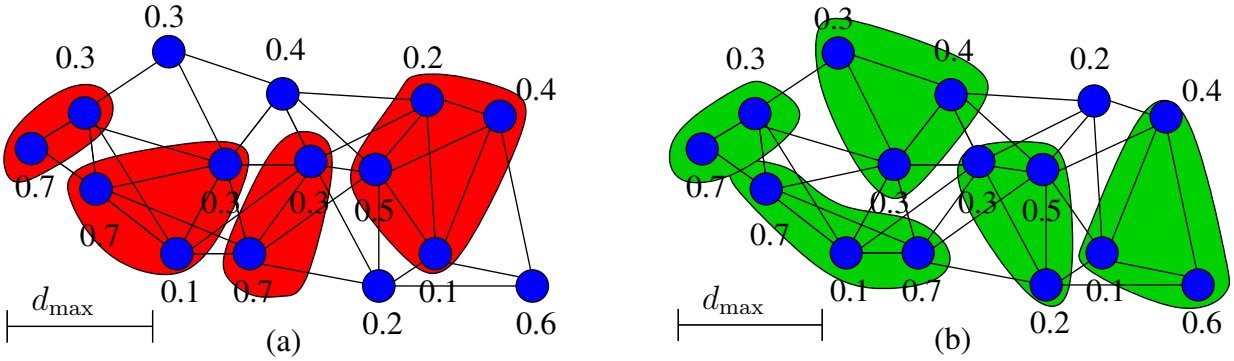


Fig. 2.1 – (a) Une instance de BCCD dans la plan euclidien, et une solution sur cette instance construisant 4 groupes. (b) La même instance de BCCD et une solution optimale sur cette instance, construisant 5 groupes. Ces deux instances utilisent le même graphe de compatibilité connectant deux éléments à moins de $d_{\max}$l'un de l'autre.

2.2 De Bin Packing à BPCD

2.2.1 Définition du problème et travaux liés

Tout comme Bin Covering, Bin Packing est un problème classique dont on peut trouver une revue de différentes variantes dans [EGCGJ97]. Ici, nous nous intéressons à une généralisation similaire à celle de BCCD : les éléments appartiennent à un espace métrique, et la distance maximale entre deux éléments appartenant à un même groupe doit être inférieure à une certaine valeur limite, $d_{\max}$. L'objectif est alors de construire un minimum de groupes satisfaisant cette contrainte de distance et dont le poids ne dépasse pas une certaine valeur fixée, de façon à ce que chaque élément soit exactement dans un groupe (il est toujours possible de mettre chaque élément dans un groupe, même s'il doit en être le seul élément, ce qui n'est pas le cas pour BCCD).

Pour revenir sur ECCD [LMY98], le problème de la couverture du plan euclidien par un ensemble de disques de capacité et de rayon fixés, un des problèmes présentés dans le Paragraphe 1.1, on peut remarquer que le problème ECCD est moins général que BPCD, dans le sens où les espaces métriques considérés par notre étude ne sont pas nécessairement euclidiens. Une autre différence notable entre ces deux problèmes est qu'une solution à ECCD cherche une

couverture par des boules centrées et de rayon fixé (des disques dans le plan euclidien), tandis que dans BPCD on cherche une couverture par des ensembles de diamètre fixé, sans notion de centre. Cette distinction est importante car, pour prendre l'exemple du plan, il existe des ensembles de points de diamètres inférieurs à 1 mais qui ne peuvent être couverts par des boules de rayon $\frac{1}{2}$. Enfin, dans ECCD, les centres ne sont pas contraints à faire partie de l'ensemble des éléments de l'instance.

Un *APTAS* est également connu pour le problème de Bin Packing [dlVL81]. Des algorithmes simples sont aussi connus, assurant des approximations légèrement supérieures à 1.15 (cf. [EGCGJ97] pour une revue de ces résultats). L'algorithme *First-Fit-Decreasing* que nous utilisons est l'un des plus simples : il consiste à trier les éléments à empaqueter dans l'ordre décroissant de leur poids. Ensuite, les éléments sont insérés dans cet ordre dans le premier groupe ayant une capacité restante suffisante. Si aucun groupe n'a une capacité suffisante, un nouveau est créé. Cet algorithme assure un facteur d'approximation asymptotique de $\frac{11}{9}$ (cf. [Yue91]).

Par ailleurs, bien qu'il existe un schéma d'approximation asymptotique pour le problème de Bin Packing, il n'existe pas d'algorithme assurant un facteur d'approximation inférieur à 3/2 pour ce problème en temps polynomial, à moins que $P = NP$, quand la taille de l'instance est bornée (donc il n'existe pas de tel algorithme fonctionnant quelle que soit l'instance). Ceci se montre également via une réduction depuis le problème 2-partition.

Il est clair que le problème du BPCD est NP-difficile, puisque si on supprime la contrainte de distance (ou quand $d_{\max}$est plus grand que la plus grande des distances entre deux éléments), on se ramène au problème de Bin Packing classique. De plus, BPCD est équivalent au problème de *Bin Packing avec Conflits* (BPaC), étudié par Epstein et Levin dans [EL08] :

Définition 2.2 (*BPaC :* Bin Packing *avec Conflits*)

Etant donnés un graphe $G = (V, E)$ et une fonction $w : V \to [0; 1[$, trouver une partition en sous-ensembles disjoints $V_1, \dots V_K$ de V de cardinalité minimale K telle que
$\forall i \leq K, \ \sum_{e \in V_i} w(e) \leq 1, \forall (e_u, e_v) \in V_i, \ \ (e_u, e_v) \notin E.$

Nous dénotons par OPT_{BPaC} la cardinalité d'une solution optimale de BPaC.

Une solution optimale de BPaC est proposée pour l'instance présentée dans la Figure 2.2. Deux éléments font partie d'un même groupe s'ils sont de couleur identique. Il est nécessaire que deux éléments d'un même groupe ne soient pas connectés par une arête.

Une instance de BPCD peut être transformée en une instance de BPaC dans laquelle deux éléments sont en conflits si et seulement si la distance les séparant est supérieure à $d_{\max}$ dans l'instance de BPCD. De manière similaire, une instance de BPaC peut être transformée en une instance de BPCD en fixant $d_{\max}$ à 1, et la distance entre deux éléments à 1 s'ils ne sont pas en conflit, à 2 sinon.

Par ailleurs, Epstein et Levin [EL08] ont prouvé que BPaC est une généralisation du problème de coloration de graphe. D'après Bellare *et al.* [BGS95], ce problème est lui-même dur à approcher mieux qu'à un facteur $|S|^{1/7-\delta}$ pour n'importe quel $\delta > 0$, BPCD l'est donc aussi. Nous présentons des résultats précis dans le Paragraphe 3.2.2 concernant l'inapproximabilité de BPCD.

Dans le Paragraphe 2.2.3, nous présentons le problème du K-centre avec capacités, fortement lié au problème de BPCD.

Dans le Paragraphe 2.2.4 nous essayons d'éclaircir les relations entre tout un ensemble de

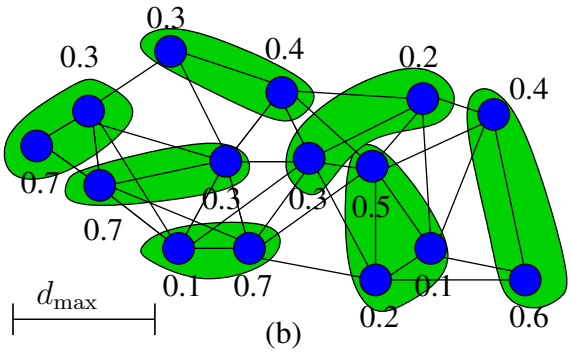


Fig. 2.2 – Une instance de BPaC, et une solution optimale sur cette instance construisant 7 groupes. Le graphe représenté ici est le complémentaire du graphe de conflits de cette instance, ce qui permet de rapprocher celle-ci de l'instance de BPCD de la Figure 2.3

problèmes différents les uns des autres d'un petit ensemble de paramètres, ou n'ayant pas les mêmes variables d'optimisation. Tous ces problèmes ont un lien avec le BPCD, et notamment sa version centrée, qui se rapproche très fortement des problèmes de placements de serveurs et de réplicats évoqués dans le Paragraphe 1.1.

2.2.2 Notations

Toute comme une instance de BCCD, une instance $\mathcal{I}$ de BPCD peut être décrite par un 4−uplet $\mathcal{I} = (S, d, w, d_{\max})$ où S est un ensemble d'éléments $S = \{e_1, \ldots, e_n\}$, (S, d) est un espace métrique fini, w une fonction de poids : $w : S \to [0; 1[$, et $d_{\max}$ un diamètre limite. En effet, on peut normaliser de la même façon les poids en les divisant par la valeur seuil souhaitée pour le poids, et on ne considère pas les éléments de poids supérieur à 1 (après normalisation), qui ne peuvent être placés dans aucun groupe. On obtient donc la définition suivante :

Définition 2.3 (*BPCD :* Bin Packing *avec Contrainte de Distance*)

Etant donnée une instance $\mathcal{I} = (S, d, w, d_{\max})$*, trouver une partition en sous-ensembles disjoints* $S_1, \ldots S_K$ *de* S *de cardinalité minimale* K *telle que* $\forall i \leq K, \quad \sum_{e \in S_i} w(e) \leq 1$, $\forall (e_u, e_v) \in S_i, \quad d(e_u, e_v) \leq d_{max}$.

On note $\text{Opt}_{BPCD}(\mathcal{I})$ (ou simplement Opt_{BPCD}) une solution où la valeur de K est minimale pour une instance $\mathcal{I}$ donnée, et $|\text{Opt}_{BPCD}(\mathcal{I})|$ cette valeur K minimale.

Une instance de BPCD est proposée dans les Figures 2.3(a) et 2.3(b) : dans ces figures le graphe de compatibilité (nous le définissons plus précisément dans le Paragraphe 2.3) connectant deux éléments à moins de $d_{\max}$ l'un de l'autre est représenté. Les points sont placés dans le plan euclidien. La valeur de $d_{\max}$ est également indiquée dans chacune de ces figures. Le poids de chaque élément est indiqué à proximité de celui-ci. Dans la Figure 2.3(a) est proposée une solution valide de BPCD sur cette instance construisant 8 groupes, dans la Figure 2.3(b), une solution optimale en construisant 7. Notons que sans contrainte de distance, sur l'instance de Bin Packing considérant les mêmes éléments pondérés, une solution optimale aurait pu ne construire que 6 groupes.

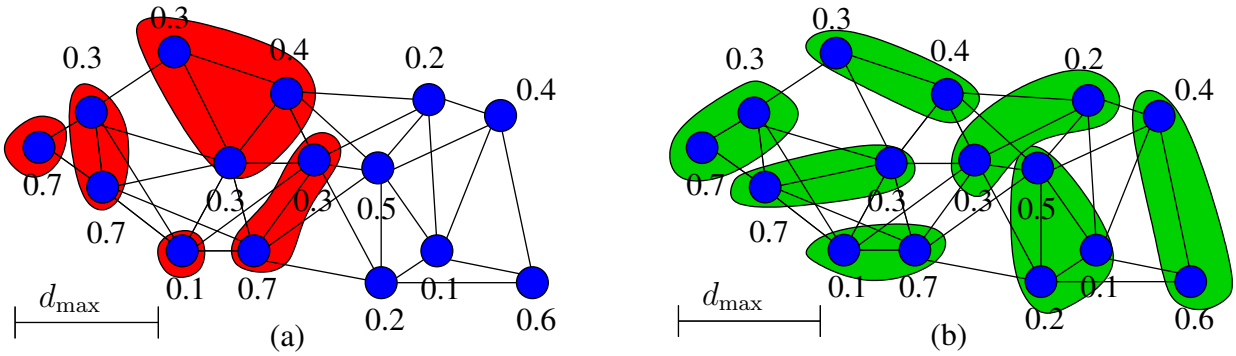


Fig. 2.3 – (a) Une instance de BPCD dans le plan euclidien, et une solution sur cette instance construisant 8 groupes. (b) La même instance de BPCD et une solution optimale sur cette instance, construisant 7 groupes.

Durant nos travaux nous nous sommes rendus compte que de nombreux rapprochements, pas toujours triviaux, peuvent être fait entre différents problèmes et une variante centrée de BPCD, BPCDc, utilisant les mêmes instances que BPCD.

Définition 2.4 (*BPCDc : BPCD centré*)

Etant donnée une instance $\mathcal{I} = (S, d, w, d_{\max})$ de BPCD, trouver un sous-ensemble $X \subseteq S$ de centres de cardinalité minimale et une association des éléments de S aux centres dans X, telle que chaque élément de S soit à distance inférieure à $d_{\max}$ du centre auquel il est associé, et que le poids total des éléments associés à chaque centre soit inférieur à 1.

Une instance de BPCDc, ainsi qu'une solution optimale sur cette instance, sont proposées dans la Figure 2.4 : dans cette figure le graphe de compatibilité (nous le définissons plus précisément dans le Paragraphe 2.3) connectant deux éléments à moins de $d_{\max}$ l'un de l'autre est représenté. Les points sont placés dans le plan euclidien. La valeur de $d_{\max}$ est également indiquée dans cette figure. Le poids de chaque élément est indiqué à proximité de celui-ci. Les centres de chaque groupe sont les éléments blancs, et tous les éléments d'un même groupe sont dans la même enveloppe.

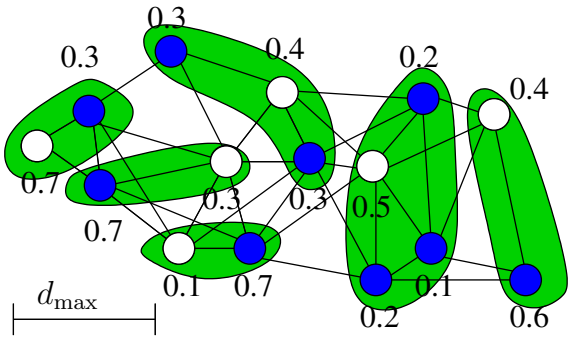


Fig. 2.4 – Une instance de BPCDc dans le plan euclidien, et une solution optimale sur cette instance, construisant 6 groupes.

Les problèmes classiques de placements de serveurs et de réplicats évoqués dans le Paragraphe 1.1 sont en fait bien plus proches de BPCDc que de BPCD. En effet la plupart de ces

problèmes visent à sélectionner un ensemble d'éléments à distinguer pour en faire des centres fournisseurs de services. Les problèmes liés à la satisfaction des demandes des clients peuvent alors se formuler en modulant les capacités et les poids des éléments, et la distance permet de modéliser les temps de communication liés aux échanges entre clients et fournisseurs de services. En particulier, les multiples formulations théoriques de variantes de ces problèmes que nous présentons dans le Paragraphe 2.2.4 sont mis en relation étroite avec BPCDc.

De la même façon que BPCD, ce problème est NP-difficile. Les groupes d'une solution d'une instance $\mathcal{I} = (S, d, w, d_{\max})$ de BPCDc ont un diamètre d'au plus $2d_{\max}$. On peut donc construire une solution à BPCDc depuis une solution au BPCD sur la même instance en choisissant arbitrairement un élément dans chaque groupe de la solution de BPCD pour en faire un centre, et en lui affectant les autres éléments de son groupe, qui sont tous inclus dans sa boule de rayon $d_{\max}$, donc de diamètre au plus $2d_{\max}$.

2.2.3 Lien entre BPCD et K-centre avec capacités

Le problème du "K-centre avec capacités" a été introduit par Bar-Ilan *et al.* dans [BIKP93] sous le nom de "Balanced K-center", et également étudié par Khuller et Sussmann dans [KS00], qui lui ont donné le nom de "Capacitated K-center". C'est une généralisation d'un problème fondamental de placement de facilités, le K-centre [TSRB71], dont on sait qu'il est NP-difficile [GJ+79]. Cette généralisation implique un ensemble d'éléments placés dans un espace métrique, et un entier K, correspondant au nombre de centres à placer dans un espace métrique préalablement choisi. Chaque élément doit être associé à l'un des K centres, tout en respectant la contrainte spécifiant le nombre maximal L d'éléments pouvant être affectés à chaque centre. L'objectif est de minimiser le rayon entre un élément et le centre auquel il est affecté, pour K et L fixés. La variante non uniforme que nous introduisons, Kcapa, s'appuie sur des instances dans lesquelles les éléments peuvent avoir des poids différents. Le problème reste le même, la contrainte sur le nombre maximal d'éléments par groupe devenant une contrainte sur le poids total de chaque groupe.

Définition 2.5 (Kcapa *: K-centre non uniforme avec capacités*)

Etant donnée une instance $\mathcal{K} = (S, d, w, K)$: un ensemble $S = \{e_1, \dots e_n\}$ d'éléments, un espace métrique (S, d), une fonction de poids $w : S \to [0; 1[$, un entier K, trouver la plus petite valeur r_{max}, un ensemble $X \subseteq S$ de centres de taille au plus K et une association des éléments de S aux centres de X tels que le poids total des éléments associés à un même centre soit inférieur à 1, et que chaque élément soit associé à un centre à distance inférieure à r_{max} de lui-même.

Remarque 2.6

Ce ne sont pas les capacités des centres qui ne sont pas uniformes, mais bien les poids de chaque élément qui doivent être assignés aux centres, de capacité uniforme.

On note $\text{Opt}_{\text{Kcapa}}(\mathcal{K})$ (ou simplement $\text{Opt}_{\text{Kcapa}}$) la valeur optimale de r_{max} pour une instance $\mathcal{K}$ donnée.

Une instance de Kcapa, pour $K = 8$, est proposée dans la Figure 2.5. Les éléments sont placés dans le plan euclidien. Les 8 centres sélectionnés sont les éléments blancs, et tous les éléments affectés à chacun de ces centres sont dans la même enveloppe que celui-ci. La distance

maximale entre un centre et un des éléments qui lui est affecté est r_{max} identifié à part, en rouge dans la figure. Notons que dans une instance de KCAPA, la notion de graphe de compatibilité évoquée pour les problèmes BCCD et BPCD n'a pas de sens, puisqu'une instance de KCAPA ne fournit pas de valeur r_{max} qui permettrait de le définir. Au contraire ici la valeur à minimiser est justement la distance maximale entre un élément et un centre de façon à affecter chaque élément à un centre en utilisant seulement 8 centres.

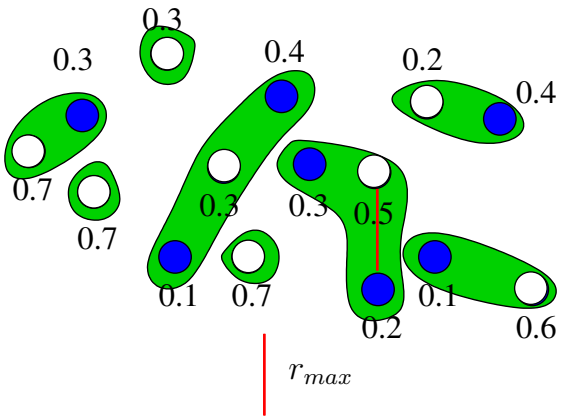


FIG. 2.5 – Une instance de KCAPA pour $K = 8$ dans le plan euclidien, et une solution optimale sur cette instance, pour laquelle on a identifié r_{max}.

La formulation usuelle du problème de K-centre avec capacités (cf. [KS00]), qu'on nomme KCAPA uniforme (KCAPAUNIF), utilise une fonction de poids uniforme, *i.e.* $\forall x \in S, w(x) = w$. La contrainte de poids sert alors simplement à signifier qu'on veut assigner un nombre maximal, $\lfloor 1/w \rfloor$, d'éléments à chaque centre.

KCAPA dans sa version uniforme, comme dans sa version non uniforme, hérite de la NP-difficulté du K-centre classique [GJ+79]. Il n'existe à notre connaissance aucun résultat d'inapproximabilité pour ces problèmes, même dans le cas uniforme.

Pour KCAPA comme pour BPCD, l'objectif est de mettre en place un petit nombre de groupes de poids borné et de taille petite. Dans BPCD, le diamètre maximal d'un groupe est fixé, et le nombre de groupes doit être minimisé, tandis que dans KCAPA, le nombre de groupes est préalablement fixé et correspond à un ensemble de centres à sélectionner ; puis le rayon maximal (attention, pas le diamètre) doit être minimisé. La notion de centre n'a pas de sens dans une solution de BPCD.

Dans [KS00], des algorithmes d'approximation sont présentés pour KCAPAUNIF. Par ailleurs BPCD est aussi proche d'une variante du K-centre original [HS85] (sans capacités) dans laquelle la distance intra-groupe doit être minimisée [Gon85] (*i.e.* la distance maximale entre deux membres d'un même groupe) au lieu de la distance au centre du groupe.

Dans le Paragraphe 3.5 nous présentons un résultat de correspondance liant BPCD et KCAPA. En particulier nous observons que nos résultats sur le BPCD fournissent des résultats intéressants sur les variantes uniformes et non uniformes de KCAPA.

2.2.4 Comparaison avec différents problèmes de placement de serveurs

Ce paragraphe est consacré à une revue bibliographique plus générale de plusieurs problèmes qui, de près ou d'un peu plus loin, sont liés à BPCD. Il existe en effet de nombreux problèmes, outre Kcapa évoqué dans le paragraphe précédent, très proches les uns des autres et traités de manières différentes voire indépendantes. Le but de ce paragraphe est d'essayer de fournir au lecteur une vision de l'ensemble de ces problèmes et des relations qu'ils partagent. Par ailleurs, certains problèmes ont été étudiés dans des classes de graphes particulières, ou en utilisant des modèles plus réducteurs concernant les capacités ou les poids des éléments. Il pourrait être intéressant d'étudier certains de ces problèmes en utilisant le modèle dans lequel est traité un autre problème proche, pour en tirer de nouveaux résultats.

Dans le cadre des applications que nous considérons, ces problèmes se rapprochent fortement de problèmes de placement de serveurs. Pour offrir un service particulier aux utilisateurs d'un réseau, un ensemble de serveurs doit être sélectionné pour servir les requêtes d'un ensemble de clients. La demande de chaque client, qui peut ou non dépendre du client (de ses ressources, du ou des utilisateur(s) auquel il est associé, etc.) est alors affectée à un seul serveur, ou distribuée sur plusieurs serveurs, selon le cas. Généralement chaque serveur ne peut gérer qu'une quantité fixée de demandes, se quantifiant soit en terme de nombre de clients, soit en terme de ressources totales requises par l'ensemble des demandes qui lui sont affectées. Chaque serveur peut alors, selon le cas, posséder une capacité propre, dépendant de ses ressources (capacité de stockage, bande passante,...), ou une capacité fixée qui est commune à tous les serveurs sélectionnés. Le coût d'installation d'un serveur peut également être pris en compte, soit en considérant qu'on transforme un noeud qui était client en serveur, soit par l'insertion d'un nouveau noeud dans le réseau, qui fait office de serveur. Le coût peut dépendre des ressources associées au serveur en question, mais également par exemple de son emplacement dans le réseau, ainsi généralement ce coût d'installation est propre à chaque serveur. Enfin, l'affectation d'un client à un serveur peut également être coûteuse. On peut tout d'abord penser à une modélisation via ce coût de problèmes de latences entre un client et le serveur auquel il est affecté, qu'on cherche à minimiser, mais il peut aussi s'agir de politiques d'échanges d'informations strictes entre fournisseurs d'accès à Internet engendrant des coûts pour certaines associations clients-serveurs plus élevées que pour d'autres, considérations qui sortent du cadre théorique qui nous intéresse.

Une analogie plus concrète avec la plupart des problèmes décrits ci-dessous peut être trouvée dans des problèmes de placements de déchetteries ou d'entrepôts : par exemple, considérons qu'une déchetterie peut traiter un volume maximal de déchets par jour, et que chaque ville produit quotidiennement une quantité propre de déchets. On cherche à construire un ensemble de déchetteries de façon à obtenir un bon équilibrage entre le nombre de déchetteries nécessaires et la longueur et/ou le coût des trajets que vont devoir effectuer chaque jour les camions poubelles pour desservir chacune des villes. Les variantes peuvent s'opérer sur la prise en compte ou non du coût de construction d'une déchetterie, la méthode de prise en compte des trajets à effectuer par les camions poubelles, les variations sur la taille des villes et la quantité de déchets produite par chacune d'entre elles, les possibilités d'avoir plusieurs types de déchetterie de capacités de traitement différentes, etc.

Nous essayons, pour chaque problème présenté, d'en détailler le lien avec BPCD, ou avec sa version centrée, présentée dans le Paragraphe 2.2.2.

Dans la suite, nous qualifions les demandes des clients (leur poids, mais dans le contexte de

ces problèmes, on parle plutôt de demande) de *sécables* ou de *insécables*, et les capacités des serveurs de *dures* ou *souples* :

- La demande d'un client est dite sécable quand elle peut être servie par différents serveurs. Elle est insécable quand elle doit être prise en charge intégralement par un même serveur. Considérons un problème où les capacités des clients sont distinctes, et dans lequel l'objectif est de les affecter, sans notion de distance, à un ensemble de serveurs de capacités fixées. Si les demandes des clients sont sécables, chaque serveur se voit affecter une quantité de demande égale à sa capacité. Ceci se fait de façon simple en coupant la demande du dernier client affecté à chaque serveur, et en l'affectant à un autre. A la fin, un serveur peut ne pas avoir exploité sa capacité dans son intégralité. Si la demande est insécable, chaque client doit être affecté à un serveur unique, et on retombe ainsi sur le problème de Bin Packing classique. Ainsi un problème avec demandes insécables est plus difficile qu'un problème avec demandes sécables.
- La capacité cap(s) d'un serveur est dite "souple" quand plusieurs instances d'un même serveur peuvent être utilisées, chaque utilisation du serveur offrant une capacité cap(s), comme si ce serveur n'était utilisé qu'une seule fois. En particulier son coût d'ouverture, s'il est considéré, n'est payé qu'une fois pour toutes les instances utilisées. Elle est dite "dure" quand une seule instance de chaque serveur peut être utilisée. Cette possibilité d'utiliser plusieurs instances d'un même serveur sans surcoût vient des difficultés à résoudre ce genre de problèmes avec des capacités "dures" : lorsque des capacités "souples" sont utilisées, des techniques de programmation linéaire classique produisent des résultats intéressants (cf. [CW99] par exemple), mais lorsque les capacités sont "dures", l'écart entre la plus petite solution fractionnelle, difficile à trouver, et la meilleure solution entière qu'on puisse obtenir avec ce genre de techniques peut être très important. Ainsi étudier un problème avec des capacités souples sur les serveurs est plus simple qu'étudier le même problème avec des capacités dures. Dans les études que nous présentons dans ce manuscrit, les problèmes étudiés utilisent des capacités dures sur les serveurs. Comme ce modèle nous semble plus réaliste et que nous sommes parvenus à obtenir des résultats intéressants dans ce cadre, nous ne nous sommes pas intéressés à l'étude de ces problèmes avec des capacités souples.

Commençons par définir l'ensemble des problèmes auxquels nous nous intéressons :

Définition 2.7 (*Facility Location Problem*)

Etant donnés :
- *un ensemble de clients j possédant une demande sécable d_j,*
- *un ensemble de serveurs i possédant une capacité u_i, ainsi qu'un coût d'ouverture f_i,*
- *un coût d'affectation de chaque client j à chaque serveur i, c_{ij}.*

trouver un ensemble de serveurs O à utiliser, et une affectation x des demandes des clients aux serveurs de façon à ce que chaque demande soit satisfaite, et qu'aucune capacité ne soit violée. La mesure à minimiser est le coût total $C = \sum_{i \in O}(f_i + \sum_j c_{ij}x(i,j))$, où $x(i,j)$ est le montant de d_j affecté à i.

Dans la suite, les cas étudiés sont les cas où la fonction de coût utilisée est une distance. Cette restriction vient simplement du fait que c'est le seul cas pour lequel il est possible d'obtenir des résultats intéressants. En particulier les cas où le coût d'une affectation est proportionnel à la distance entre le client et le serveur (ou une ville et la déchetterie qui l'a à sa charge) est inclus dans cette restriction.

Ce problème est étudié avec des capacités dures sur les serveurs, aussi bien par Pàl *et al.*

dans [PTW01] que par Korupolu *et al.* dans [KPR00] et par Shmoys *et al.* dans [STA97].

Définition 2.8 (*Capacitated k-median Problem [KPR00]*)

Etant donnés :
- *un ensemble de clients N,*
- *un ensemble de serveurs potentiels $F \subseteq N$,*
- *un nombre entier k de serveurs à utiliser,*
- *une demande d_j pour chaque client $j \in N$,*
- *une borne M sur le nombre de clients que peut gérer chaque serveur,*
- *un coût d'affectation de chaque client j à chaque serveur i, c_{ij}, où l'ensemble des coûts est une distance,*

trouver un ensemble O d'au plus k serveurs à ouvrir, et une association x des demandes des clients aux serveurs de façon à ce que chaque demande soit satisfaite, et qu'aucune capacité ne soit violée. La mesure à minimiser est le coût total $C = \sum_{i \in O} \sum_j c_{ij} x(i,j)$, où $x(i,j)$ est le montant de d_j affecté à i.

Les résultats connus pour ce problème nécessitent une augmentation de ressources sur le nombre de serveurs utilisés. Ainsi ils se trouvent sous la forme de (α, β)-algorithmes d'approximation, utilisant au plus αk serveurs, et assurant un facteur d'approximation de β par rapport à l'optimal utilisant k serveurs. Les capacités des serveurs sont dures.

Définition 2.9 (*Capacitated Domination Problem [KLL09]*)

Etant donné un graphe $G = (V, E)$, avec :
- *une demande d_j à chaque sommet $j \in V$,*
- *une capacité c_j à chaque sommet $j \in V$,*

trouver un ensemble $D \subseteq V$ et une association des demandes aux sommets de D de façon à ce que chaque demande d'un client soit satisfaite par un serveur qui est son voisin dans G, et qu'aucune capacité ne soit violée. La mesure à minimiser est la cardinalité de D, dans la version où les capacités des sommets sont dures (une seule occurence de chaque sommet), la somme des occurences de chaque sommet dans la version souple (possibilité de mettre plusieurs images du même sommet dans D).

Remarque 2.10

La version avec demandes insécables signifie que toute la demande d'un sommet doit être satisfaite par les copies d'un même sommet, mais pas nécessairement par une unique copie d'un même sommet (sinon on aurait, localement à chaque serveur, à résoudre le problème de Bin Packing *classique).*

Définition 2.11 (*Capacitated Dominating Sets [KM07]*)

Etant donné un graphe $G = (V, E)$, dans lequel chaque sommet $v \in V$ ne peut couvrir qu'un nombre $cap(v)$ de sommets voisins, trouver un ensemble $S \subseteq V$ de cardinalité minimale tel que chaque sommet de V soit associé à un sommet voisin de S sans violer les capacités de ceux-ci. Les capacités, ici, sont dures, et les demandes insécables.

Définition 2.12 (*Capacitated Set Cover Problem [Chu06]*)

Etant donnés :
- *un ensemble de clients j à couvrir,*
- *un ensemble de serveurs i possédant une capacité u_i, ainsi qu'un coût d'ouverture f_i,*

> – *un graphe $G = (V, E)$ d'interconnexion entre clients et serveurs : un client ne peut être couvert que par un serveur dont il est voisin dans G,*
>
> *trouver un ensemble de serveurs de coût minimal (i.e. pour lequel la somme des coûts d'ouverture est minimale), et une affectation des clients aux serveurs sélectionnés, sachant que chaque serveur i ne peut couvrir que u_i clients. Les capacités sont dures et les demandes insécables.*

Essayons à présent de présenter de façon claire les différents résultats concernant ces problèmes. Nous avertissons le lecteur qu'il est compliqué d'extraire simplement une hiérarchie de difficulté de ces problèmes, et que les résultats obtenus pour les différents problèmes utilisent des modèles de capacités et de demandes différents les uns des autres (entrainant de possibles confusions, donc). Le tableau de la Figure 2.6 décrit les résultats obtenus pour trois des problèmes définis ci-dessus, respectivement le *Facility Location Problem*, le *Capacitated k-median Problem* et le *Capacitated Domination Problem*. Pour chacun de ces problèmes nous présentons les résultats connus en précisant de quels types sont les capacités des serveurs considérés, et en fonction du type de demande des clients.

Dans la suite, nous ajoutons quelques commentaires concernant les résultats pour certains de ces problèmes et présentons pour certains problèmes d'autres résultats. Nous essayons également de donner des intuitions quant au lien existant entre BPCDC et ces problèmes, et entre ces problèmes eux-mêmes.

***Facility Location Problem* [PTW01] :**

Les solutions pour la version avec demandes insécables de ce problème (et même certaines solutions avec demandes sécables) s'appuient sur la même technique, en résolvant d'abord le cas sans capacités, puis en utilisant des techniques de programmation linéaire pour construire une solution en présence de serveurs à capacités restreintes. Ce type de méthode n'a pas permis d'obtenir des résultats d'approximation sans augmentation de ressources.

Il est intéressant de noter que la variante de ce problème ne prenant pas en compte les capacités des serveurs (*i.e.* un nombre non borné de clients peut être associé à chaque serveur), n'est approximable par un algorithme polynomial qu'à un facteur de $6 + \varepsilon$([CW99]).

Lien avec BPCD : Ce problème peut être vu comme une généralisation stricte de BPCDC, dans le cas où la fonction régissant les coûts d'affectation n'est pas une distance (ce qui n'est pas le cas généralement étudié !). Etant donnée une instance $\mathcal{I} = (S, d, w, d_{\max})$ de BPCDC, transformons la en une instance du *Facility Location Problem* :

- définissons les capacités des serveurs comme étant uniformes et égales à 1,
- les demandes d_j des clients sont les poids des éléments dans $\mathcal{I}$,
- les coûts d'ouverture des serveurs sont fixés à $f_i = \sum_{j \in S} d_j$ pour tout serveur i,
- les coûts d'affectation sont définis comme suit : $c_{i,j} = \infty$ si $d(i, j) > d_{\max}$, et $c_{i,j} = 1$ sinon.

Une solution optimale à la version avec demandes insécables et capacités dures du *Facility Location Problem* sur cette instance minimise la somme des coûts d'ouverture, donc le nombre de serveurs utilisés, donc le nombre de groupes créés dans l'instance de BPCDC d'origine, tout en assurant que les affectations respectent la contrainte de distance. Comme dit plus haut, la fonction de coût d'affectation n'est pas une distance, et les divers résultats sur ce problème ne peuvent donc s'appliquer simplement à BPCDC.

Notons qu'à cette restriction près, ce problème est proche d'englober tous les autres. La Figure 2.7 nous présente une instance (sur la figure ne sont pas spécifiées les valeurs des demandes ni les capacités des serveurs et leur coût d'ouverture) imageant la différence entre

	Facility Location Problem [PTW01]	*Capacitated k-median Problem* [KPR00]	*Capacitated Domination Problem* [KLL09]	
Capacités	dures	dures	souples	dures
Facteur d'approximation connu avec demandes sécables	$9 + \varepsilon$ (améliorable à $8.53 + \varepsilon$) [PTW01], capacités uniformes : $8 + \varepsilon$ [KPR00], 7 si une augmentation de ressources sur la capacité des serveurs est utilisée, *i.e.* en construisant une solution dans laquelle la capacité des serveurs est supérieure d'un facteur de $\frac{7}{2}$ par rapport aux serveurs utilisés dans la solution optimale à laquelle on se compare (Shmoys [STA97])	$(5 + 5/\varepsilon, 1 + \varepsilon)$ et $(5 + \varepsilon, 1 + 5/\varepsilon)$ [KPR00]	Δ dans un graphe quelconque (degré maximal du graphe), NP-difficile dans un arbre, algorithme pseudo-linéaire le résolvant (dans un arbre toujours), et un autre avec approximation à 3/2 en temps polynomial, présentés par Kao *et al.* dans [KLL09]	Aucun résultat connu
Facteur d'approximation connus avec demandes insécables	$16 + \varepsilon$ en utilisant une augmentation de ressources sur la capacité des serveurs d'un facteur 2 [KPR00], 9 en utilisant une augmentation de ressources sur la capacité des serveurs d'un facteur 4 [STA97]	$(5 + 5/\varepsilon, 1 + \varepsilon)$ et $(5 + \varepsilon, 1 + 5/\varepsilon)$ en utilisant une augmentation de ressources sur la capacité des serveurs de 2 (les serveurs peuvent couvrir une demande de $2M$) [KPR00]	Aucun résultat connu	Inapproximable (Chuzhoy [Chu06]), même sur les graphes bipartis. NP-difficile sur les arbres.

Fig. 2.6 – Tableau présentant les résultats connus concernant trois problèmes en utilisant différents modèles de capacités des serveurs et de demandes des clients

BPCD et le *Facility Location Problem* : étant donnée une instance de BPCD, l'instance du *Facility Location Problem* correspondante nécessite d'affecter un coût infini à deux éléments non connectés dans l'instance de BPCD (reliés par un trait pointillé). Dans ce cas la fonction de coût d'affectation des clients aux serveurs n'est plus une distance. Ainsi les résultats pour le *Facility Location Problem* lorsque la fonction de coût est une métrique ne s'étendent pas au BPCD dans son cadre général.

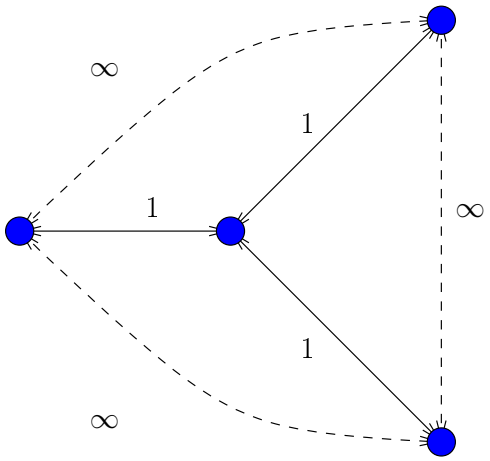


Fig. 2.7 – Une instance de BPCD (dans laquelle ne sont pas spécifiés les poids), dans laquelle deux éléments sont connectés si l'arête les liant est de poids 1, et l'instance de *Facility Location* correspondante, pour laquelle est spécifiée sur chaque arête le coût d'affectation d'une des extrémités à l'autre. Cette fonction d'affectation n'est pas une distance.

***Capacitated Domination Problem* [KLL09] :**

Kao *et al.* conjecturent dans [KLL09] qu'avec des demandes insécables et des capacités dures sur les serveurs, ce problème est également inapproximable dans les arbres [KLL09]. Ils conjecturent également dans ce cas l'existence d'un algorithme s'exécutant en temps linéaire et permettant de résoudre le problème de façon optimale si les capacités des serveurs sont uniformes.

Lien avec BPCD : Ce problème peut être vu comme une généralisation stricte de BPCDc. Etant donnée une instance $\mathcal{I} = (S, d, w, d_{\max})$, transformons la en une instance du *Capacitated Domination Problem* :
- considérons la version centrée de l'instance $\mathcal{I}$ de BPCD,
- définissons les capacités des serveurs comme étant uniformes et égales à 1,
- les demandes d_j des clients sont les poids des éléments dans $\mathcal{I}$.

Une solution optimale au *Capacitated Domination Problem* avec demandes insécables et capacités dures sur cette instance minimise le nombre de centres nécessaires, tout en assurant que les affectations respectent la contrainte de distance et les capacités, donc correspond à une solution optimale de BPCD.

Les résultats d'inapproximabilité et de NP-difficulté sur le *Capacitated Domination Problem* ne nous donnent aucune information quant à la difficulté de notre problème (BPCDc en l'occurence) car il en est une généralisation dans laquelle les capacités des serveurs ne sont pas uniformes.

Lien avec le *Facility Location Problem* : Considérons une instance du *Capacitated Domination problem*. Fixons les coûts de la manière suivante :

– les coûts d'ouverture sont fixés à $f_i = \sum_{j \in S} d_j$ pour tout serveur i,
– les coûts d'affectation sont définis comme suit : $c_{i,j} = \infty$ si $(i,j) \in E$, et $c_{i,j} = 1$ sinon.
Une solution optimale au *Facility Location Problem* sur cette instance minimise la somme des coûts d'ouverture, donc le nombre de serveurs utilisés, tout en assurant que les affectations respectent la contrainte de distance et les capacités. Notons qu'encore une fois cela nécessite d'utilise une fonction de coûts d'affectation qui n'est pas une distance. Cela est dû au fait que le passage d'un problème de minimisation du coût total induit par l'utilisation d'un ensemble de serveurs et une affectation particulière au problème de minimisation de la cardinalité de l'ensemble de serveurs utilisée se fait en utilisant une fonction de coûts qui ne peut être une distance.

***Capacitated Dominating Sets* [KM07] :**

Résultats : Non-localité : Kuhn et Moscibroda ont prouvé dans [KM07] qu'il n'existe pas d'algorithme distribué permettant d'obtenir un facteur d'approximation constant. Il existe un algorithme polynomial permettant d'obtenir un facteur d'approximation constant pour une variante de ce problème considérant que les capacités des serveurs sont uniformes, et que le graphe G possède un nombre d'indépendence borné (*i.e.* pour tout sommet $v \in V$, son voisinage à distance r possède au plus $f(r)$ voisins indépendants (pas voisin deux à deux), et $f(r)$ est polynomial en r), présenté dans [KM07].

Lien avec BPCD : Ce problème est un cas particulier du *Capacitated Domination Problem*.

Considérons la variante centrée du BPCD. En comparaison, dans le problème de *Capacitated Dominating Sets*, les capacités des serveurs peuvent varier. Par contre, les poids des sommets sont identiques, seul le nombre de sommets couvrables par un même serveur change. Même si ces deux problèmes ont l'air très proches, il ne semble pas que l'un soit une généralisation directe de l'autre.

***Capacitated Set Cover Problem* [Chu06] :**

Résultats : Ce problème est inapproximable à facteur constant, puisque le problème classique du *Set Cover* dont il est la généralisation est inapproximable à facteur constant [RS97]. Il existe un algorithme d'approximation présenté par Chuzhoy dans [Chu06] assurant une $O(\log n)$ approximation, identique à celle connue pour le *Set Cover* classique [Joh74].

Lien avec les autres problèmes : Ce problème est une forme de généralisation du *Capacitated Dominating Sets* car chaque serveur possède un coût d'ouverture propre. Il hérite donc du théorème concernant la non-localité prouvé dans [KM07] pour le *Capacitated Dominating Sets*. Par ailleurs il a le même type de lien avec BPCD que ce dernier : dans BPCD, les demandes ne sont pas uniformes, mais la capacité de chaque serveur est la même, ici c'est le contraire, les capacités diffèrent mais les demandes sont identiques.

Enfin, considérons une instance du *Capacitated Domination Problem*. Si on ajoute à chaque sommet un coût d'ouverture et qu'on essaie de minimiser le coût global au lieu de la cardinalité de l'ensemble de serveurs utilisés, puis qu'on fixe les demandes des clients comme étant toutes égales, on obtient une instance du *Capacitated Set Cover*.

2.3 Quelques notations supplémentaires

Nous utilisons par la suite la notion de graphe de compatibilité, qui nous permet, étant donnés un ensemble d'éléments dans un espace métrique et une contrainte de distance $d_{\max}$, de travailler

sur un graphe liant les éléments qui peuvent être placés dans un même groupe, sans considération pour leurs poids respectifs.

Définition 2.13 (*Graphe de Compatibilité*)

Le graphe de compatibilité $Comp(\mathcal{I}, d)$ associé à une instance $\mathcal{I} = (S, d, w, d_{\max})$ (de BCCD ou de BPCD) est le graphe $G = (S; E)$ tel que $\forall (u, v) \in S^2, (u, v) \in E \Leftrightarrow d(u, v) \leq d$.

Cette définition implique que $(u, x) \in E$ et $(x, v) \in E \Rightarrow d(u, v) \leq 2d$. Notons que si les éléments de S sont des points dans le plan euclidien, $\text{Comp}(\mathcal{I}, 1)$ est le graphe de disque-unité [CCDC90] ("Unit-Disk Graph" pour Shakespeare). La plupart du temps nous utilisons cette notation avec $d = d_{\max}$ ou un multiple de $d_{\max}$.

Dans la suite, nous notons $w(B)$ le poids total d'un ensemble $B = \{e_1, \ldots, e_n\}$ d'éléments pondérés, *i.e.* $w(B) = \sum_{e \in B} w(e)$, en étendant cette notation aux ensembles de groupes disjoints $\mathcal{B} = \{B_1, \ldots, B_k,\ \forall i \neq j, B_i \cap B_j = \emptyset\}$ en posant $w(\mathcal{B}) = \sum_{B \in \mathcal{B}} w(B)$.

Nous indiquons le diamètre d'un groupe B, soit la distance maximale entre deux éléments de B, par la notation suivante : $\text{Diam}(B) = \max_{(e_i, e_j) \in B^2} d(e_i, e_j)$. La cardinalité d'un ensemble E est notée $|E|$. Enfin, étant donné un graphe G et un élément $v \in G$, $N(v)$ est l'ensemble des voisins de v dans G.

Enfin, la notion de groupe valide est souvent utilisée :

Définition 2.14 (*Groupe valide*)

Un groupe valide dans une instance $\mathcal{I} = (S, d, w, d_{\max})$ de BCCD (resp. BPCD) est un ensemble $B \subseteq S$ de diamètre inférieur à $d_{\max}$ et tel que $w(B) \geq 1$ (resp. $w(B) \leq 1$).

2.4 Variantes uniformes

Il peut être intéressant de se pencher sur des versions uniformes des problèmes BCCD et BPCD, notamment car elles se rapprochent plus de problèmes déjà étudiés comme le K-centre avec capacités, comme nous l'avons évoqué dans le Paragraphe 2.2.3.

On peut définir ces variantes comme suit : la variante uniforme du problème de BCCD (resp. de BPCD) est la restriction de BCCD aux instances $\mathcal{I} = (S, d, w, d_{\max})$ dans lesquelles la fonction de poids w associe à chaque élément de S un poids identique. Cette restriction correspond alors au problème de trouver le nombre maximum (resp. minimum) de groupes à construire de telle sorte que chaque groupe contienne au moins (resp. au plus) un certain nombre d'éléments, en respectant toujours la contrainte de distance au sein de chaque groupe.

On peut identifier des variantes uniformes particulières de BCCD et BPCD :

- Si tous les éléments sont de poids 1, une solution optimale triviale pour BCCD comme pour BPCD est de mettre chaque élément seul dans un groupe.
- Si tous les éléments sont de poids 1/2, les deux problèmes se ramènent au même problème de couplage maximum dans le graphe de compatibilité de l'instance $\mathcal{I} = (S, d, w, d_{\max})$ de BCCD ou de BPCD considérée.

  Définition 2.15 (*Couplage Maximum*)

  Etant donné un graphe $G = (V, E)$, trouver un ensemble d'arêtes indépendantes (i.e. n'ayant

> *pas d'extrémité en commun) de cardinalité maximale.*

En effet, une solution optimale de BCCD maximise le nombre de groupes contenant deux éléments voisins dans le graphe de compatibilité, ce qui est également une solution optimale au problème de couplage maximum sur ce graphe, et une solution optimale de BPCD a également un nombre maximum de groupes contenant deux éléments, tous les autres en contenant un seul. En effet si ce n'était pas le cas, une solution construisant plus de groupes contenant deux éléments aurait besoin de construire moins de groupes globalement, puisqu'il y aurait moins d'éléments restants à mettre seul dans un groupe.
De la même façon, dans l'autre sens, la donnée d'une solution optimale du couplage maximum sur le graphe de compatibilité d'une instance $\mathcal{I}$ de BCCD comme de BPCD permet la construction d'une solution optimale pour chacun de ces problèmes. Comme Edmonds a proposé un algorithme désormais bien connu [Edm65], amélioré par Even et Kariv [EK75] s'exécutant en $O(n^{2.5})$ pour résoudre le problème de couplage maximum dans un graphe, BCCD et BPCD peuvent être résolus en temps polynomial dans ce cas.

– Si tous les éléments sont de poids nul, BCCD n'a pas de solution, et BPCD est réduit au problème de partition en cliques de cardinalité minimale dans le graphe de compatibilité de l'instance $\mathcal{I} = (S, d, w, d_{\max})$ de BPCD considérée. Nous évoquons cette dernière réduction dans le Paragraphe 3.2.2 concernant l'inapproximabilité de BPCD.

2.5 Conclusion partielle

Dans ce chapitre nous avons formellement défini les problèmes BCCD et BPCD qui nous intéressent dans ce manuscrit. Nous avons ensuite examiné tout un ensemble de problèmes de placements de ressources plus particulièrement liés à BPCD, sans en être équivalent.

Nous avons également évoqué la NP-difficulté de BCCD et de BPCD de par leurs liens avec les problèmes NP-difficile classiques que sont Bin Covering et Bin Packing. Dans l'introduction, nous avons évoqué l'utilisation d'un procédé d'augmentation de ressources qui pourrait permettre de relâcher les contraintes de ces problèmes pour en tirer des résultats intéressants. Ainsi dans la suite de ce document nous examinons quelles approximations on peut espérer tirer, pour chacun de ces problèmes, de l'utilisation de ce procédé, et quelles approximations nous parvenons effectivement à obtenir.

Enfin nous étudions ces problèmes dans des espaces métriques particuliers qui sont utilisés pour modéliser les réseaux de grande échelle qui nous intéressent, et dans lesquels nous obtenons de meilleurs résultats théoriques.

Chapitre 3

Travaux avec augmentation de ressources dans des espaces métriques généraux

3.1 Introduction du concept d'augmentation de ressources

Comme les deux problèmes étudiés, Bin Covering *avec Contrainte de Distance* (BCCD) et Bin Packing *avec Contrainte de Distance* (BPCD), sont inapproximables à facteur constant, nous avons dans la suite recours à un procédé d'augmentation de ressources (cf. [CW02, EvS07] pour des exemples d'utilisation de ce genre de techniques). Plus précisément, nous comparons le nombre de groupes obtenus lors de l'exécution d'un algorithme fonctionnant en temps polynomial et autorisé à construire des groupes de diamètre au plus $\beta d_{\max}$, pour $\beta > 1$, au nombre optimal de groupes de diamètre au plus $d_{\max}$.

Nous allons alors réutiliser le même type de notations que celles utilisées dans [KPR00] par Korupolu *et al.* pour divers problèmes de placements de facilités (*facility location*). A savoir que considérant un problème d'optimisation donné, contenant une variable A à optimiser, et une variable B fixée (la contrainte de distance pour nous, mais on peut aussi penser à des contraintes de capacités, au nombre maximal de serveurs dont on peut disposer, etc.), nous dirons qu'un algorithme $\mathcal{A}$ utilisant une augmentation de ressources de β est un (α, β)-*algorithme d'approximation* pour ce problème si il fonctionne en temps polynomial et s'autorise à relâcher la contrainte sur la variable B en travaillant à la place avec la valeur βB, tout en assurant un facteur d'approximation de α sur la variable A par rapport à une solution optimale contrainte à utiliser la valeur B.

En particulier, un (α, β)-algorithme d'approximation polynomiale pour BCCD s'exécute en temps polynomial et construit des groupes de diamètre au plus $\beta d_{\max}$. Le nombre de groupes qu'il construit est au moins $\alpha|\textsc{Opt}_{BCCD}|$, avec $\alpha \leq 1$, où $|\textsc{Opt}_{BCCD}|$ est le nombre de groupes d'une solution optimale contrainte à construire des groupes de diamètre $d_{\max}$.

En ce qui concerne BPCD, on parle d'(α, β)-algorithme d'approximation polynomiale pour BPCD à propos d'un algorithme s'exécutant en temps polynomial et construisant des groupes de diamètre au plus $\beta d_{\max}$. Le nombre de groupes qu'il construit est au plus $\alpha|\textsc{Opt}_{BPCD}|$, avec

$\alpha \geq 1$, où $|\text{OPT}_{BPCD}|$ est le nombre de groupes d'une solution optimale contrainte à construire des groupes de diamètre $d_{\max}$.

Nous dirons alors que BCCD (resp. BPCD) n'est pas (α, β)-approximable, pour certaines valeurs de α et de β, s'il n'existe pas d'(α, β)-algorithme d'approximation s'exécutant en temps polynomial pour BCCD (resp. BPCD) pour ces valeurs de α et β, à moins que $P = NP$.

Ce type d'augmentation de ressources sur le diamètre des groupes construits, en plus d'être efficace, semble réaliste au vu des applications considérées.

En effet, BCCD se place dans un contexte où l'objectif est la construction de groupes de noeuds dans un réseau dont la puissance agrégée (ou la capacité de stockage) doit dépasser une certaine valeur pour être capable de traiter une tâche. L'objectif est de maximiser le nombre de groupes de travail ainsi créés, pour ainsi maximiser le nombre de tâches indépendantes qui peuvent être traitées en parallèle. La contrainte sur la capacité des groupes est très forte, pour qu'ils soient utiles, tandis que la contrainte sur la latence maximale séparant deux hôtes d'un même groupe est plus faible, et a pour but principal de signifier qu'on souhaite que deux noeuds au sein d'un même groupe ne soient pas trop loin l'un de l'autre pour réduire les pertes de temps liées aux communications nécessaires au sein d'un même groupe. Parmi les trois contraintes liées à BCCD que sont le nombre de groupes à construire, la puissance de chaque groupe et le diamètre de chaque groupe, la contrainte sur le diamètre des groupes est bien la plus naturelle à relâcher.

BPCD quant à lui se rapproche beaucoup de problèmes de placements de serveurs. Dans ce type de problèmes, l'énoncé courant est qu'on dispose d'un nombre fixe de serveurs qu'on souhaite placer de la manière la plus adéquate dans le réseau de façon à minimiser la distance maximale entre un serveur et un noeud client du réseau. Ainsi il existe une contrainte forte sur le nombre de serveurs. Dans BPCD nous choisissons d'ailleurs d'en faire la variable à optimiser, pour minimiser le nombre de serveurs nécessaires. Lorsqu'une contrainte de capacités sur le nombre de clients que peut manipuler chaque serveur est ajoutée, il s'agit également d'une contrainte forte : un serveur ne peut pas gérer plus de connexions que sa capacité le lui permet (sa bande passante par exemple). Par contre, la distance d'un client à un serveur est moins importante, et on peut se permettre de connecter un client à un serveur un peu plus loin sans rendre inefficace l'association clients-serveurs : les requêtes du client mettent un peu plus de temps à être transmises au serveur. Dans des contextes où les volumes de données sont de plus en plus gros, la bande passante du serveur devient d'ailleurs vite un paramètre beaucoup plus important que la latence entre le client et le serveur. Ainsi de la même façon que pour BCCD, dans le contexte d'applications concernant BPCD, parmi les trois contraintes que sont le nombre de groupes à construire, la puissance de chaque serveur et la contrainte de distance entre clients et serveurs, la contrainte de distance semble la plus naturelle à relâcher.

On peut voir cette façon de fixer un des paramètres (ici le diamètre) et d'optimiser l'autre (ici le nombre de groupes) comme un moyen de contourner le fait que ces problèmes, BCCD comme BPCD, soient des problèmes d'optimisation multi-critères : on commence par identifier les contraintes dures (le poids de chaque groupe), puis parmi celles restantes, on identifie celle qu'il convient le mieux d'optimiser, selon notre problème (ici nous optimisons le nombre de groupes, mais dans le cadre du K-centre défini dans le Paragraphe 2.2.3, c'est le nombre de groupes qui est fixé et la distance maximale d'un membre d'un groupe à son centre qui est optimisée).

Nous démontrons par la suite que même avec une augmentation de ressources de $(2 - \varepsilon)$, les deux problèmes, BCCD et BPCD, restent inapproximables à facteur constant. Nous démontrons par ailleurs qu'il n'existe pas d'algorithme assurant une approximation de BCCD (resp. de BPCD)

à un facteur strictement supérieur à 1/2 (resp. inférieur à 3/2) en temps polynomial, et ce quelle que soit l'augmentation de ressources qu'on s'autorise à utiliser, à moins que $P = NP$. Ces deux derniers résultats sont valables quels que soient les espaces métriques considérés.

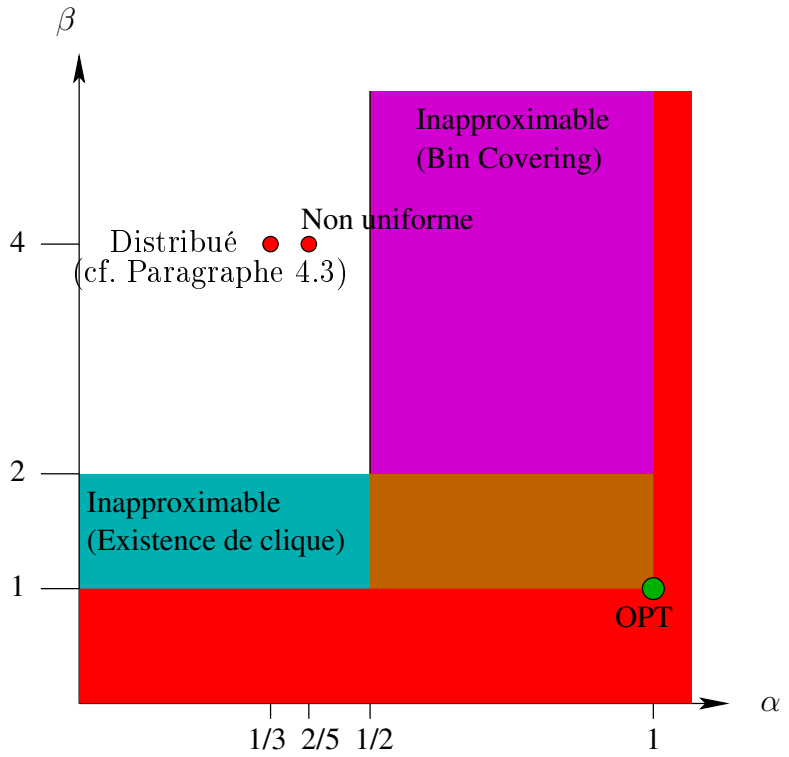


FIG. 3.1 – Résumé des résultats concernant BCCD dans un espace métrique général. Dans le Chapitre 4 nous présentons le $(\frac{1}{3}, 4)$-algorithme distribué indiqué ici, fonctionnant dans un espace métrique particulier.

Nous présentons, dans le Paragraphe 3.3, un $(2/5, 4)$-algorithme d'approximation centralisé pour BCCD, qui fonctionne dans n'importe quel espace métrique. Parallèlement, nous proposons dans le Paragraphe 3.4 un $(7/3, 2)$-algorithme d'approximation pour BPCD.

La Figure 3.1 résume les résultats d'inapproximabilité et d'approximation que nous obtenons pour BCCD dans ce chapitre. Les parties colorées correspondent aux couples (α, β) pour lesquels nous prouvons dans le Paragraphe 3.2.1 que BCCD n'est pas (α, β)-approximable. Est indiqué sur cette figure le $(\frac{2}{5}, 4)$-algorithme d'approximation présenté dans la Paragraphe 3.3, ainsi que le $(\frac{1}{3}, 4)$-algorithme d'approximation que nous présentons dans le Paragraphe 4.3, qui s'applique au cas où les éléments sont placés dans l'espace métrique $\mathbb{R}^D$ muni de la norme L_∞ pour définir les distances.

La Figure 3.2 résume les résultats d'inapproximabilité et d'approximation que nous obtenons pour BPCD dans ce chapitre. Les parties colorées correspondent aux couples (α, β) pour lesquels nous prouvons dans le Paragraphe 3.2.2 que BPCD n'est pas (α, β)-approximable. Est indiqué sur cette figure le $(\frac{7}{3}, 2)$-algorithme d'approximation présenté dans le Paragraphe 3.4, ainsi que le $(2, 2)$-algorithme d'approximation pour la variante uniforme de BPCD, tiré du résultat pour BPCD non uniforme. Comme nous le détaillons dans le Paragraphe 3.2.2, les résultats d'inapproximabilité de BPCD ne s'appliquent pas tous à la variante uniforme de BPCD. Ainsi dans la

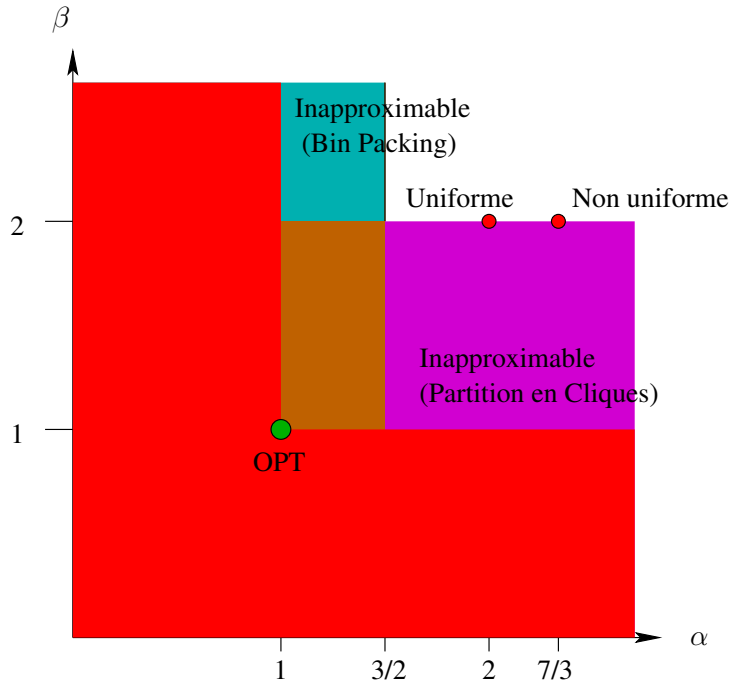


FIG. 3.2 – Résumé des résultats concernant BPCD dans un espace métrique général. On indique le $(2, 2)$-algorithme d'approximation pour la variante uniforme de BPCD bien que les résultats d'inapproximabilité de BPCD ne s'étendent pas tous à cette variante.

Figure 3.2, si l'on considère l'inapproximabilité de la variante uniforme, il ne faut pas considérer comme colorée la partie turquoise.

3.2 Inapproximabilité en dépit de l'utilisation d'augmentation de ressources

Dans ce paragraphe, nous allons présenter plusieurs résultats concernant l'inapproximabilité des deux problèmes considérés, BCCD et BPCD. Nous montrons que ces deux problèmes sont inapproximables à un facteur constant, même en utilisant une augmentation de ressources de $(2 - \varepsilon)$ sur le diamètre des groupes. Par ailleurs, nous démontrons aussi que BCCD n'est pas (α, β)-approximable pour $\alpha > 1/2$, quelle que soit la valeur de $\beta \geq 1$, et, de même, que BPCD n'est pas (α, β)-approximable pour $\alpha < 3/2$, quelle que soit la valeur de $\beta \geq 1$, et ce dans n'importe quel espace métrique.

3.2.1 Inapproximabilité de BCCD

Nous allons montrer dans ce paragraphe que BCCD est inapproximable à un quelconque facteur dépendant de $|S|$ même en utilisant une augmentation de ressources de $(2 - \varepsilon)$, via une réduction au problème d'existence d'une clique. Par ailleurs, nous démontrons que BCCD n'est pas (α, β)-approximable pour $\alpha > 1/2$, quelle que soit la valeur de $\beta \geq 1$, et ce dans n'importe quel espace métrique.

Définition 3.1 (*Problème d'existence d'une clique*)

Etant donnés un graphe $G = (V, E)$ et un entier L, G contient-il une clique de taille L, i.e. un sous-graphe complet $(V', E'), V' \subseteq V, E' \subseteq E$ tel que $|V'| = L$?

Theorème 3.2

$\forall \varepsilon > 0, \forall \alpha > 0$, BCCD n'est pas $(\alpha, (2 - \varepsilon))$-approximable.

Démonstration : Soient un graphe $G = (V, E)$ et L un entier inférieur à $|V|$. Considérons l'instance $\mathcal{I}(G) = (S, d, w, 1)$, dans laquelle chaque sommet de V correspond à un élément de S, dont le poids est $\frac{1}{L}$. Définissons la fonction de distance d comme la distance dans le graphe G. $\mathcal{I}(G)$ peut être construite en un temps polynomial à partir de G, et la taille de $\mathcal{I}(G)$ est bien polynomiale en la taille de l'instance originale.

Dans $\mathcal{I}(G)$, trouver un groupe valide, *i.e.* de poids supérieur à 1 et de diamètre inférieur à $d_{\max}= 1$ est aussi difficile que de trouver une clique de taille L dans G. Comme décider si une telle clique existe est NP-difficile, décider s'il est possible de construire 0 ou 1 groupe est aussi NP-difficile dans $\mathcal{I}(G)$. En d'autres termes, s'il existait un algorithme d'approximation fonctionnant en temps polynomial, il serait capable, quel que soit son facteur d'approximation, de résoudre le problème de décision de l'existence d'une clique. ■

Enfin, bien qu'il existe un schéma d'approximation asymptotique pour le problème de Bin Covering, il n'existe pas d'algorithme polynomial permettant d'approcher le nombre optimal de groupes dans une instance donnée à un facteur supérieur à 1/2, quand la taille de l'instance est bornée. Ceci se montre via une réduction depuis le problème de 2-partition [CJK01]. On peut en déduire le Théorème 3.3.

Theorème 3.3

$\forall\ \beta \geq 1$ et $\forall\ 1/2 < \alpha \leq 1$, BCCD n'est pas (α, β)-approximable, et ce quel que soit l'espace métrique utilisé dans les instances de BCCD.

Démonstration : En ignorant la contrainte de distance pour BCCD, la donnée d'un (α, β)-algorithme d'approximation pour BCCD avec $\alpha \geq 1/2$ est un algorithme d'approximation pour Bin Covering assurant un facteur d'approximation de α. Cette preuve se conclut en utilisant le résultat d'inapproximabilité connu pour Bin Covering classique. ■

Corollaire 3.4

En particulier, il n'existe pas de famille d'algorithmes $\{\mathcal{A}_\varepsilon\}_{\varepsilon>0}$ telle que, pour $\varepsilon > 0$ fixé, quand la taille d'une instance tend vers l'infini, $\mathcal{A}_\varepsilon$ soit un $(1 - \varepsilon, \beta)$-algorithme d'approximation pour BCCD, quelle que soit l'augmentation de ressources $\beta > 1$ utilisée.

Démonstration : Soit β le facteur d'augmentation de ressources utilisé. Considérons une instance $\mathcal{I} = (S, d, w, d_{\max})$ de BCCD dans laquelle le nombre de groupes d'une solution optimale est bornée. Construisons l'instance $\mathcal{I}' = (S^m, d, w, d_{\max})$ consistant en la reproduction de l'instance $\mathcal{I}$ en un ensemble d'instances $\mathcal{I}_1, \ldots \mathcal{I}_m$ de telle sorte que deux éléments de deux instances $\mathcal{I}_i$ et $\mathcal{I}_j$, $j \neq i$ soient à distance strictement supérieure à $\beta d_{\max}$ l'un de l'autre. L'augmentation de ressources dans cette instance n'est d'aucune utilité en dehors de chaque reproduction de l'instance $\mathcal{I}$.

Si m tend vers l'infini, le nombre de groupes d'une solution optimale sur $\mathcal{I}'$ tend vers l'infini, et pourtant le théorème précédent continue de s'appliquer à chaque des reproductions de l'instance $\mathcal{I}$. Ainsi, même quand le nombre de groupes d'une solution optimale tend vers l'infini, il n'existe pas d'(α, β)-algorithme d'approximation avec $\alpha \geq 1/2$ pour BCCD, quelle que soit l'augmentation de ressources β utilisée. ■

On peut enfin se poser la question suivante : la variante uniforme, dans laquelle tous les éléménts ont un poids identique, est-elle plus simple à résoudre ? En remarquant que la réduction utilisée pour la preuve du Théorème 3.2 utilise des instances de poids uniforme, on a le Corollaire 3.5.

Corollaire 3.5

$\forall \varepsilon > 0$, et $\forall \alpha > 0$, BCCD uniforme n'est pas $(\alpha, (2-\varepsilon))$-approximable.

Le Théorème 3.3 s'appuie fortement sur la non-uniformité des poids, puisque sa preuve utilise des résultats provenant de Bin Covering, et ne s'étend donc pas à la variante uniforme du BCCD.

3.2.2 Inapproximabilité de BPCD

Comme prouvé dans le Paragraphe 2.2.1, BPCD est dur à approcher mieux qu'à un facteur $|S|^{1/7-\delta}$ pour n'importe quel $\delta > 0$. Nous présentons ci-dessous ce résultat en l'étendant à une inapproximabilité similaire en dépit de l'utilisation d'une augmentation de ressources de $(2-\varepsilon)$ sur le diamètre des groupes construits.

Theorème 3.6

$\forall \varepsilon > 0, \forall \delta > 0$ et $\forall 0 < \alpha \leq |S|^{1/7-\delta}$, BPCD n'est pas $(\alpha, (2-\varepsilon))$-approximable.

Démonstration : Considérons une réduction depuis le problème de BPaC vers BPCD. Dans un graphe de conflits, les distances entre les éléments ont des valeurs entières, impliquant que dans l'instance de BPCD correspondante, le diamètre de n'importe quel ensemble d'éléments est entier. Ainsi, pour chaque groupe B construit sur une instance de BPCD, si le diamètre de B est inférieur ou égal à $(2-\varepsilon)$, alors c'est au plus 1. L'utilisation d'une augmentation de ressources de $(2-\varepsilon)$ n'aide donc pas à approximer BPCD. ■

De la même façon que pour Bin Covering, bien qu'il existe un schéma d'approximation asymptotique pour le problème de Bin Packing, il n'existe pas d'algorithme polynomial permettant d'approcher le nombre optimal de groupes dans une instance donnée à un facteur inférieur à 3/2 (sinon on saurait résoudre le problème de 2-partition et décider si 2 groupes ou 3 sont nécessaires), quand la taille de l'instance est bornée. On peut déduire de ceci le Théorème 3.7.

Theorème 3.7

$\forall\ \beta \geq 1$ et $\forall\ 1 \leq \alpha < 3/2$, BPCD n'est pas (α, β)-approximable, et ce quel que soit l'espace métrique utilisé dans les instances de BCCD.

Démonstration : La preuve est similaire à celle du Théorème 3.3 : en ignorant la contrainte de distance pour BPCD, on se ramène au problème de Bin Packing classique. Ainsi la donnée d'un (α, β)-algorithme d'approximation pour BPCD avec $\alpha \leq 3/2$ est un algorithme d'approximation pour Bin Packing assurant un facteur d'approximation de α, ce qui, conjointement avec le résultat d'inapproximabilité connu pour Bin Packing classique, permet de conclure la preuve. ■

Corollaire 3.8

En particulier, il n'existe pas de famille d'algorithmes $\{\mathcal{A}_\varepsilon\}_{\varepsilon>0}$ telle que, pour $\varepsilon > 0$ fixé, quand la taille d'une instance tend vers l'infini, $\mathcal{A}_\varepsilon$ soit un $(1-\varepsilon, \beta)$-algorithme d'approximation pour BPCD, quelle que soit l'augmentation de ressources $\beta > 1$ utilisée.

Démonstration : Soit β le facteur d'augmentation de ressources utilisé. Considérons une instance $\mathcal{I} = (S, d, w, d_{\max})$ de BPCD dans laquelle le nombre de groupes d'une solution optimale est bornée. Construisons l'instance $\mathcal{I}' = (S^m, d, w, d_{\max})$ consistant en la reproduction de l'instance $\mathcal{I}$ en un ensemble d'instances $\mathcal{I}_1, \ldots \mathcal{I}_m$ de telle sorte que deux éléments de deux instances $\mathcal{I}_i$ et $\mathcal{I}_j$, $j \neq i$ soient à distance strictement supérieure à $\beta d_{\max}$ l'un de l'autre. L'augmentation de ressources dans cette instance n'est d'aucune utilité en dehors de chaque reproduction de l'instance $\mathcal{I}$.

Si m tend vers l'infini, le nombre de groupes d'une solution optimale sur $\mathcal{I}'$ tend vers l'infini, et pourtant le théorème précédent continue de s'appliquer à chaque des reproductions de l'instance $\mathcal{I}$. Ainsi même quand le nombre de groupes d'une solution optimale tend vers l'infini, il n'existe pas d'(α, β)-algorithme d'approximation avec $\alpha \leq 3/2$ pour BPCD, quelle que soit l'augmentation de ressources β utilisée. ■

Une question naturelle se pose alors : la variante uniforme est-elle plus simple à résoudre ? Le Théorème 3.7 s'appuie fortement sur la non-uniformité des poids, puisque sa preuve utilise des résultats provenant de Bin Packing, et ne s'étend pas à la variante uniforme du BPCD. Par contre il est possible de démontrer un résultat équivalent au Théorème 3.6 en utilisant une réduction depuis le problème de partition d'un graphe en cliques. Ce résultat s'étend bien sûr au BPCD non uniforme, et donc le Théorème 3.10 contient le résultat présenté dans le Théorème 3.6.

Définition 3.9 (*Partition Minimale en Cliques*)

Etant donné un graphe $G = (V, E)$, trouver une partition de V en sous-ensembles disjoints $V_1, \ldots V_K$ de V de cardinalité minimale K, telle que $\forall i \leq K$, V_i soit une clique, i.e. un sous-graphe complet de G.

Une instance du problème de partition minimale en cliques, ainsi qu'une solution optimale sur cette instance, sont proposées dans la Figure 3.3.

Paz et Moran [P+81] ont montré que ce problème est équivalent au problème de coloration, et, d'après Bellare *et al.* [BGS95], est donc dur à approximer mieux qu'à un facteur $|V|^{1/7-\delta}$ pour n'importe quel $\delta > 0$.

Theorème 3.10

$\forall \varepsilon > 0, \forall \delta > 0$, et $\forall\ 0 < \alpha \leq |S|^{1/7-\delta}$, BPCD uniforme n'est pas $(\alpha, (2-\varepsilon))$-approximable.

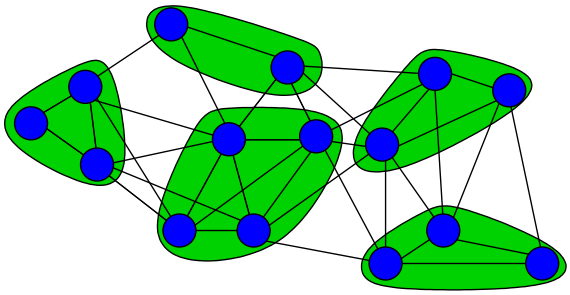


FIG. 3.3 – Un graphe sur lequel on cherche une partition minimale en cliques : une partition optimale est de cardinalité 5.

Démonstration : Considérons une réduction depuis le problème de partition en cliques. Soit $G = (V, E)$ un graphe, on construit l'instance $\mathcal{I} = (V, d, w, d_{\max})$, en définissant la fonction de distance d comme étant la distance dans le graphe G, en fixant $d_{\max}$ à 1, et en définissant la fonction de poids w de telle sorte que $\forall v \in V, w(v) = \frac{1}{|V|+1}$, où $|V|$ est le nombre de sommets du graphe G. Ainsi, même en mettant tous les éléments dans un même groupe, le poids de 1 n'est jamais atteint. Etant donnée une partition en cliques de cardinalité minimale, on peut construire une solution optimale à l'instance correspondante de BPCD de poids uniforme, en mettant dans chaque groupe les éléments appartenant à une même clique. Réciproquement, dans une solution de BPCD, les éléments d'un même groupe forment une clique dans le graphe G. Ainsi une solution optimale à l'un des problèmes est également une solution à l'autre. Comme dans un graphe, le diamètre d'un ensemble de points est entier, une augmentation de ressources de $(2 - \varepsilon)$, quelque soit $\varepsilon > 0$, n'aiderait pas à résoudre le problème du BPCD, sinon on résoudrait aussi celui de la partition en cliques. ■

Dans la suite de ce chapitre nous travaillons sur BCCD et BPCD en utilisant une augmentation de ressources suffisante pour obtenir des algorithmes centralisés assurant des facteurs d'approximation constants, dans des espaces métriques quelconques. Plus précisément, dans le Paragraphe 3.3 nous présentons un $(2/5, 4)$-algorithme d'approximation centralisé pour BCCD, et dans le Paragraphe 3.4 nous présentons un $(7/3, 2)$-algorithme d'approximation centralisé pour BPCD.

3.3 (2/5,4)-algorithme d'approximation centralisé pour BCCD

Dans ce paragraphe nous présentons un $(\frac{2}{5}, 4)$-algorithme centralisé d'approximation pour BCCD.

3.3.1 Algorithme et preuve du facteur d'approximation

Dans ce paragraphe nous présentons l'Algorithme 1 qui s'exécute en temps polynomial en produisant une $(\frac{2}{5}, 4)$ approximation pour le problème de BCCD. Cet algorithme utilise un algorithme glouton de BIN COVERING, l'algorithme *Next-Fit*, décrit dans le Paragraphe 2.1.1, et qui assure un facteur d'approximation de $\frac{1}{2}$ pour BIN COVERING classique.

Rappelons qu'étant donné un graphe $G = (V, E)$, pour $v \in V$, $N(v) = \{u \in V, (v, u) \in E\}$ désigne le voisinage de v. L'algorithme que nous allons présenter utilise le graphe de compatibilité associé à une instance $\mathcal{I} = (S, d, w, d_{\max})$ de BCCD. Comme nous l'avons défini dans le Paragraphe 2.3, ce graphe est composé des éléments de S, reliés par une arête si la distance les séparant est inférieure à $d_{\max}$.

Définition 3.11 (*Graphe de Compatibilité*)

Le graphe de compatibilité Comp($\mathcal{I}, d$) associé à une instance $\mathcal{I} = (S, d, w, d_{\max})$ (de BCCD ou de BPCD) est le graphe $G = (S; E)$ tel que $\forall (u, v) \in S^2, (u, v) \in E \Leftrightarrow d(u, v) \leq d$.

Dans la suite nous travaillons sur des solutions économes :

Définition 3.12 (*Solution économe*)

Une solution économe à une instance $\mathcal{I} = (S, d, w, d_{\max})$ de BCCD est une solution dans laquelle chaque groupe de poids $1 + w$ ne contient que des éléments de poids strictement supérieur à w.

L'Algorithme 1 s'exécute en deux phases : dans la première phase, il construit de façon économe des groupes de diamètre $2d_{\max}$ de telle sorte qu'au terme de cette phase il n'existe plus de groupe valide de diamètre $d_{\max}$ à partir des éléments non utilisés. Dans une seconde phase, l'Algorithme 1 construit des groupes de diamètre $4d_{\max}$ de telle sorte qu'au terme de son exécution il n'existe plus de groupe valide de diamètre $3d_{\max}$.

Pour prouver le facteur d'approximation de cet algorithme, nous utilisons la notion de zone étendue d'un groupe, illustrée dans la Figure 3.4(a).

Définition 3.13 (*Zone étendue*)

Etant donnés une instance $\mathcal{I} = (S, d, w, d_{\max})$ de BCCD et un groupe B d'une solution valide sur cette instance, on définit la zone étendue de B comme l'ensemble des éléments $e \in S$ tels que $d(e; B) \leq d_{max}$, où $d(e; B) = min_{u \in B} d(e, u)$.

La zone étendue d'un groupe de diamètre $d_{\max}$ est de diamètre $3d_{\max}$.

Intuitivement, la seconde phase sert à construire des groupes dans la zone étendue des groupes construits durant la première phase, de façon à ne pas perdre trop de poids par rapport à une solution optimale. En effet nous prouvons que tout élément utilisé par une solution optimale est soit utilisée par un groupe construit durant la première phase, soit dans la zone étendue d'un tel groupe. Une fois que les éléments de chaque zone étendue ont été utilisés, le poids restant dans chacune de ces zones est inférieur au poids nécessaire à la construction d'un groupe, ce qui permet alors d'obtenir le facteur d'approximation annoncé.

Un exemple d'exécution est présenté dans la Figure 3.5[3]. Durant la Phase 1, des éléments sont tour à tour sélectionnés (et apparaissent alors en blanc dans l'exemple) pour exécuter l'algorithme *Next-Fit* dans leur voisinage à distance 1, délimité par une enveloppe en pointillés, de diamètre au plus $2d_{\max}$. Ces éléments sélectionnés ne font pas nécessairement partie du ou des groupes construits (en rouge) durant cette phase. Rappelons que durant cette phase, les voisinages de chacun des éléments sont examinés, mais par souci de clarté, nous ne représentons en blanc que

[3]Le lecteur attentif (pointilleux) a remarqué que les poids des éléments ne sont pas parfaitement identiques à ceux de leurs homologues des exemples précédents, par souci de simplicité et dans l'idée de travailler sur le même graphe de compatibilité.

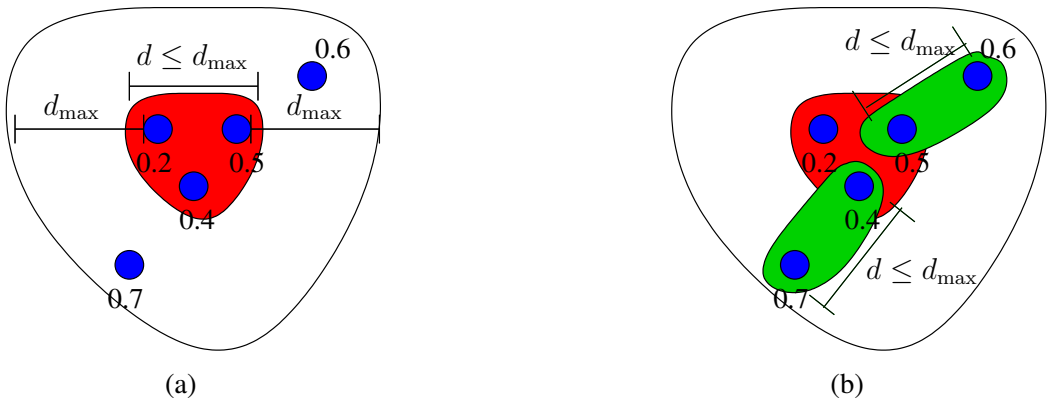


FIG. 3.4 – (a) La zone étendue d'un groupe valide dans le plan euclidien, de diamètre inférieur à $3d_{\max}$. (b) Tout groupe d'une solution optimale intersecte un groupe de Phase 1, et est donc inclus dans sa zone étendue.

Algorithme 1 $(\frac{2}{5}, 4)$-algorithme d'approximation glouton pour BCCD

$U \leftarrow S$ // Eléments qui n'appartiennent à aucun groupe

Phase 1 :

$G \leftarrow \text{Comp}(\mathcal{I}, d_{\max})$
pour chaque élément e de S **faire**
  $C \leftarrow U \cap \{e \cup N(e)\}$
  appliquer l'algorithme *Next-Fit* sur C
  pour chaque groupe B construit **faire**
    tant que il existe $e \in B$ tel que $w(B) - w(e) \geq 1$ **faire**
      supprimer e de B
    fin tant que
  fin pour
  supprimer de U les éléments appartenant à un des groupes construits.
fin pour

Phase 2 :

appliquer la Phase 1 sur $G' = \text{Comp}(\mathcal{I}, 2d_{\max})$.

les éléments dans le voisinage desquels un groupe est effectivement construit. Notons bien que les enveloppes pointillées ne forment pas nécessairement une partition des éléments de l'instance.

Dans la Phase 2, les éléments grisés sont tour à tour sélectionnés pour faire de même dans leur voisinage à distance 2 (également délimité par des pointillés) et construire des groupes de diamètre au plus $4d_{\max}$.

Theorème 3.14

L'Algorithme 1 est un $(\frac{2}{5}, 4)$-algorithme d'approximation pour BCCD fonctionnant en temps polynomial.

Démonstration : Commençons par observer que tous les groupes construits par l'Algorithme 1 ont un diamètre d'au plus $4d_{\max}$, d'où le facteur d'augmentation de ressources de 4. En effet, chaque groupe de Phase 1 est inclus dans une boule de rayon 1 de

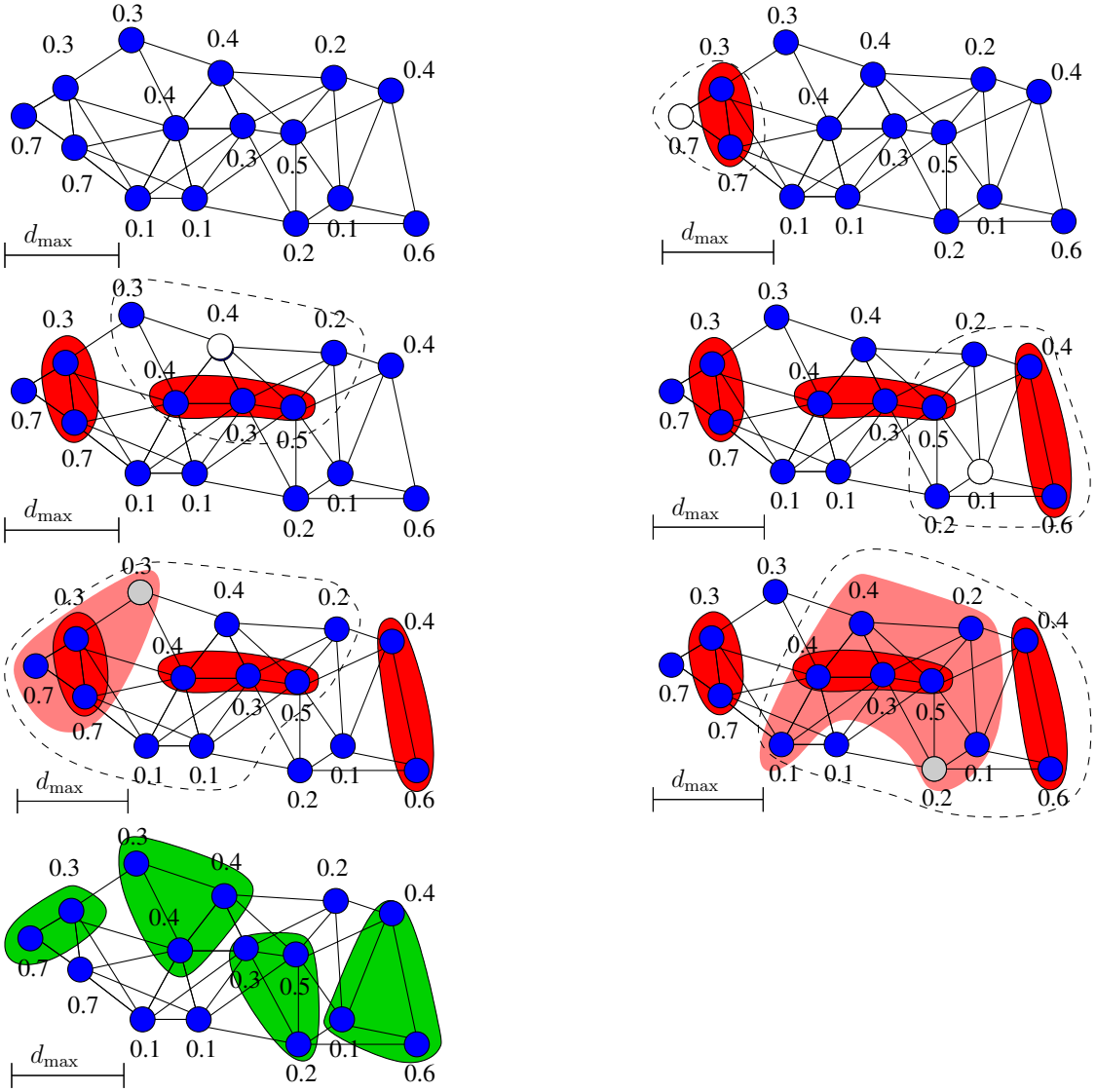


FIG. 3.5 – Exemple d'exécution de l'Algorithme 1 construisant 5 groupes sur une instance dans le plan euclidien sur laquelle la solution optimale, illustrée dans la dernière figure, construit 4 groupes. Durant la Phase 1 (4 premières figures) des éléments sont tour à tour sélectionnés (blanc) pour construire des groupes (rouges foncés) dans leur voisinage à distance 1 (enveloppe pointillée). Durant la Phase 2, les éléments gris sont sélectionnés pour faire de même dans leur voisinage à distance 2 (enveloppe pointillée également) et construire des groupes (rouges clairs) de plus grand diamètre.

$\mathrm{Comp}(\mathcal{I}, d_{\max})$, centrée en un élément e de S utilisé pour construire l'ensemble C à chaque étape. Chaque groupe de Phase 1 est donc de diamètre au plus $2d_{\max}$. De la même façon, chaque groupe de Phase 2 est inclus dans une boule de rayon 1 de $\mathrm{Comp}(\mathcal{I}, 2d_{\max})$, et a donc un diamètre d'au plus $4d_{\max}$.

On choisit une solution optimale économe, notée OPT_{BCCD}. Une telle solution optimale existe toujours, puisqu'il suffit d'en prendre une qui n'est pas économe, et d'ôter de chaque groupe ses éléments inutiles.

On note $\mathcal{B}^{(k)}$, $k = 1, 2$ l'ensemble des groupes construits durant la Phase k, avec $b_k = |\mathcal{B}^{(k)}|$.

Dans un premier temps, prouvons le lemme suivant :

Lemme 3.15

Tout groupe $B_{\text{OPT}_{BCCD}}$ de OPT_{BCCD} intersecte au moins un groupe de $\mathcal{B}^{(1)}$, et est inclus dans sa zone étendue.

Démonstration : Quand la Phase 1 se termine, aucun groupe valide ne peut être formé avec les éléments de U restants. Ceci signifie qu'au moins un élément de chaque groupe $B_{\text{OPT}_{BCCD}}$ est dans un groupe B de $\mathcal{B}^{(1)}$. En conséquence, chaque groupe $B_{\text{OPT}_{BCCD}}$ est inclus dans la zone étendue d'un groupe B de $\mathcal{B}^{(1)}$, puisque tous ses éléments sont à distance au plus $d_{\max}$ d'un tel groupe B. ■

La Figure 3.4 représente un groupe de $\mathcal{B}^{(1)}$, sa zone étendue, et deux groupes optimaux l'intersectant, se trouvant donc inclus dans sa zone étendue.

Dans la suite, partitionnons les groupes $B_i^{(1)}$, $1 \leq i \leq b_1$ de $\mathcal{B}^{(1)}$ en deux sous-ensembles $\mathcal{K}$ et $\mathcal{M}$, en fonction du nombre d'éléments de chaque groupe :

$$\begin{cases} \mathcal{K} = \{B_i^{(1)} \in \mathcal{B}^{(1)},\ \ |B_i^{(1)}| = 2\},\ \ k = |\mathcal{K}| \text{ et } \text{OPT}_K \text{ correspond aux groupes} \\ \text{de } \text{OPT}_{BCCD} \text{ qui intersectent des groupes de } \mathcal{K}, \\ \mathcal{M} = \mathcal{B}^{(1)} \setminus \mathcal{K},\ m = b_1 - k \text{ et } \text{OPT}_M \text{ correspond aux groupes de} \\ \text{OPT}_{BCCD} \setminus \text{OPT}_K \text{ qui intersectent des groupes de } \mathcal{M}. \end{cases}$$

Rappelons qu'aucun groupe ne contient qu'un unique élément, car on ne tient pas compte des éléments de poids supérieur à 1 (voir Paragraphe 2.1.2). Comme tous les groupes de OPT_{BCCD} intersectent un groupe de Phase 1, on a :

$$|\text{OPT}_{BCCD}| \leq |\text{OPT}_K| + |\text{OPT}_M|. \tag{3.1}$$

Comme tous les groupes de $\mathcal{K}$ ont exactement deux éléments, et que les groupes de la solution optimale sont disjoints, alors chaque groupe de $\mathcal{K}$ intersecte au plus deux groupes de OPT_K. D'où

$$|\text{OPT}_K| \leq 2k. \tag{3.2}$$

Dans le but de majorer la cardinalité de OPT_M, on introduit la notion de poids perdu :

Définition 3.16 (*Poids perdu*)

Etant donnée instance $\mathcal{I} = (S, d, w, d_{\max})$ de BCCD, et une solution construite sur $\mathcal{I}$ par l'Algorithme 1, w_{perdu} est le poids total des éléments qui n'appartiennent à aucun groupe de cette solution, mais qui appartiennent à un groupe de OPT_M.

On utilise pour la suite de la preuve le Lemme 3.17.

Lemme 3.17

$w_{perdu} < m$

Démonstration : Dans la zone étendue de chaque groupe de $\mathcal{M}$, le poids total des éléments qui n'appartiennent à aucun des groupes construits par l'Algorithme 1 est strictement inférieur à 1. En effet, si ce n'était pas le cas, un autre groupe

aurait été construit en Phase 2. Comme tout élément appartenant à un groupe de OPT_M se trouve dans la zone étendue d'un des groupes de $\mathcal{M}$, $w_{\text{perdu}} < m$. ■

Par ailleurs on a l'équation suivante régissant la cardinalité de OPT_M :

$$|\text{OPT}_M| \leq w(\text{OPT}_M) \leq w(\mathcal{M}) + w(\mathcal{B}^{(2)}) + w_{\text{perdu}} \tag{3.3}$$

On utilise le Lemme 3.18 pour majorer le poids de chaque groupe en fonction de sa cardinalité.

Lemme 3.18

$\forall B \in \mathcal{M}, w(B) < 1 + \frac{1}{|B|-1}$

Démonstration : Posons $w(B) = 1 + w$. Dans l'Algorithme 1, tant que le poids de B est supérieur à 1 et qu'on peut en ôter un élément, on l'en supprime. Ainsi tous les éléments de B ont un poids strictement supérieur à w, et donc $w(B) > w \times |B|$ et $1 + w > w \times |B|$. Finalement, $w < \frac{1}{|B|-1}$. ■

Ainsi, comme tout groupe B de $\mathcal{M}$ contient au moins trois éléments, $w(B) < \frac{3}{2} (= 1 + \frac{1}{3-1})$. Ainsi $w(\mathcal{M}) < \frac{3}{2}m$. Comme chaque groupe de $\mathcal{B}_2$ a un poids strictement inférieur à 2, on a :

$$\begin{array}{rlll} & |\text{OPT}_M| & < \frac{3}{2}m + 2b_2 + m & = \frac{5}{2}m + 2b_2 \\ \text{et donc,} & |\text{OPT}_{BCCD}| & < \frac{5}{2}m + 2b_2 + 2k & < \frac{5}{2}(b_1 + b_2), \end{array}$$

ce qui conclut la preuve du Théorème 3.14. ■

3.3.2 Augmentation de ressources sur le poids des groupes

Dans ce paragraphe nous nous interrogeons sur ce que pourrait apporter à l'Algorithme 1 une augmentation de ressources sur le poids des groupes, *i.e.* si on autorisait cet algorithme à construire des groupes de poids supérieur à γ, où $\gamma \leq 1$, en comparant les solutions construites à une solution optimale contrainte à construire des groupes de poids supérieur à 1. En particulier, nous parlons de l'algorithme $NF(\gamma)$ comme la variante de l'algorithme *Next-Fit* construisant des groupes de poids γ.

Tout d'abord remarquons que les résultats d'inapproximabilité démontrés pour BCCD ne s'étendent pas trivialement au cadre où une augmentation de ressources sur le poids des groupes est utilisée. Aucun résultat d'inapproximabilité n'est donc connu en utilisant une telle augmentation de ressources. Dans la suite nous observons que l'utilisation d'une telle augmentation de ressources permet effectivement d'obtenir de meilleurs résultats, mais nous n'avons aucune garantie que BCCD ne devient pas facile à étudier en utilisant ce procédé.

Par exemple, dans Bin Covering classique, si on autorise les groupes à n'avoir qu'un poids de 1/2, l'algorithme $NF(1/2)$ construit un nombre de groupes supérieur au nombre de groupes d'une solution optimale contrainte à un poids 1. Notons $|NF(1/2)|$ le nombre de groupes de poids supérieur à 1/2 créés par $NF(1/2)$. La cardinalité d'une solution optimale, $|OPT|$, est inférieure au poids total des éléments de l'instance, w_{total}. Chaque groupe B construit par $NF(1/2)$ va

avoir un poids $w(B)$ tel que $1/2 \leq w(B) < 1$, sauf le dernier groupe construit, dont le poids n'atteint peut être pas $1/2$. On a alors :

$$w_{total} = \sum_{B \in NF(1/2)} w(B) < |NF(1/2)| + 1/2, \tag{3.4}$$

soit $|NF(1/2)| \geq |OPT| - 1/2$, ce qui, comme le nombre de groupes est entier, nous amène à conclure qu'avec une augmentation de ressources de 2 sur le poids des groupes, une solution à Bin Covering classique de cardinalité égale à celle de la solution optimale peut être trouvée en temps polynomial.

Pour un γ fixé, en modifiant l'Algorithme 1 en utilisant $NF(\gamma)$ en lieu et place de l'algorithme *Next-Fit* classique, on obtient le Théorème 3.19, illustré par une courbe dans la Figure 3.6, traçant l'augmentation de ressources γ nécessaire pour un facteur d'approximation α voulu.

Theorème 3.19

L'Algorithme 1 modifié en utilisant $NF(\gamma)$ est un algorithme d'approximation pour BCCD assurant un facteur d'approximation fonction de γ :

pour $\gamma \geq 4/5$ **:** *c'est un* $(\frac{2}{5\gamma}, 4)$*-algorithme d'approximation,*

pour $2/3 \leq \gamma \leq 4/5$ **:** *c'est un* $(\frac{1}{2}, 4)$*-algorithme d'approximation,*

pour $\gamma \leq 2/3$ **:** *c'est un* $(\frac{1}{3\gamma}, 4)$*-algorithme d'approximation.*

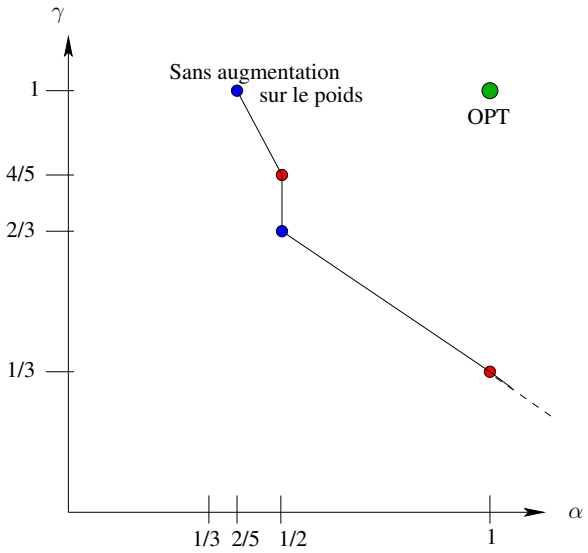


Fig. 3.6 – Schéma d'approximation de BPCD en fonction de l'augmentation de ressources sur le poids de γ utilisée. L'augmentation de ressources utilisée sur le diamètre est constante, de $\beta = 4$.

Démonstration : Soit $\mathcal{A}(\gamma)$ l'Algorithme 1 utilisant $NF(\gamma)$ pour construire des groupes.

On réutilise la preuve du facteur d'approximation obtenu en utilisant l'Algorithme 1, et notamment les notations qui y sont introduites, que nous rappelons ici :

- On note Opt_{BCCD} une solution optimale économe.
- On note $\mathcal{B}^{(k)}$, $k = 1, 2$ l'ensemble des groupes construits durant la Phase k, avec $b_k = |\mathcal{B}^{(k)}|$.

– On partitionne les groupes $B_i^{(1)}$, $1 \leq i \leq b_1$ de $\mathcal{B}^{(1)}$ en deux sous-ensembles $\mathcal{K}$ et $\mathcal{M}$, en fonction du nombre d'éléments de chaque groupe :

$$\begin{cases} \mathcal{K} = \{B_i^{(1)} \in \mathcal{B}^{(1)},\ \ |B_i^{(1)}| = 2\},\ \ k = |\mathcal{K}| \text{ et } \textsc{Opt}_K \text{ correspond aux groupes} \\ \text{de } \textsc{Opt}_{BCCD} \text{ qui intersectent des groupes de } \mathcal{K}, \\ \mathcal{M} = \mathcal{B}^{(1)} \setminus \mathcal{K},\ m = b_1 - k \text{ et } \textsc{Opt}_M \text{ correspond aux groupes de} \\ \textsc{Opt}_{BCCD} \setminus \textsc{Opt}_K \text{ qui intersectent des groupes de } \mathcal{M}. \end{cases}$$

On va avoir recours à des définitions différentes du poids perdu en fonction de la valeur de γ, que l'on va majorer différemment :

(a) : soit, comme dans la preuve précédente, on remarque que tous les groupes de $\mathcal{K}$ ont exactement deux éléments, et que les groupes de la solution optimale sont disjoints, et que donc chaque groupe de $\mathcal{K}$ intersecte au plus deux groupes de $\textsc{Opt}_K$, d'où $|\textsc{Opt}_K| \leq 2k$.

On définit alors le poids perdu w_{perdu} de la même façon que dans la preuve précédente, comme le poids total des éléments qui n'appartiennent à aucun groupe de la solution construite par $\mathcal{A}(\gamma)$, mais qui appartiennent à un groupe de $\textsc{Opt}_M$.

On reprend alors la preuve du Lemme 3.17, pour obtenir l'inégalité suivante :

$$w_{\text{perdu}} < \gamma m. \tag{3.5}$$

En effet, dans chacune des m zones étendues des groupes de $\mathcal{M}$ construits en Phase 1, il n'existe plus d'ensemble d'éléments de poids suffisant pour construire un groupe (donc, ici, de poids γ), d'où la majoration (voir la preuve du Lemme 3.17 en cas de doute).

(b) : soit on redéfinit le poids perdu w'_{perdu} comme le poids total des éléments qui n'appartiennent à aucun groupe construit par $\mathcal{A}(\gamma)$, mais qui appartiennent à un groupe de $\textsc{Opt}_{BCCD}$, et plus nécessairement de $\textsc{Opt}_M$. On va pouvoir reprendre la preuve du Lemme 3.17 et utiliser l'inégalité suivante :

$$w'_{\text{perdu}} < \gamma b_1. \tag{3.6}$$

En effet, dans chacune des b_1 zones étendues des groupes de $\mathcal{B}^{(1)}$, il n'existe plus d'ensemble d'éléments de poids suffisant pour construire un groupe (donc, ici, de poids γ), d'où la majoration.

L'équation (4.1) reste vraie, car chacun des groupes de $\textsc{Opt}_{BCCD}$ intersecte un groupe de Phase 1.

$$|\textsc{Opt}_{BCCD}| \leq |\textsc{Opt}_K| + |\textsc{Opt}_M|.$$

On peut alors obtenir plusieurs approximations, selon la valeur de γ. L'utilisation de l'algorithme $NF(\gamma)$ a pour conséquence que tous les groupes B construits par notre algorithme ont un poids $w(B)$ tel que $\gamma \leq w(B) < 2\gamma$.

L'adaptation de la preuve du Lemme 3.18 prouve directement le lemme suivant :

Lemme 3.20

$\forall B \in \mathcal{M}, w(B) < \gamma + \frac{\gamma}{|B|-1}$

Démonstration : Posons $w(B) = \gamma + w$. Tant que le poids de B est supérieur à γ et qu'un élément peut en être ôté en conservant cette propriété, $\mathcal{A}(\gamma)$ l'en supprime. Ainsi tous les éléments de B ont un poids strictement supérieur à w, et donc $w(B) > w \times |B|$ et $\gamma + w > w \times |B|$. Finalement, $w < \frac{\gamma}{|B|-1}$. ■

Comme tout groupe B de $\mathcal{M}$ contient au moins trois éléments, par définition, alors $w(B) \leq \frac{3\gamma}{2} (= \gamma + \frac{\gamma}{3-1})$. Ainsi $w(\mathcal{M}) < \frac{3\gamma}{2} m$.

Notre étude peut à présent être divisée en trois cas, selon la valeur de γ :

$\gamma \geq 4/5$ **:** On utilise l'équation suivante régissant la cardinalité de OPT_M :

$$|\text{OPT}_M| \leq w(\text{OPT}_M) \leq w(\mathcal{M}) + w(\mathcal{B}^{(2)}) + w_{\text{perdu}} \tag{3.7}$$

Enfin, en la combinant à l'inégalité $|\text{OPT}_K| \leq 2k$ et à l'inégalité (3.5) on obtient :

$$\begin{aligned} |\text{OPT}_{BCCD}| &\leq 2k + w(\mathcal{M}) + w(\mathcal{B}^{(2)}) + w_{\text{perdu}} \\ |\text{OPT}_{BCCD}| &< 2k + \frac{3\gamma}{2} m + 2\gamma b_2 + \gamma m < \frac{5\gamma}{2}(b_1 + b_2) \end{aligned}$$

car $\gamma \geq 4/5 \Leftrightarrow \frac{5\gamma}{2} \geq 2$. Ainsi le facteur d'approximation est de $\frac{2}{5\gamma}$.

$2/3 \leq \gamma \leq 4/5$ **:** on utilise les mêmes équations que dans le cas précédent, sauf qu'on a $\gamma \leq 4/5 \Leftrightarrow \frac{5\gamma}{2} \leq 2$, et donc la dernière équation devient :

$$|\text{OPT}_{BCCD}| < 2(b_1 + b_2). \tag{3.8}$$

ce qui donne un facteur d'approximation de $\frac{1}{2}$,

$\gamma \leq 2/3$ **:** On se contente de majorer le poids de chaque groupe par 2γ, et en utilisant l'inégalité (3.6) on obtient :

$$|\text{OPT}_{BCCD}| \leq w(\mathcal{K}) + w(\mathcal{M}) + w(\mathcal{B}^{(2)}) + w'_{\text{perdu}} \tag{3.9}$$

$$|\text{OPT}_{BCCD}| \leq 2\gamma k + 2\gamma m + 2b_2 + b_1 \tag{3.10}$$

$$|\text{OPT}_{BCCD}| \leq 3\gamma(b_1 + b_2) \tag{3.11}$$

et le facteur d'approximation est de $\frac{1}{3\gamma}$. ■

3.4 Un $(7/3, 2)$-algorithme d'approximation pour BPCD

Dans ce paragraphe nous présentons un $(\frac{7}{3}, 2)$-algorithme d'approximation centralisé pour BPCD. Cet algorithme est une adaptation de l'algorithme proposé par Epstein et Levin dans leurs travaux sur BPAC(cf. [EL08]), utilisant l'algorithme *First-Fit-Decreasing* évoqué dans le Paragraphe 2.2.1 pour la construction des groupes.

3.4.1 Outils pour la preuve du facteur d'approximation utilisant les résultats d'Epstein et Levin

Les lemmes que nous présentons ici sont très fortement inspirés de ceux d'Epstein et Levin. Les preuves de ces lemmes sont très calculatoires, mais par souci de rigueur il nous a semblé nécessaire de les présenter à nouveau en les adaptant à notre cadre, ce qui requiert de très légères modifications.

Leur étude de BPaC utilise des fonctions de poids étendus. L'idée de ces fonctions est d'affecter à chaque élément x un poids étendu $e(x)$ dépendant de son poids $w(x)$ et de son emplacement dans une solution choisie. Ces poids étendus sont affectés de telle façon que le poids de la solution construite par un algorithme soit proche du poids étendu total des éléments. Pour compléter l'analyse, il est généralement nécessaire de considérer également le poids étendu total qui peut être placé dans un même groupe d'une solution optimale.

Introduisons tout d'abord la fonction de poids étendu suivante :

Définition 3.21 (*Poids étendu*)

Etant donné un élément x de poids $w(x)$, son poids étendu $e(x)$ est tel que :
- $e(x) = 1$ *si* $w(x) > \frac{1}{2}$,
- $e(x) = w(x) + \frac{1}{j(j+1)}$ *si* $w(x) \in \mathcal{I}_j =]\frac{1}{j+1}, \frac{1}{j}]$ *pour un entier* $j > 1$ *quelconque.*

Etant donnée cette fonction de poids étendus, remarquons que pour un élément $x \in \mathcal{I}_j, j \geq 2$, le rapport entre son poids étendu $e(x)$ et son poids $w(x)$ est borné comme suit : $\frac{j+2}{j+1} \leq \frac{e(x)}{w(x)} \leq \frac{j+1}{j}$.

Pour un ensemble d'éléments X, on note la somme des poids étendus des éléments de X $e(X)$, *i.e.* $e(X) = \sum_{x \in X} e(x)$. Nous présentons dans la suite le résultat d'Epstein et Levin prouvant que tout algorithme partitionnant l'ensemble d'éléments S d'une instance $\mathcal{I} = (S, d, w, d_{\max})$ de BPCD en μ sous-ensembles, et qui ensuite applique l'algorithme *First-Fit-Decreasing* sur chaque classe indépendamment, vérifie la condition suivante concernant la cardinalité d'une solution qu'il construit, dépendant du poids étendu total des éléments et de μ.

Lemme 3.22 (*D'après [EL08]*)

Soit $\mathcal{A}$ un algorithme et un sous-ensemble J d'éléments qui vont être groupés en utilisant l'algorithme First-Fit-Decreasing sans tenir compte des distances les séparant. Soit $|\mathcal{A}(J)|$ le nombre de groupes construit par cet algorithme sur J. Alors $|\mathcal{A}(J)| \leq e(J) + 1$.

Démonstration : Remarquons tout d'abord qu'en utilisant la fonction de poids étendu que nous avons défini, tout groupe contenant un élément x de poids $w(x)$ dans $]\frac{1}{2}, 1]$ a un poids étendu total supérieur ou égal à 1.

Remarquons également que le poids étendu d'un élément dans $\mathcal{I}_j =]\frac{1}{j+1}, \frac{1}{j}]$ est supérieur ou égal à $\frac{1}{j+1} + \frac{1}{j(j+1)} = \frac{1}{j}$. Ainsi, tout groupe contenant j éléments de poids (non étendu) dans l'intervalle $]\frac{1}{j+1}, \frac{1}{j}]$ a un poids étendu total supérieur ou égal à 1.

On peut donc s'affranchir de la considération de tels groupes dans cette preuve et se concentrer sur les autres groupes, qu'on appelle des *groupes de transition* (si aucun tel groupe n'existe, alors la preuve est terminée).

Un groupe de transition contient uniquement des éléments de poids (non étendu) inférieur ou égal à $\frac{1}{2}$. Nous dirons qu'un groupe de transition est de *type* j (pour $j \geq 2$)

si le premier élément x ajouté à ce groupe par l'algorithme *First-Fit-Decreasing* (FFD) a un poids $w(x) \in \mathcal{I}_j$.

L'argument suivant est qu'il existe au plus un groupe de transition de chaque type. En effet, comme les éléments sont groupés en utilisant FFD, les groupes de transition sont construits de façon ordonnée, en commençant par le type le plus petit (car par l'élément le plus grand). S'il existe deux groupes de même type j, c'est qu'entre le placement des premiers éléments de chacun de ces groupes, tous les éléments possédaient également un poids dans $\mathcal{I}_j$. Ainsi, le premier groupe s'est vu attribuer j éléments dans $\mathcal{I}_j$ avant que le second groupe de transition soit commencé, et ainsi le premier groupe n'est pas un groupe de transition.

Soit k le type le plus grand parmi tous les groupes de transition utilisés (*i.e.* le groupe de transition ayant le plus petit des premiers éléments). Retirons de la solution tous les éléments x de poids $w(x) \leq \frac{1}{k+1}$. Cette suppression a pour seule conséquence de décroître le poids étendu total des éléments. Comme dit précédemment, le poids étendu de tous les éléments restants dans les groupes de transition est supérieur ou égal à $\frac{k+2}{k+1}$ multiplié par leur poids non étendu.

Soit α le poids du premier élément du dernier groupe de transition (alors $\alpha \in]\frac{1}{k+1}, \frac{1}{k}]$). Comme le dernier groupe de transition est ouvert, tous les autres groupes ont un poids total (non étendu) strictement supérieur à $1-\alpha$. Soit $i_1 < \cdots < i_t < k$ la liste ordonnée des types des groupes de transition. Considérons deux cas, qui sont $t \leq \lfloor \frac{k+2}{2} \rfloor$, et $t > \lfloor \frac{k+2}{2} \rfloor$. Dans les deux cas il est nécessaire de prouver que le poids étendu total de tous les groupes de transition est d'au moins t (puisqu'il y a $t+1$ groupes de transition).

- Dans le premier cas, si $t = 0$, c'est réglé. Supposons donc que $t \geq 1$. Le poids étendu total est strictement supérieur à

$$\begin{aligned} t(1-\alpha)\tfrac{k+2}{k+1} + \alpha + \tfrac{1}{k(k+1)} &= t\tfrac{k+2}{k+1} + \tfrac{1}{k(k+1)} - \alpha(t\tfrac{k+2}{k+1} - 1) \\ &\geq t\tfrac{k+2}{k+1} + \tfrac{1}{k(k+1)} - \tfrac{t\frac{k+2}{k+1}-1}{k} \\ &= t\tfrac{k+2}{k+1}\tfrac{k-1}{k} + \tfrac{k+2}{k(k+1)}. \end{aligned}$$

  Cette inégalité est vraie car le coefficient multiplicatif de α est négatif, et $\alpha \leq \frac{1}{k}$. Nous voulons montrer que ce poids étendu est supérieur ou égal à t, *i.e.* que $t(\frac{k^2+k-2}{k^2+k} - 1) + \frac{k+2}{k^2+k} \geq 0$. Ceci est vrai pour $t \leq \frac{k+2}{2}$.
- Considérons le second cas. La preuve du premier cas a prouvé qu'il est désormais suffisant de considérer les $f = t - \lfloor \frac{k+2}{2} \rfloor$ premiers groupes de transition, et de prouver que le poids étendu total de ces groupes est supérieur à f. Ces groupes sont de types $i_1, \ldots, i_f$. Considérons le groupe de transition de type i_{f+1}. Remarquons que $i_{f+1} \leq k - \lfloor \frac{k+2}{2} \rfloor$ car deux groupes de transition ne sont pas de même type, et $i_{f+1} \geq 3$ car $i_f \geq 2$. Soit β le poids non étendu du premier élément du groupe de transition de type i_{f+1}. Posons $m = i_{f+1}$ (ainsi $\beta \in]\frac{1}{m+1}, \frac{1}{m}]$). En considérant seulement les éléments x de poids $w(x) \in]\frac{1}{m}, \frac{1}{2}]$, tout groupe de transition parmi les f premiers a un poids non étendu total supérieur ou égal à $1-\beta$. Cependant, ils ont également un ensemble d'éléments de $]\frac{1}{k+1}, \frac{1}{2}]$ de poids non étendu total supérieur à $1-\alpha$. Ainsi le poids étendu des éléments dans chaque groupe est strictement supérieur à $(1-\beta)\frac{m+2}{m+1} + (\beta - \alpha)\frac{k+2}{k+1}$.

Nous prouvons à présent que cette valeur est toujours supérieure ou égale à 1. Cette expression est minimisée pour des valeurs maximum de α et β. Comme $\alpha \in]\frac{1}{k+1}, \frac{1}{k}]$ et $\beta \in]\frac{1}{m+1}, \frac{1}{m}]$, il faut montrer que $(1 - \frac{1}{m})\frac{m+2}{m+1} + (\frac{1}{m} - \frac{1}{k})\frac{k+2}{k+1} - 1 \geq 0$, ce qui équivaut à

$\frac{k-m}{km}\frac{k+2}{k+1} \geq \frac{2}{m(m+1)}$. Remarquons que $k - m \geq \frac{k+1}{2}$, $m + 1 \geq 4$ et ainsi

$$\frac{k-m}{km}\frac{k+2}{k+1}\frac{m(m+1)}{2} \geq \frac{k+2}{k} > 1,$$

ce qui conclut cette preuve. ■

Pour analyser l'algorithme que nous présentons, nous avons besoin de définir une autre fonction de poids étendu, qui donne des poids moins grands aux éléments de $\mathcal{I}_1 =]\frac{1}{2}, 1]$. On redéfinit ainsi le poids étendu d'un élément x de poids $w(x) \in \mathcal{I}_1$ comme $e(x) = w(x) + \frac{1}{6}$.

L'utilisation de cette nouvelle fonction de poids étendu restreint le type d'algorithmes qui vérifient des propriétés intéressantes. En particulier le lemme suivant s'applique à un type d'algorithme particulier : étant donnée une instance $\mathcal{I} = (S, d, w, d_{\max})$ de BPCD, ce type d'algorithme partitionne l'ensemble S en ν sous-ensembles, contenant chacun au plus un élément de $\mathcal{I}_1$.

Dans $\mu \leq \nu$ parmi ces ν ensembles, des groupes sont construits en utilisant FFD, sans prise en compte des distances au sein de chacun de ces ensembles. Chaque autre ensemble est placé dans un groupe et se voit affecter un poids étendu total supérieur à 1.

Lemme 3.23

Un algorithme $\mathcal{A}$ exécuté sur une instance $\mathcal{I} = (S, d, w, d_{\max})$ de BPCD, qui partitionne S en ν sous-ensembles contenant chacun au plus un élément de $\mathcal{I}_1$, et qui applique FFD sur $\mu \leq \nu$ de ces sous-ensembles, en plaçant chacun des autres ensembles dans un groupe, vérifie l'inégalité $|\mathcal{A}(\mathcal{I})| \leq e(S) + \mu$.

Démonstration : Tout d'abord remarquons que si pour chacun des ν ensembles de la partition construite par $\mathcal{A}$, le nombre de groupes $|\mathcal{A}(\mu)|$ construit par FFD sur μ est tel que $|\mathcal{A}(\mu)| \leq e(\mu) + 1$, alors le lemme est vrai.

Pour tous les ensembles μ ne contenant pas d'élément dans $\mathcal{I}_1$, cette propriété est vérifiée d'après le Lemme 3.22.

Considérons maintenant un des μ ensembles contenant un seul élément de $\mathcal{I}_1$. Cet élément est le premier à être placé dans un groupe par FFD. Dans cette preuve on considère ce groupe comme étant également un groupe de transition. Soit k le type du dernier groupe de transition comme défini dans le Lemme 3.22.

Si $k \geq 6$, le poids non étendu total des éléments dans le premier groupe de transition est supérieur ou égal à $\frac{5}{6}$ (sinon on y aurait mis le premier élément du groupe de transition de type k), et donc son poids étendu total est supérieur à $\frac{5}{6} + \frac{1}{6} \geq 1$ (car le poids étendu de l'élément le plus grand de ce groupe est sa taille plus $\frac{1}{6}$, et que pour tout élément son poids étendu est plus grand que son poids non étendu). La preuve pour les autres groupes de transition est la même que dans le Lemme 3.22.

Il reste donc 5 cas à étudier : $k = 1, 2, 3, 4, 5$. Dans le premier cas, il n'y a qu'un groupe de transition, et donc la preuve est finie. Dans le second cas, il y a deux groupes de transition. La somme des éléments des deux groupes est supérieure à 1, et donc la somme de leur poids étendu également.

Si $k = 3$, on peut supposer qu'il y a trois groupes de transition (s'il n'y en a que deux, on se ramène au cas $k = 2$). Comme dans la preuve du Lemme 3.22, on minore la somme des poids étendus des groupes qui ne sont ni le premier ni le dernier groupe de transition

par $\frac{k+2}{k+1}(1-\alpha)$. Comme $\frac{1}{k+1} \leq \alpha \leq \frac{1}{k}$, cette valeur est strictement inférieure à 1. Le poids étendu des éléments dans le premier groupe de transition est supérieur à $1-\alpha+\frac{1}{6}$, et dans le dernier groupe de transition est supérieur à $\alpha+\frac{1}{k(k+1)}$. Pour $k=3$, on obtient un poids étendu total supérieur à $1-\alpha+\frac{1}{6}+\frac{5}{4}(1-\alpha)+\alpha+\frac{1}{12} = \frac{5}{2}-\frac{5\alpha}{4} \geq \frac{25}{12} > 2$.

Pour $k=4$ il y a trois ou quatre groupes de transition, soit t ce nombre. On obtient un poids étendu total supérieur à

$$1-\alpha+\frac{1}{6}+(t-2)\frac{6}{5}(1-\alpha)+\alpha+\frac{1}{20} = -\frac{6}{5}\alpha(t-2)+\frac{6}{5}(t-2)+\frac{73}{60} \geq \frac{9}{10}(t-2)+\frac{73}{60} > t-1 \text{ pour } t \leq 4.$$

Finalement, pour $k=5$, on note t le nombre de groupes de transition (au moins 3, au plus 5) et on effectue le même type de calcul pour obtenir

$$1-\alpha+\frac{1}{6}+(t-2)\frac{7}{6}(1-\alpha)+\alpha+\frac{1}{30} = -\frac{7}{6}\alpha(t-2)+\frac{7}{6}(t-2)+\frac{6}{5} \geq \frac{14}{15}(t-2)+\frac{6}{5} > t-1 \text{ pour } t \leq 5.$$ ■

Dans la suite nous présentons un algorithme satisfaisant les spécificités nécessaires à l'application de ce Lemme 3.23. Plus précisément étant donnée une instance $\mathcal{I} = (S, d, w, d_{\max})$ de BPCD, cet algorithme partitionne l'ensemble S en ν sous-ensembles, contenant chacun au plus un élément de $\mathcal{I}_1$.

Dans $\mu \leq \nu$ parmi ces ν ensembles, des groupes sont construits en utilisant FFD, sans prise en compte des distances au sein de chacun de ces ensembles. Chaque autre ensemble est placé dans un groupe.

3.4.2 Algorithme et preuve du facteur d'approximation

Rappelons qu'on dénote par OPT_{BPAC} une solution optimale de BPAC, et $|\text{OPT}_{\text{BPAC}}|$ sa cardinalité.

Algorithme 2 $(\frac{7}{3}, 2)$-algorithme d'approximation glouton pour BPCD

1: $U \leftarrow S$ // Eléments non groupés
2: $\mathcal{C} = \text{Comp}(\mathcal{I}, d_{\max})$
3: **tant que** il existe un triangle $\{a, b, c\}$ tel que a, b et c peuvent être placés dans un même groupe, *i.e.* $w(a)+w(b)+w(c) \leq 1$, et tels que $e(a)+e(b)+e(c) > 1$ et $w(c) \leq w(b) \leq w(a) \leq \frac{1}{2}$; ou une paire d'éléments connectés (a, b) qui peut être placée dans un même groupe, *i.e.* $w(a) + w(b) \leq 1$, telle que $e(a) + e(b) > 1$ **faire**
4: choisir un de ces ensembles de poids étendu maximal, et placer tous ses éléments dans un nouveau groupe
5: supprimer de U cet ensemble d'éléments
6: **fin tant que**
7: construire une partition μ de U en utilisant l'Algorithme 3
8: appliquer l'algorithme *First-Fit-Decreasing* à chacun des $|\mu|$ ensembles induits par la partition μ de U.

L'Algorithme 3 décrit comment construire une partition μ de U en ensembles de diamètre $2d_{\max}$, telle que $|\mu| \leq \text{OPT}_{BPCD}$, en utilisant un ensemble indépendant maximal particulier, de

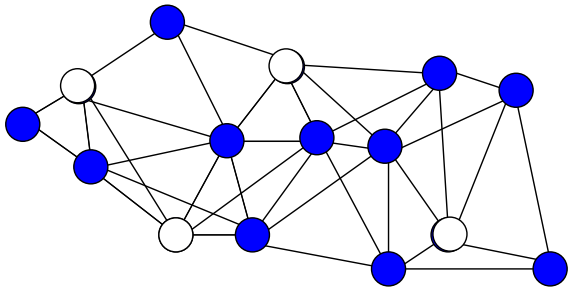


FIG. 3.7 – Un exemple d'ensemble indépendant maximal sur ce graphe (composé des éléments blancs).

façon à ce que chaque ensemble de la partition contienne au plus un élément de poids supérieur ou égal à $\frac{1}{2}$.

Dans chaque ensemble, des groupes peuvent alors être construits sans tenir compte des contraintes de distance, en utilisant l'algorithme FFD, et on se retrouve dans le cadre d'application du Lemme 3.23.

En contrepartie, certains groupes peuvent avoir un diamètre de $2d_{\max}$ au lieu de $d_{\max}$, d'où le facteur $\beta = 2$ d'augmentation de ressources, qui est nécessaire pour avoir un facteur d'approximation constant, d'après le résultat d'inapproximabilité de BPCD donné par le Théorème 3.6 (cf. Paragraphe 3.2.2).

Définition 3.24 (*Ensemble Indépendant Maximal*)

Etant donné un graphe $G = (V, E)$, un ensemble indépendant maximal $M \subseteq V$ est tel que $\forall (u, v) \in M^2, (u, v) \notin E$ et M est maximal pour l'inclusion.

Un exemple de construction d'un ensemble indépendant maximal est présenté dans la Figure 3.7. Notons qu'il n'est pas unique dans le cas général, et qu'en l'occurence il en existe un de cardinalité plus grande sur cet exemple (de cardinalité 5 pour être précis).

Une des propriétés d'un tel ensemble est que tout élément n'appartenant pas à cet ensemble possède un voisin dans cet ensemble. L'obtention d'un tel ensemble se fait de façon triviale dans n'importe quel graphe, en prenant un élément du graphe et en l'ajoutant à M, puis en le supprimant de V ainsi que ses voisins, tant que V est non vide. Ce problème est à ne pas confondre avec la recherche d'un ensemble indépendant maximum, qui, étant donné un graphe G, a pour but de trouver un ensemble indépendant de cardinalité maximale pour G. De par son équivalence au problème de trouver la clique maximale d'un graphe, il est dur à approcher en temps polynomial dans le cas général [Hås99].

Un exemple d'exécution de l'Algorithme 2 est présenté dans la Figure 3.8. Dans cet exemple, on n'exécute pas la ligne 3 de cet algorithme pour conserver un exemple clair et relativement simple. En premier lieu, on exécute l'Algorithme 3, qui commence par construire un ensemble maximal indépendant, dont les éléments sont identifiés en blanc. Ensuite les éléments n'en faisant pas partie sont rattachés à un des éléments de cet ensemble, et on obtient les ensembles roses clairs. Dans chacun de ces ensembles correspondant à chacune des composantes de la partition μ, on applique l'algorithme *First-Fit-Decreasing* pour construire des groupes de diamètre au plus $2d_{\max}$ (rouges). La solution optimale est la même que dans la Figure 2.3(b), et construit 7 groupes.

Algorithme 3 Construction d'une partition μ de S' en ensembles de diamètre au plus $2d_{\max}$

1: construire le graphe Comp'$(\mathcal{I}, d_{\max})$ comme étant le graphe Comp$(\mathcal{I}, d_{\max})$ duquel on supprime les arêtes entre deux éléments u et v de S' de poids $w(u)$ et $w(v)$ strictement supérieur à $\frac{1}{2}$
2: construire un Ensemble Indépendant Maximal μ tel que les éléments de poids supérieur à $\frac{1}{2}$ appartiennent à μ
3: associer arbitrairement chaque sommet n'appartenant pas à μ à l'un de ses voisins dans μ, puis retourner les $|\mu|$ ensembles d'éléments ainsi construits.

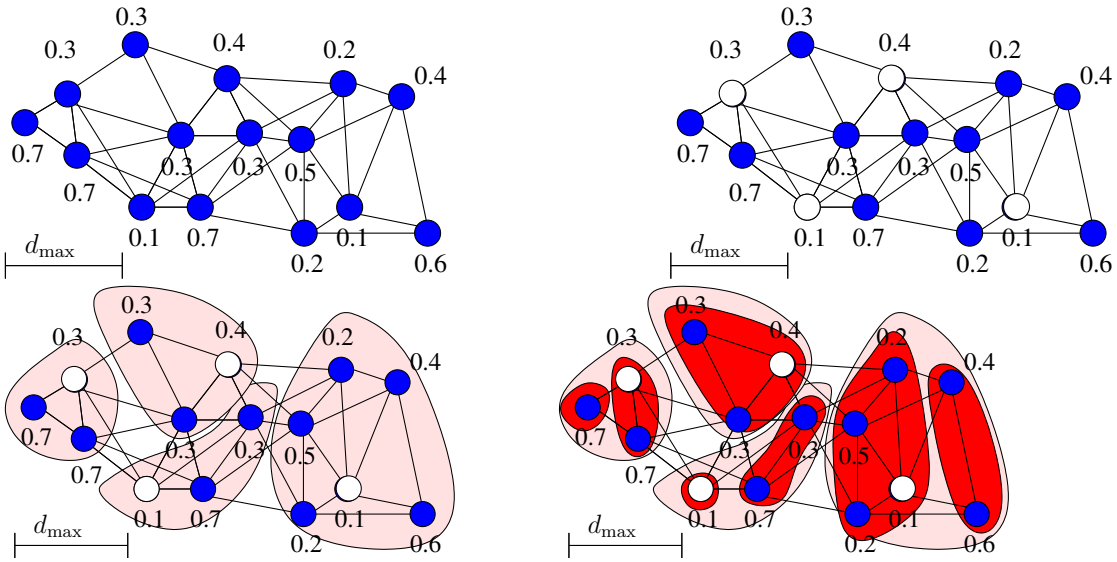


FIG. 3.8 – Exemple d'exécution de l'Algorithme 2 construisant 7 groupes sur une instance dans le plan euclidien sur laquelle la solution optimale de la Figure 2.3(b) construit également 7 groupes.

Lemme 3.25

La partition μ de S' construite par l'Algorithme 3 est telle que $|\mu| \leq \text{OPT}_{BPCD}$.

Démonstration : Deux éléments de μ ne peuvent appartenir au même groupe dans une solution optimale de BPCD. En effet, puisqu'ils appartiennent à un ensemble indépendant maximal de Comp'$(\mathcal{I}, d_{\max})$:

- soit la distance les séparant dans Comp$(\mathcal{I}, d_{\max})$ est supérieure ou égale à 2, et donc ils sont à plus de $d_{\max}$ l'un de l'autre dans $\mathcal{I}$ et ne peuvent être dans le même groupe ;
- soit ils sont tous deux de poids supérieur à $\frac{1}{2}$, et dans ce cas, un groupe les contenant tous les deux aurait un poids trop important.

D'où l'inégalité $|\mu| \leq \text{OPT}_{BPCD}$. ■

Theorème 3.26

L'Algorithme 2 est un $(\frac{7}{3}, 2)$-algorithme d'approximation s'exécutant en temps polynomial pour BPCD.

Démonstration : Fixons une solution optimale OPT_{BPCD}.

Pour prouver ce théorème il nous faut appliquer le Lemme 3.23, et en particulier obtenir une majoration du poids étendu de chacun des groupes d'une solution optimale. Cette preuve réutilise également très largement des éléments de la preuve du Théorème 3 d'Epstein et Levin dans [EL08].

Nous allons en effet prouver que pour tout ensemble d'élément E groupés dans un même groupe de OPT_{BPCD}, $e(E) \leq \frac{4}{3}$, et donc $e(S) \leq \frac{4}{3}|\text{OPT}_{BPCD}|$.

Comme par ailleurs le Lemme 3.25 nous dit que $|\mu| \leq |\text{OPT}_{BPCD}|$, l'application du Lemme 3.23 nous donne $|\mathcal{A}(\mathcal{I})| \leq (\frac{4}{3} + 1)|\text{OPT}_{BPCD}|$ ce qui conclut cette preuve.

Dans un premier temps, nous utilisons une nouvelle fonction de poids étendu, e_1, en plus de la fonction e déjà définie, qui va affecter des poids étendus aux éléments en fonction de leur situation dans la solution optimale. Cette nouvelle fonction de poids va nous permettre de traiter les éléments appartenant à des groupes construits durant la première phase (ligne 3 de l'Algorithme 2) pour pouvoir obtenir pour ces groupes un résultat similaire au Lemme 3.22 et donc appliquer la méthode que nous venons de présenter.

Pour un groupe de OPT_{BPCD} qui ne contient aucun élément de $\mathcal{I}_1$, et qui ne contient pas de triplet d'éléments de poids étendu strictement supérieur à 1 selon la fonction e, on utilise $e_1 = e$ pour définir le poids étendu des éléments d'un tel groupe. Pour un groupe contenant un élément $x \in \mathcal{I}_1$, mais qui ne contient aucun autre élément y tel que $e(x) + e(y) > 1$, on utilise à nouveau $e_1 = e$ pour chacun des éléments d'un tel groupe.

Pour un groupe ne contenant aucun élément de $\mathcal{I}_1$, mais contenant un triplet d'éléments dont la somme des poids étendus selon e est strictement supérieure à 1, soit a_1, a_2, a_3 les trois éléments tels que $e(a_1), e(a_2)$ et $e(a_3)$ sont les poids étendus les plus grands dans ce groupe, ordonnés selon leur poids étendu. Remarquons que $w(a_1) \in \mathcal{I}_2 \cup \mathcal{I}_3$, sinon la somme des poids étendus des trois éléments ne peut dépasser 1 ($(\frac{1}{4} + \frac{1}{4*5}) * 3 < 1$). On définit la valeur de réduction pour ce groupe comme

$$\Delta = \frac{e(a_1) + e(a_2) + e(a_3) - 1}{3}.$$

Pour tout élément b de ce groupe tel que $b \neq a_i$ pour $i = 1, 2, 3$ on définit $e_1(b) = e(b)$. Remarquons que dans la première phase (ligne 3) de l'Algorithme 2, au moins un des trois éléments a_1, a_2 et a_3 est supprimé sinon si tous les trois sont toujours présents, cette phase ne peut se terminer. Soit i' l'indice d'un élément a_i tel que $1 \leq i' \leq 3$ et $a_{i'}$ n'est pas supprimé après a_j pour $1 \leq j \leq 3$. On définit $e_1(a_{i'}) = e(a_{i'}) - \Delta$, et pour $i \neq i'$, $e_1(a_i) = e(a_i)$.

Pour un groupe qui contient un élément $x \in \mathcal{I}_1$ et contenant un autre élément y tel que $e(x) + e(y) > 1$, prenons y un tel élément de poids étendu maximum selon la fonction e. On définit la valeur de réduction pour ce groupe comme $\Delta = \frac{e(x)+e(y)-1}{3}$. Pour tout élément b dans un tel groupe tel que $b \neq x, y$, on définit $e_1(b) = e(b)$. Remarquons que, de la même façon que précédemment, dans la première phase (ligne 3) de l'Algorithme 2, au moins un des deux éléments x et y est supprimé. Si y n'est pas supprimé après x, on définit $e_1(y) = e(y) - \Delta$ et $e_1(x) = e(x)$, sinon $e_1(y) = e(y)$ et $e_1(x) = e(x) - \Delta$.

Considérons un groupe construit dans la première phase (ligne 3) de l'Algorithme 2. Le prochain argument est que le poids étendu total des éléments d'un tel groupe selon e_1 est supérieur à 1. Remarquons tout d'abord que le poids étendu de ces éléments selon e est strictement supérieur à 1. Ainsi, si pour tout élément a d'un tel groupe, $e_1(a) = e(a)$, la somme des poids étendus dans ce groupe est strictement supérieure à 1. Nous montrons

dans la suite qu'une possible réduction des poids étendus ne fait pas décroître la somme de ceux-ci à moins de 1. Soit B l'ensemble des éléments dans un tel groupe (construit durant la première phase), et $\Gamma = \frac{(\sum_{a\in B} e(a))-1}{3}$. Pour un élément $a \in B$ on a $e_1(a) < e(a)$ si la condition suivante est vraie. Considérons le groupe auquel appartient a dans OPT_{BPCD}. Alors une valeur $\Delta(a)$ a été calculée pour ce groupe telle que $e_1(a) = e(a) - \Delta(a)$. L'élément a est supprimé durant la première phase, et au moment de la suppression de a, il appartient à un ensemble d'éléments A de poids étendu maximal, valide pour former un groupe durant la première phase. De plus, aucun élément de A n'a déjà été supprimé au moment où a est supprimé. Ainsi au cours de la suppression des éléments dans cette première phase on a toujours $\Gamma \geq \Delta(a)$. Ainsi on a $\sum_{a\in B} e_1(a) = \sum_{a\in B}(e(a) - \Delta(a)) \geq \sum_{a\in B} e(a) - 3\Gamma = 1$. Ainsi, chacun des groupes construits durant la première phase a un poids étendu supérieur à 1. On peut donc utiliser le Lemme 3.23 puisque les poids étendus des éléments groupés en utilisant FFD sont ceux définis par la fonction e (ceux qui ont un poids différent de e sont tous dans des groupes construits durant la première phase).

Il ne nous reste donc "plus qu'à" majorer le poids étendu d'un groupe de la solution optimale. Cette majoration se fait en utilisant la fonction de poids étendu e_1.

Considérons les éléments d'un groupe $B \in \text{OPT}_{BPCD}$.

Si tous ces éléments ont un poids (non étendu) inférieur ou égal à $\frac{1}{3}$, alors pour tout élément $a \in B, e_1(a) \leq \frac{4}{3}w(a)$, et donc le poids étendu total des éléments de B est au plus $\frac{4}{3}$. Ceci couvre à la fois le cas où il n'y a pas de réduction dans les poids étendus des éléments en utilisant e_1 par rapport à e, et le cas où une telle réduction est utilisée pour certains éléments.

Considérons à présent que B contient un élément x de poids $w(x) \in \mathcal{I}_2$, mais que tous les poids étendus de B sont affectés en utilisant e (*i.e.* pour tout $a \in B, e_1(a) = e(a)$). Ceci peut arriver dans deux cas de figures.

- Si B contient un élément supplémentaire y tel que $w(y) \in \mathcal{I}_2$, alors $B = \{x, y\}$. En effet un troisième élément impliquerait l'existence d'un triplet de poids étendu total strictement supérieur à 1 et on aurait alors $e_1(x) \neq e(x)$. Ainsi, dans ce cas où $B = \{x, y\}$ on obtient un poids étendu total de $w(x) + \frac{1}{6} + w(y) + \frac{1}{6} \leq 1 + \frac{1}{3} \leq \frac{4}{3}$.
- Sinon, pour tout $y \in B\backslash\{x\}$, on peut conclure que $w(y) \in]0, \frac{1}{3}]$.
  - Si tous les éléments de $B\backslash\{x\}$ ont en fait un poids $w(y) \in]0, \frac{1}{4}]$, alors $e(y) \leq \frac{5}{4}w(y)$, et le poids non étendu total des éléments de $B\backslash\{x\}$ est au plus $1 - w(x)$, ce qui donne après recombination un poids étendu total d'au plus $w(x) + \frac{1}{6} + \frac{5}{4}(1 - w(x)) = \frac{17}{12} - \frac{w(x)}{4}$. Cette valeur est maximale quand $w(x)$ est minimale, et donc le poids étendu total des éléments de B est au plus $\frac{5}{4}$.
  - Finalement, si il existe $y \in B$ tel que $w(y) \in \mathcal{I}_3$, alors tous les éléments $z \in B\backslash\{x, y\}$ ont nécessairement un poids $w(z) \in]0, \frac{1}{6}]$, et ainsi leur poids étendu est au plus $\frac{7}{6}$ fois leur poids non étendu. Un troisième élément v de poids $w(v) \in]\frac{1}{6}, \frac{1}{4}]$ dans ce groupe impliquerait l'existence d'un triplet de poids étendu total au moins égal à $\frac{1}{2} + \frac{1}{3} + \frac{1}{5} > 1$, ce qui nous forcerait à avoir $e_1(x) < e(x)$ en contradiction avec le cas que nous considérons. Ainsi, dans ce cas, le poids étendu des éléments de B peut être majoré par $w(x)+\frac{1}{6}+w(y)+\frac{1}{12}+(1-w(x)-w(y))\frac{7}{6} = \frac{17}{12} - \frac{w(x)+w(y)}{6} \leq \frac{17}{12} - \frac{7}{72} = \frac{95}{72} < \frac{4}{3}$.

Supposons à présent que B contient un élément y de poids $w(y) \in \mathcal{I}_1$. Si $B = \{y\}$, alors le poids étendu total est inférieur ou égal à $\frac{7}{6}$. Sinon, posons $B = \{y, x_1, \ldots, x_t\}$

où $w(x_1) \geq \cdots \geq w(x_t)$. Alors, x_1 est l'élément de plus grand poids étendu selon e dans $B\backslash\{y\}$. Soit $j \geq 2$ un entier tel que $w(x_1) \in \mathcal{I}_j$. Soit

$$\Delta = \frac{e(y) + e(x_1) - 1}{3} = \frac{w(y) + \frac{1}{6} + w(x_1) + \frac{1}{j(j+1)} - 1}{3}$$

la valeur de réduction de ce groupe. On a $e(y) + e(x_1) = 3\Delta + 1$. Le poids étendu total (selon e_1) des éléments $y, x_1, \ldots, x_t$ est alors inférieur à

$$\begin{array}{rl} & \frac{j+1}{j}\left(\sum_{i=2}^{t} w(x_i)\right) + e(y) + e(x_1) - \Delta \\ \leq & \frac{j+1}{j}(1 - w(y) - w(x_1)) + 1 + \frac{2}{3}(w(y) + \frac{1}{6} + w(x_1) + \frac{1}{j(j+1)} - 1) \\ = & -\frac{(j+3)(w(y)+w(x))}{3j} + \frac{j+1}{j} + \frac{4}{9} + \frac{2}{3j(j+1)}. \end{array}$$

On utilise $w(y) \geq \frac{1}{2}$ et $w(x_1) \geq \frac{1}{j+1}$, et on obtient un poids étendu total d'au plus

$$\begin{array}{rl} & -\frac{3(j+3)^2}{18j(j+1)} + \frac{18(j+1)^2+8j(j+1)+12}{18j(j+1)} \\ = & \frac{23j^2+26j+3}{18j(j+1)} \\ = & \frac{23}{18} + \frac{1}{6j}. \end{array}$$

Si $j \geq 3$, c'est fini. Cependant, si $j = 2$, alors tous les éléments sauf y et x_1 ont un poids non étendu strictement inférieur à $\frac{1}{6}$, et alors leurs poids étendus respectifs sont d'au plus $\frac{7}{6}$ fois leurs poids non étendus. On obtient ainsi un poids étendu global d'au plus

$$\begin{array}{rl} & \frac{7}{6}(1 - w(y) - w(x_1)) + 1 + \frac{2}{3}(w(y) + \frac{1}{6} + w(x_1) + \frac{1}{6} - 1) \\ = & -\frac{w(x_1)+w(y)}{2} + \frac{31}{18} \leq -\frac{5}{12} + \frac{31}{18} \\ = & \frac{47}{36} \\ < & \frac{4}{3}. \end{array}$$

Il reste à considérer le cas où tous les éléments de B ont un poids non étendu inférieur ou égal à $\frac{1}{2}$, mais où e_1 n'a pas affecté le même poids étendu que e à tous les éléments. Un tel groupe doit contenir au moins un élément dans $\mathcal{I}_2$ (sinon ce cas a déjà été étudié, tous les éléments sont de poids non étendu inférieur à $\frac{1}{3}$). Il y a deux types de groupes dans ce cas : soit B contient deux éléments de poids non étendu dans $\mathcal{I}_2$, soit B n'en contient qu'un.

Dans le second cas, B contient nécessairement au moins un élément dans $\mathcal{I}_3$, autrement ce cas a déjà été étudié quand $e_1 = e$ pour tous les éléments (en effet dans la preuve du cas $e_1 = e$ présentée précédemment nous n'utilisions pas la supposition selon laquelle le groupe B ne se voyait appliquer aucune réduction dans la valeur du poids étendu de ses éléments, mais utilisait seulement des considérations sur les poids des éléments et leur poids étendu. Ces considérations restent valides dans le cas présent, puisque en l'occurence le poids étendu selon e_1 peut seulement être inférieur à celui selon e).

Considérons le premier cas. Soit $y_1 \in B$ et $y_2 \in B$ les éléments de poids respectivement $w(y_1) \in \mathcal{I}_2$ et $w(y_2) \in \mathcal{I}_2$, et supposons que $B = \{y_1, y_2, x_1, \ldots x_t\}$ où $w(x_1) \geq \cdots \geq w(x_t)$, et donc x_1 est un élément de poids étendu maximal selon e dans $B\backslash\{y_1, y_2\}$. Soit $j \geq 3$ un entier tel que $w(x_1) \in \mathcal{I}_j$. Soit $\Delta = \frac{e(y_1)+e(y_2)+e(x_1)-1}{3} = \frac{w(y_1)+\frac{1}{6}+w(y_2)+\frac{1}{6}+w(x_1)+\frac{1}{j(j+1)}-1}{3}$ la valeur de réduction de ce groupe. On a $e(y_1) + e(y_2) + e(x_1) = 3\Delta + 1$. Le poids étendu total (selon e_1) des éléments de B est alors au plus

$$\begin{array}{rl} & \frac{j+1}{j}\left(\sum_{i=2}^{t} w(x_i)\right) + e(y_1) + e(y_2) + e(x_1) - \Delta \\ = & \frac{j+1}{j}\left(\sum_{i=2}^{t} w(x_i)\right) + 2\Delta + 1 \\ \leq & \frac{j+1}{j}(1 - w(y_1) - w(y_2) - w(x_1)) + 1 + \frac{2}{3}(w(y_1) + \frac{1}{6} + w(y_2) + \frac{1}{6} + w(x_1) + \frac{1}{j(j+1)} - 1) \\ = & -\frac{(j+3)(w(y_1)+w(y_2)+w(x_1))}{3j} + \frac{j+1}{j} + \frac{5}{9} + \frac{2}{3j(j+1)}. \end{array}$$

On utilise $w(y_1) \geq \frac{1}{3}$ et $w(y_2) \geq \frac{1}{3}$ et $w(x_1) \geq \frac{1}{j+1}$, et on obtient un poids étendu total d'au plus

$$\begin{array}{rl} & -\frac{(j+3)(2j+5)}{9j(j+1)} + \frac{9(j+1)^2+5j(j+1)+6}{9j(j+1)} \\ = & -\frac{2j^2+11j+15}{9j(j+1)} + \frac{9j^2+18j+9+5j^2+5j+6}{9j(j+1)} \\ = & \frac{4}{3}. \end{array}$$

Considérons le second cas. Soit y_1, y_2 les éléments de poids respectivement $w(y_1) \in \mathcal{I}_2$ et $w(y_2) \in \mathcal{I}_3$, et posons $B = \{y_1, y_2, x_1, \ldots, x_t\}$ où $w(x_1) \geq \cdots \geq w(x_t)$, et donc x_1 est un élément de poids étendu maximal selon e dans $B \backslash \{y_1, y_2\}$. Soit $j \geq 3$ un entier tel que $w(x_1) \in \mathcal{I}_j$. Soit $\Delta = \frac{e(y_1)+e(y_2)+e(x_1)-1}{3} = \frac{w(y_1)+\frac{1}{6}+w(y_2)+\frac{1}{12}+w(x_1)+\frac{1}{j(j+1)}-1}{3}$ la valeur de réduction de ce groupe. On a $e(y_1)+e(y_2)+e(x_1) = 3\Delta+1$. Le poids étendu total (selon e_1) des éléments de B est alors au plus

$$\begin{array}{rl} & \frac{j+1}{j}\left(\sum_{i=2}^{t} w(x_i)\right) + e(y_1) + e(y_2) + e(x_1) - \Delta \\ \leq & \frac{j+1}{j}(1 - w(y_1) - w(y_2) - w(x_1)) + 1 + \frac{2}{3}(w(y_1) + \frac{1}{6} + w(y_2) + \frac{1}{12} + w(x_1) + \frac{1}{j(j+1)} - 1) \\ = & -\frac{(j+3)(w(y_1)+w(y_2)+w(x_1))}{3j} + \frac{j+1}{j} + \frac{1}{2} + \frac{2}{3j(j+1)}. \end{array}$$

On utilise $w(y_1) \geq \frac{1}{3}$ et $w(y_2) \geq \frac{1}{4}$ et $w(x_1) \geq \frac{1}{j+1}$, et on obtient un poids étendu total d'au plus

$$\begin{array}{rl} & -\frac{(j+3)(7j+19)}{36j(j+1)} + \frac{36(j+1)^2+18j(j+1)+24}{36j(j+1)} \\ = & -\frac{7j^2+40j+57}{36j(j+1)} + \frac{36j^2+72j+36+18j^2+18j+24}{36j(j+1)} \\ = & \frac{47}{36} + \frac{1}{12j} \\ \leq & \frac{4}{3}. \end{array}$$

Tout groupe d'une solution optimale a donc un poids étendu inférieur ou égal à $\frac{4}{3}$. On peut donc, comme nous l'avons expliqué au début de cette preuve, appliquer le Lemme 3.23 qui, combiné au Lemme 3.25 nous permet de prouver que l'Algorithme 2 assure un facteur d'approximation de $\frac{4}{3} + 1 = \frac{7}{3}$ ■

L'Algorithme 2 est donc optimal en terme d'augmentation de ressources, dans le sens où il ne peut exister d'algorithme d'approximation assurant un facteur constant en utilisant une augmentation de ressources strictement inférieure à 2 (cf. Paragraphe 3.2.2 pour plus de détails).

Dans [EL08] les auteurs utilisent, pour construire une solution à BPaC, une solution au problème d'extension de précoloration, étudié notamment dans [B+92, HT93, HT08].

Définition 3.27 (*Problème d'extension de précoloration*)

Etant donnés un graphe $G = (V, E)$, un sous-ensemble $W \subseteq V$, une coloration c' de W

> *et un entier K, existe-t-il une coloration propre (i.e. telle que deux voisins dans G n'aient pas la même couleur) de G utilisant K couleurs, et étendant la coloration c' (i.e. telle que $\forall w \in W, c(w) = c'(w)$).*

Nous dénotons par $\text{OPT}_{\text{GRAPHCOL}}$ une solution optimale au problème d'extension de précoloration, et $|\text{OPT}_{\text{GRAPHCOL}}|$ sa cardinalité. Ce problème est une généralisation du problème classique de coloration de graphe dans laquelle une contrainte additionnelle spécifie à l'avance la couleur de certains noeuds.

Dans les Figures 3.9(a) et 3.9(b) est présenté un graphe pour lequel sont proposées respectivement dans la Figure 3.9(a) et 3.9(b) une solution optimale au problème de coloration classique, et une solution au problème d'extension de précoloration, où les sommets carrés sont ceux dont on a préalablement choisi la couleur. On observe que suite au choix qui a été effectué, le nombre de couleurs nécessaires dans l'instance d'extension de précoloration, s'appuyant sur le même graphe, est plus élevé que le nombre de couleurs nécessaire au coloriage classique de ce graphe.

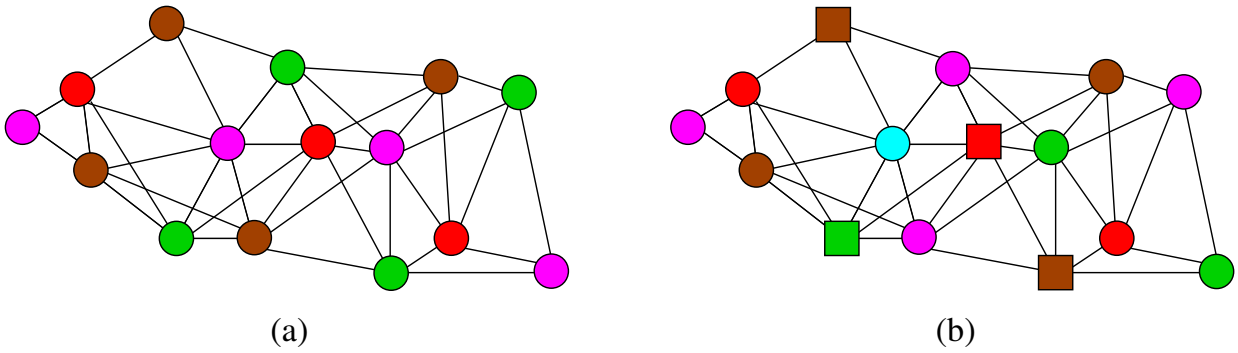


FIG. 3.9 – (a) Une solution optimale au problème de coloration sur le graphe proposé nécessite l'utilisation de 4 couleurs. (b) Une solution optimale au problème d'extension de précoloration sur le graphe proposé et étant donné qu'une couleur a été préalablement choisi pour les sommets marqués par un carré nécessite l'utilisation de 5 couleurs.

Dans [EL08] les auteurs travaillent sur une classe de graphe particulière sur laquelle une extension de précoloration optimale des sommets d'un graphe de cette classe peut être obtenue en un temps polynomial. Ils construisent alors une partition μ des éléments telle que $|\mu| \leq \text{OPT}_{\text{GRAPHCOL}}$ en ligne 7 de l'Algorithme 2, au lieu d'utiliser l'Algorithme 3.

Dans chacun des $|\mu|$ ensembles construits correspondant à une classe de couleur, des groupes peuvent être construits "localement", sans tenir compte des conflits, en utilisant un algorithme de BIN PACKING classique.

Comme on sait d'après [EL08] que BPAC est une généralisation du problème de coloration de graphe, $\text{OPT}_{\text{GRAPHCOL}} \leq \text{OPT}_{\text{BPAC}}$.

En tirant avantage de l'augmentation de ressources, on obtient un résultat général valable pour tout graphe.

3.4.3 Note sur la variante uniforme de BPCD

Rappelons que, de par sa ressemblance au problème du K-centre avec capacités (défini dans le Paragraphe 2.2.3), il est intéressant de considérer la variante uniforme du BPCD, comme définie dans le Paragraphe 2.4, qui considère les instances $\mathcal{I} = (S, d, w, d_{\max})$ de BPCD dans lesquelles la fonction de poids w associe à chaque élément de S un poids identique.

Dans ce cas, l'Algorithme 4, une adaptation simple de l'Algorithme 2, permet d'obtenir le Corollaire 3.28. Cet algorithme construit directement une partition des éléments en s'appuyant sur un ensemble indépendant maximal, et construit dans chacune des composantes de la partition des groupes contenant $\lfloor \frac{1}{w(x)} \rfloor$ éléments, $x \in S$ quelconque (puisque tous les éléments de S ont le même poids).

Algorithme 4 Un $(2,2)$-algorithme d'approximation pour le BPCD uniforme

1: construire une partition μ de S en utilisant l'Algorithme 3
2: appliquer l'algorithme *First-Fit-Decreasing* à chacun des $|\mu|$ ensembles induits par la partition μ.

Corollaire 3.28

L'Algorithme 4 est un $(2,2)$-algorithme d'approximation pour BPCD uniforme.

Démonstration : Posons $L = \lfloor \frac{1}{w(x)} \rfloor, \forall x \in S$.

La preuve du facteur d'approximation de cet algorithme se fait simplement en considérant le nombre de groupes ne contenant pas L éléments, que l'on va majorer par la taille de la partition, et donc la cardinalité d'une solution optimale, et le nombre de groupes de L éléments, qu'on peut majorer de la même façon.

Considérons une solution optimale de BPCD sur une instance $\mathcal{I} = (S, d, w, d_{\max})$ uniforme, notée OPT_{BPCD}, qui maximise le nombre de groupes contenant un nombre maximal d'éléments (il en existe toujours une).

Notons P l'ensemble des groupes de L éléments construit par l'Algorithme 4, et Q l'ensemble des groupes construit par cet algorithme, contenant strictement moins de L éléments.

On a $|P| \leq \lfloor \frac{|S|}{L} \rfloor < |\text{OPT}_{BPCD}|$, car $\lfloor \frac{|S|}{L} \rfloor$ représente le nombre de groupes nécessaires pour $|S|$ éléments s'ils contiennent tous exactement L éléments, donc l'optimal est plus grand, et $|P|$ ne peut pas être plus grand.

Remarquons que dans le cas uniforme, l'algorithme *First-Fit-Decreasing* (comme tout autre algorithme glouton), va construire presque uniquement des groupes contenant un nombre maximal d'éléments. En fait, au plus un groupe pour chacun des $|\mu|$ ensembles générés par la partition μ ne contient pas L éléments. Ainsi, $|Q| \leq |\mu|$, et, par le Lemme 3.25, $|\mu| \leq |\text{OPT}_{BPCD}|$. Au total, le nombre total de groupes construit par l'Algorithme 4, égal à $|P|+|Q|$, peut être majoré par $2|\text{OPT}_{BPCD}|$, d'où le facteur d'approximation annoncé. ■

3.5 Conséquences des résultats obtenus pour le K-centre avec capacités

3.5.1 Théorème de correspondance avec BPCD

Dans ce paragraphe nous présentons comment construire un algorithme d'approximation pour KCAPA (uniforme ou non) à partir d'un algorithme d'approximation pour BPCD, et *vice versa*.

Rappelons la définition de KCAPA, que l'on peut trouver dans le Paragraphe 2.2.3 : étant donnée une instance $\mathcal{K} = (S, d, w, K)$: un ensemble $S = \{e_1, \dots e_n\}$ d'éléments, un espace métrique (S, d), une fonction de poids $w : S \to [0; 1[$, un entier K, l'objectif est de trouver la plus petite valeur r_{max}, un ensemble $X \subseteq S$ de centres de taille au plus K et une association des éléments de S aux centres de X tels que le poids total des éléments associés à un même centre soit inférieur à 1, et que chaque élément soit associé à un centre à distance inférieure à r_{max} de lui-même.

Pour exploiter l'augmentation de ressources pour KCAPA, nous dirons qu'un algorithme $\mathcal{A}$ est un (α, β, γ)-algorithme d'approximation pour KCAPA (éventuellement uniforme) si il fonctionne en temps polynomial et construit une solution utilisant au plus αK centres (au lieu de K), telle que chaque élément soit associé à un centre à distance au plus $\beta \mathrm{OPT}_{\mathrm{KCAPA}}$ (au lieu de $\mathrm{OPT}_{\mathrm{KCAPA}}$), chaque centre se voyant associer un poids d'au plus γ (au lieu de 1).

Nos résultats ne traitent pas d'une éventuelle augmentation de ressources sur la capacité de chaque centre. Ainsi, tous nos résultats sont présentés avec $\gamma = 1$. Cependant dans [KS00] sont présentés des résultats utilisant de telles augmentations de ressources, que nous évoquons dans le Paragraphe 3.5.2. Pour pouvoir rapprocher les résultats que nous obtenons des résultats existants, nous avons conservé ce paramètre.

Theorème 3.29

Un (α, β)-algorithme d'approximation pour BPCD peut être transformé en un $(\alpha, 2\beta, 1)$-algorithme d'approximation pour KCAPA. *Dans l'autre sens, un $(\alpha, \beta, 1)$-algorithme d'approximation pour* KCAPA *peut être transformé en un $(\alpha, 2\beta)$-algorithme d'approximation pour BPCD.*

Pour démontrer ce théorème nous allons utiliser deux transformations, T et T', qui permettent respectivement de construire une solution de KCAPA depuis une solution de BPCD, et *vice versa*. En utilisant ces transformations et les propriétés des solutions qu'elles construisent, nous prouvons que si on a une solution à l'un des problèmes fournie par un algorithme d'approximation pour celui-ci, en utilisant la transformation adéquate, il est possible d'obtenir une solution approchée à un certain facteur de l'autre problème.

Etant donné un ensemble d'éléments pondérés dans un espace métrique, défini par la donnée du triplet (S, d, w), considérons deux transformations naturelles entre une construction de groupes et une affectation des éléments à des centres :

La transformation T :

- prend en entrée un ensemble de groupes, solution de l'instance $\mathcal{I} = (S, d, w, d_{\max})$ de BPCD, pour $d_{\max}$ fixé,
- dans chaque groupe, un élément est arbitrairement choisi pour devenir le centre de ce groupe,

– tous les autres éléments de ce groupe lui sont assignés ;

le rayon de l'affectation ainsi obtenue est au plus le diamètre maximal d'un groupe parmi l'ensemble des groupes en entrée, $d_{\max}$. Cette transformation est illustrée sur un exemple dans la Figure 3.10.

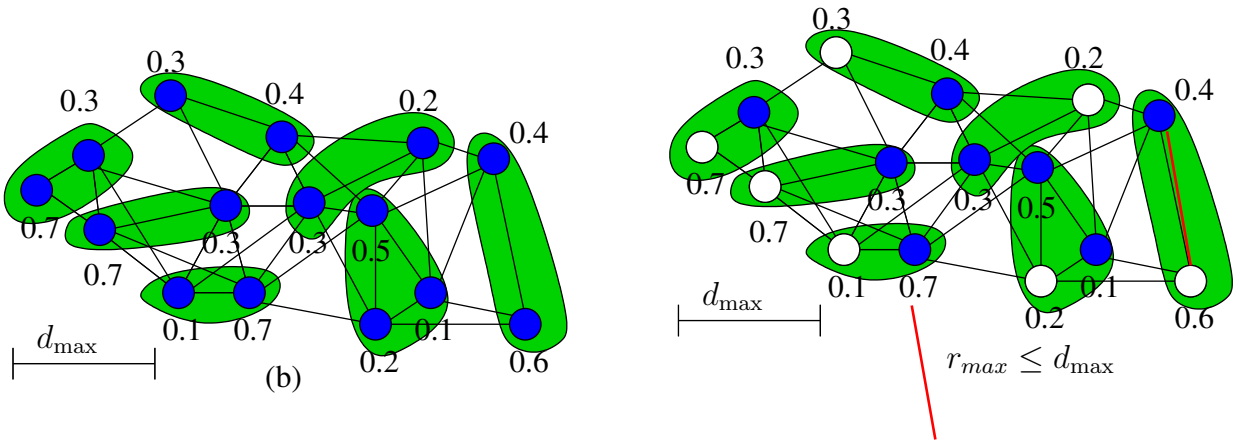


FIG. 3.10 – (a) Une instance de BPCD dans le plan euclidien, et une solution sur cette instance construisant 7 groupes. (b) La même instance ayant subie la transformation T pour en tirer une solution de KCAPA pour $K = 7$. Dans cette solution, $r_{max} \leq d_{\max}$.

La transformation T' :

– prend en entrée une affectation, solution de l'instance de KCAPA$\mathcal{K} = (S, d, w, K)$, pour K fixé,

– pour chaque centre, elle construit un groupe composé de tous les éléments affectés à ce centre ;

le diamètre des groupes obtenus est au plus deux fois le rayon de l'affectation correspondante, $2|\text{OPT}_{\text{KCAPA}}(\mathcal{K})|$. Cette transformation est illustrée sur un exemple dans la Figure 3.11.

En examinant la transformation T, on pourrait vouloir trouver un centre plus judicieux qu'un point arbitraire dans chaque groupe, pour diminuer le rayon de l'affectation obtenue. Remarquons qu'une solution à une instance $\mathcal{I} = (S, d, w, 1)$ de BPCD (avec, donc, $d_{\max}= 1$) où (S, d) est un espace métrique dans lequel tous les éléments sont à distance 1 les uns des autres construit des groupes de diamètre 1. La transformation T conduit à une affectation de rayon 1, qui est minimal sur une telle instance. Il existe donc des espaces métriques dans lesquels on ne peut pas utiliser de meilleures transformations pour passer d'un problème à l'autre.

Démonstration du Théorème 3.29 : Rappelons qu'on note $\text{OPT}_{BPCD}(\mathcal{I})$ une solution optimale de BPCD sur l'instance $\mathcal{I}$ et $|\text{OPT}_{BPCD}(\mathcal{I})|$ le nombre de groupes de cette solution. De même on note $\text{OPT}_{\text{KCAPA}}(\mathcal{K})$ une solution optimale de KCAPA sur l'instance $\mathcal{K}$ et $|\text{OPT}_{\text{KCAPA}}(\mathcal{K})|$ le rayon maximal de cette solution.

Soit $\mathcal{K} = (S, d, w, K)$ une instance de KCAPA. Pour $D > 0$, on utilise l'instance $\mathcal{I}(D) = (S, d, w, D)$ de BPCD construite sur le même ensemble d'éléments pondérés.

L'Algorithme 5 décrit comment un (α, β)-algorithme d'approximation $\mathcal{A}$ pour BPCD permet la construction d'une $(\alpha, 2\beta, 1)$ approximation pour KCAPA. La Figure 3.12 présente un schéma de la preuve (les flèches représentent un enchainement logique dans la preuve, dont les justifications sont détaillées dans cette preuve !).

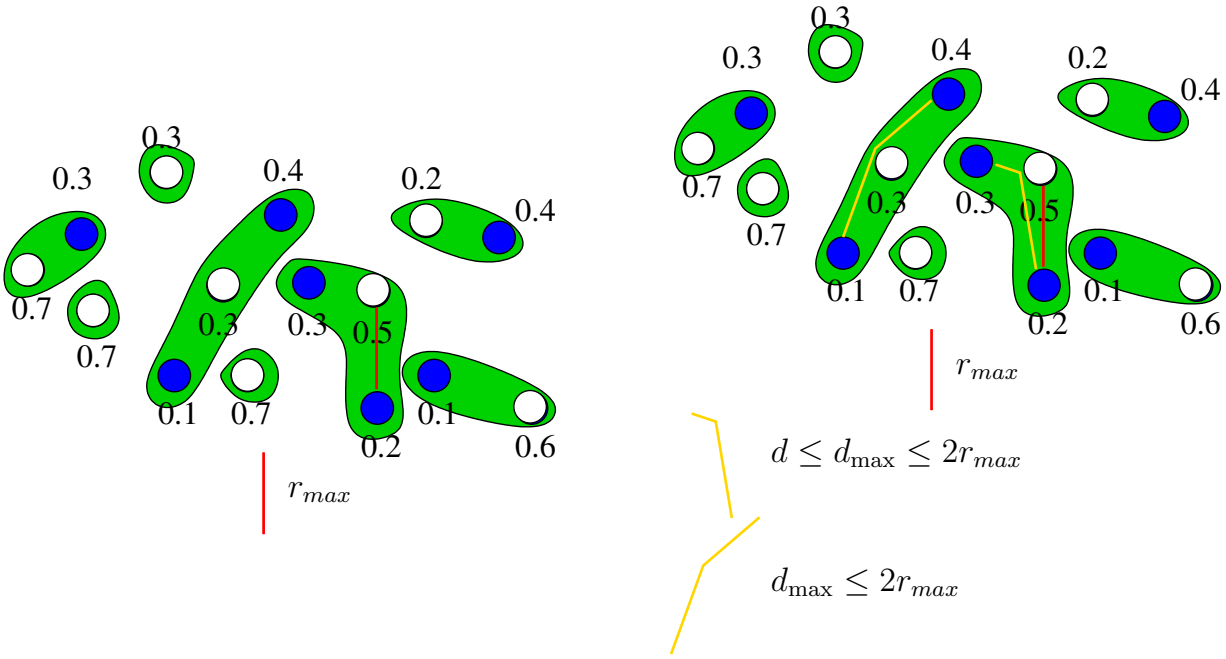


FIG. 3.11 – (a) Une instance de KCAPA dans le plan euclidien avec $K = 8$, et une solution sur cette instance. (b) La même instance ayant subie la transformation T' pour en tirer une solution de BCCD. Dans cette solution, $d_{\max} \leq 2r_{max}$.

On note $\mathcal{A}(\mathcal{I}(D))$ la solution construite par $\mathcal{A}$ sur $\mathcal{I}(D)$ et $|\mathcal{A}(\mathcal{I}(D))|$ le nombre de groupes de cette solution.

D'après l'Algorithme 5, d_α est la plus petite valeur telle que $|\mathcal{A}(\mathcal{I}(d_\alpha))| \leq \alpha K$. En effet, pour tout $d_i \leq D < d_{i+1}, \text{Comp}(\mathcal{I}(D), D) = \text{Comp}(\mathcal{I}(d_i), d_i)$. Ainsi il suffit de tester un nombre polynomial de valeurs de D pour trouver d_α.

Comme $\mathcal{A}$ est un (α, β)-algorithme d'approximation pour BPCD :

$$|\mathcal{A}(\mathcal{I}(D))| \leq \alpha |\text{OPT}_{BPCD}(\mathcal{I}(D))|.$$

Pour tout $D < d_\alpha$, on a :

$$|\mathcal{A}(\mathcal{I}(D))| > \alpha K$$

et donc

$$|\text{OPT}_{BPCD}(\mathcal{I}(D))| \geq \frac{|\mathcal{A}(\mathcal{I}(D))|}{\alpha} > \frac{\alpha K}{\alpha} = K. \tag{3.12}$$

Par ailleurs, $T'(\text{OPT}_{\text{KCAPA}}(\mathcal{K}))$ est une solution valide pour $\mathcal{I}(2|\text{OPT}_{\text{KCAPA}}(\mathcal{K})|)$ construisant au plus K groupes. En utilisant l'observation précédente, on a donc $2|\text{OPT}_{\text{KCAPA}}(\mathcal{K})| \geq d_\alpha$.

En construisant $T(\mathcal{A}(\mathcal{I}(d_\alpha)))$ on obtient donc sur l'instance $\mathcal{K}$ une affectation utilisant αK centres, de rayon maximal $\beta d_\alpha \leq 2\beta |\text{OPT}_{\text{KCAPA}}(\mathcal{K})|$, et où chaque centre se voit assigné un poids d'au plus 1. L'Algorithme 5 est donc bien un $(\alpha, 2\beta, 1)$-algorithme d'approximation pour KCAPA, si $\mathcal{A}$ est un (α, β)-algorithme d'approximation pour BPCD.

La preuve de l'autre sens est similaire. Soit $\mathcal{I} = (S, d, w, d_{max})$ une instance de BPCD. Pour K entier strictement positif, on utilise l'instance $\mathcal{K}(K) = (S, d, w, K)$ de KCAPA utilisant le même ensemble d'éléments pondérés.

Algorithme 5 Utilisation d'un (α, β)-algorithme d'approximation $\mathcal{A}$ pour BPCD pour la construction d'une $(\alpha, 2\beta, 1)$ approximation pour KCAPA.

1: trier les distances entre chaque paire d'éléments de S dans l'ordre croissant : $\{d_1, \ldots, d_{n(n+1)/2}\}$ /* d_1 est la distance entre les deux éléments les plus proches, $d_{n(n+1)/2}$ est le diamètre de S */
2: $i \leftarrow 1$
3: **tant que** $|\mathcal{A}(\mathcal{I}(d_i))| > \alpha K$ **faire**
4: $\quad i \leftarrow i + 1$
5: **fin tant que**
6: **si** $i > 1$ **alors**
7: $\quad i \leftarrow i - 1$
8: **sinon**
9: $\quad$ renvoyer "pas de solution pour KCAPA"
10: **fin si**
11: $d_\alpha \leftarrow d_i$ /* Pour clarifier les explications qui suivent */
12: construire la solution $\mathcal{A}(\mathcal{I}(d_\alpha))$
13: appliquer la transformation T sur cette solution

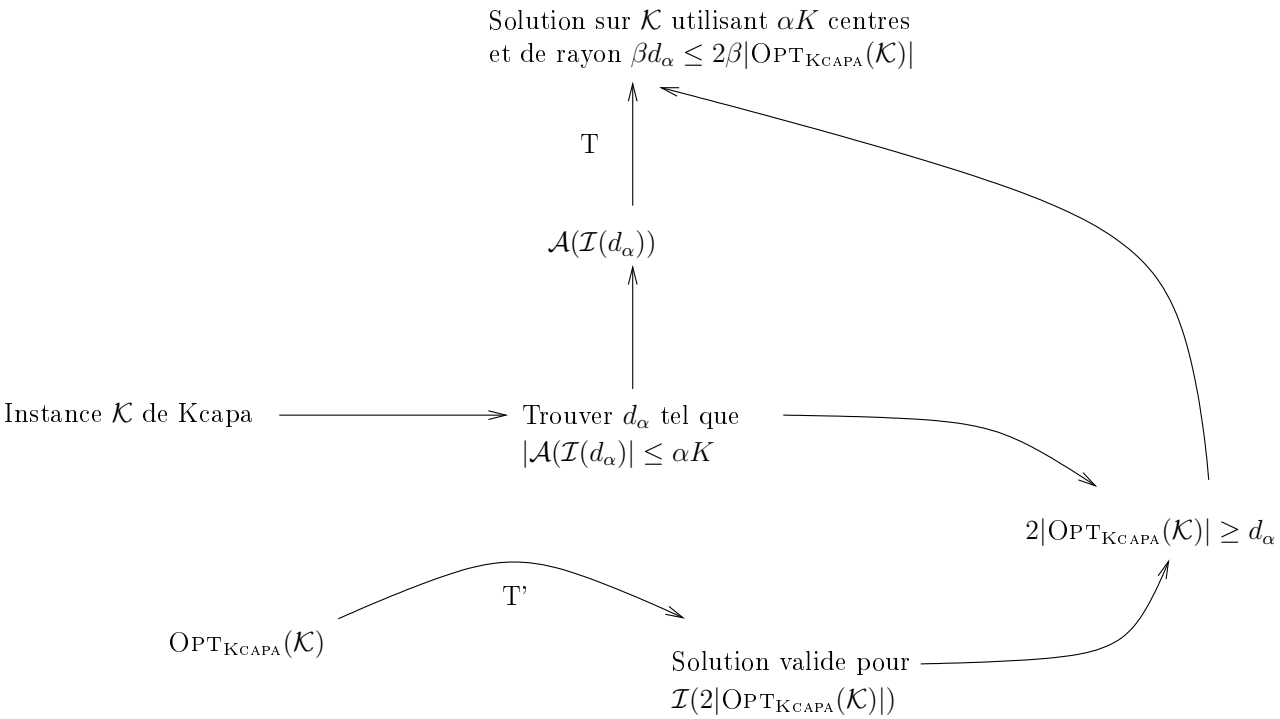


FIG. 3.12 – Schéma de la preuve qu'étant donné un (α, β)-algorithme d'approximation $\mathcal{A}$ pour BPCD, on peut obtenir une $(\alpha, 2\beta, 1)$ approximation pour KCAPA.

L'Algorithme 6 décrit comment on peut utiliser un algorithme d'approximation pour KCAPA afin de construire un algorithme d'approximation pour BPCD. Considérons un $(\alpha, \beta, 1)$-algorithme d'approximation $\mathcal{A}'$ pour KCAPA. La Figure 3.13 présente à son tour un schéma de cette preuve, les justifications de l'enchainement des flèches étant de même détaillées ici.

On note $\mathcal{A}'(\mathcal{K}(K))$ la solution construite par $\mathcal{A}'$ sur $\mathcal{K}(K)$, et $|\mathcal{A}'(\mathcal{K}(K))|$ le rayon de groupes de cette solution.

Algorithme 6 Utilisation d'un $(\alpha, \beta, 1)$-algorithme d'approximation $\mathcal{A}'$ pour KCAPA pour la construction d'une $(\alpha, 2\beta)$ approximation pour BPCD.

1: $K \leftarrow 0$
2: par dichotomie sur le nombre d'éléments de l'instance, trouver K la plus petite valeur telle que $|\mathcal{A}'(\mathcal{K}(K))| \leq \beta d_{\max}$
3: $K_\beta \leftarrow K$ // Pour clarifier les explications qui suivent
4: construire la solution $\mathcal{A}(\mathcal{K}(K_\beta))$
5: appliquer la transformation T' sur cette solution

Comme $\mathcal{A}'$ est un $(\alpha, \beta, 1)$-algorithme d'approximation pour KCAPA :

$$|\mathcal{A}'(\mathcal{K}(K))| \leq \beta |\text{OPT}_{\text{KCAPA}}(\mathcal{K}(K))|.$$

Pour tout $K < K_\beta$ on a :

$$|\mathcal{A}'(\mathcal{K}(K))| > \beta d_{\max},$$

et donc

$$\text{OPT}_{\text{KCAPA}}(\mathcal{K}(K)) \geq \frac{|\mathcal{A}'(\mathcal{K}(K))|}{\beta} > \frac{\beta d_{\max}}{\beta} = d_{\max}. \tag{3.13}$$

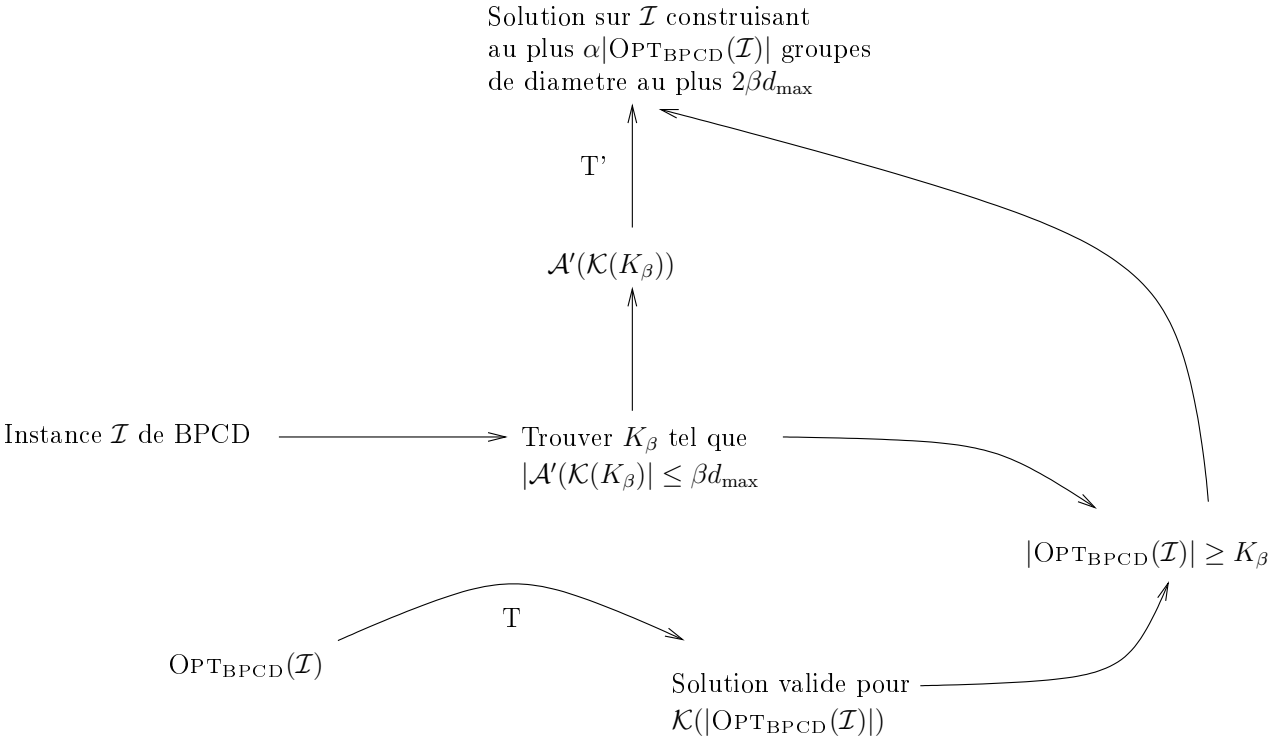


FIG. 3.13 – Schéma de la preuve qu'étant donné un $(\alpha, \beta, 1)$-algorithme d'approximation $\mathcal{A}'$ pour KCAPA, on peut obtenir une $(\alpha, 2\beta)$ approximation pour BPCD.

Par ailleurs, $T(\text{OPT}_{BPCD}(\mathcal{I}))$ est une solution valide pour $\mathcal{K}(\text{OPT}_{BPCD}(\mathcal{I}))$ de rayon au plus $d_{\max}$. En utilisant l'observation précédente, on a donc $|\text{OPT}_{BPCD}(\mathcal{I})| \geq K_\beta$.

En construisant $T'(\mathcal{A}'(\mathcal{K}(K_\beta)))$ on obtient sur l'instance $\mathcal{I}$ un ensemble d'au plus $\alpha K_\beta \leq \alpha|\text{OPT}_{BPCD}(\mathcal{I})|$ groupes de diamètres au plus $2\beta d_{\max}$ et de poids inférieur à 1. L'Algorithme 6 est donc bien un $(\alpha, 2\beta)$-algorithme d'approximation pour BPCD, si $\mathcal{A}'$ est un $(\alpha, \beta, 1)$-algorithme d'approximation pour KCAPA. ■

Remarque 3.30

Ce résultat s'applique aussi entre la variante uniforme du BPCD et la version classique (i.e. uniforme) de KCAPA*, i.e. un (α, β)-algorithme d'approximation pour BPCD uniforme peut être transformé en un $(\alpha, 2\beta, 1)$-algorithme d'approximation pour* KCAPAUNIF*, et, dans l'autre sens, un $(\alpha, \beta, 1)$-algorithme d'approximation pour* KCAPAUNIF *peut être transformé en un $(\alpha, 2\beta)$-algorithme d'approximation pour BPCD uniforme.*

3.5.2 Conséquences pour les versions uniformes

En combinant le Théorème 3.29 et le Corollaire 3.28, l'Algorithme 4, qui est un $(2, 2)$-algorithme d'approximation pour BPCD uniforme, peut être transformé en un $(2, 4, 1)$-algorithme d'approximation pour KCAPAUNIF. [KS00] donne plus généralement un $(\frac{2}{c}, 4, c)$-algorithme d'approximation pour tout $c \in [1; 2]$; nous venons donc de redémontrer le cas $c = 1$.

Remarquons que dans l'autre sens, le $(2, 4, 1)$-algorithme d'approximation présenté pour KCAPAUNIF devient un $(2, 8)$-algorithme d'approximation pour BPCD uniforme, en utilisant le Théorème 3.29, tandis que l'Algorithme 4 produit déjà une $(2, 2)$ approximation du même problème.

Par ailleurs, dans [KS00] est proposé un algorithme produisant une $(1, 6, 1)$-approximation pour KCAPAUNIF. En appliquant toujours le Théorème 3.29, on obtient un $(1, 12)$-algorithme d'approximation pour BPCD uniforme, c'est à dire qu'avec une augmentation de ressources de 12 sur le diamètre maximal d'un groupe, il est possible de construire un nombre de groupes optimal. Ce résultat est intéressant dans le sens où le Théorème 3.7 démontré dans le Paragraphe 3.2.2 nous indique que pour la version non uniforme de BPCD, il est impossible d'atteindre un nombre de groupes optimal, quelle que soit l'augmentation de ressources utilisée. Dans la version uniforme, donc, avec une augmentation de ressources suffisante, c'est possible. Remarquons tout de même que cette augmentation doit être supérieure à $(2 - \varepsilon)$, car le résultat du Théorème 3.6 s'applique aussi à la variante uniforme.

3.5.3 Conséquences pour les versions non uniformes

Concernant l'inapproximabilité des variantes non uniformes de ces problèmes, on peut tirer de l'équivalence entre KCAPA et BPCD le Théorème 3.31 en utilisant le Théorème 3.7. L'autre résultat d'inapproximabilité présenté dans le Paragraphe 3.2.2, le Théorème 3.6, n'apporte aucune autre information que celle de dire qu'il est dur d'atteindre la solution optimale en temps polynomial, *i.e.* que le K-centre classique est NP-difficile, ce qu'on sait déjà d'après [GJ+79].

Par extension de la notion d'inapproximabilité utilisée pour BCCD et BPCD, nous dirons que KCAPA n'est pas (α, β, γ)-approximable, pour certaines valeurs de α, β et γ, s'il n'existe pas

d'(α, β, γ)-algorithme d'approximation s'exécutant en temps polynomial pour KCAPA pour ces valeurs de α, β et γ, à moins que $P = NP$.

Theorème 3.31

$\forall\ \beta \geq 1$ et $\forall\ 1 \leq \alpha < 3/2$, KCAPA n'est pas $(\alpha, \beta, 1)$-approximable.

Démonstration : Un $(\alpha, \beta, 1)$-algorithme d'approximation pour KCAPA se transforme, d'après le Théorème 3.7, en un $(\alpha, 2\beta)$-algorithme d'approximation pour BPCD.

Pour $\beta \geq 1$ et $1 \leq \alpha < 3/2$, le Théorème 3.7 interdit l'existence d'un tel algorithme. ■

Ainsi il est impossible d'approcher à un quelconque facteur β près une solution optimale de KCAPA en temps polynomial sans avoir recours à une augmentation de ressources d'au moins 3/2 sur le nombre de centres à placer.

Enfin, en combinant le Théorème 3.29 avec le Théorème 3.26, l'Algorithme 2 peut être transformé en un $(\frac{7}{3}, 4, 1)$-algorithme d'approximation pour KCAPA, devenant ainsi le premier algorithme d'approximation connu pour ce problème. Remarquons que l'algorithme présenté dans [KS00] ne nous semble pas pouvoir être simplement adapté à une version non uniforme. On peut finalement voir la différence entre les facteurs d'approximations des versions uniformes et non uniformes (entre $\frac{7}{3}$ et 2) comme le coût induit par la non-uniformité des poids.

3.6 Conclusion partielle

Dans ce chapitre nous avons abordé l'inapproximabilité de BCCD et de BPCD sous plusieurs angles. Nous avons tout d'abord prouvé que, pour toute augmentation de ressources sur le diamètre des groupes strictement inférieure à 2, BCCD et BPCD ne sont pas approximables à facteur constant (respectivement Théorème 3.2 et Théorème 3.6). Nous avons par ailleurs démontré que BCCD (resp. BPCD) n'est pas approximable à un facteur strictement supérieur à 1/2 (resp. inférieur à 3/2), et ce quelle que soit l'augmentation de ressources utilisée (respectivement Théorème 3.3 et Théorème 3.7). Ces deux derniers résultats sont valables dans n'importe quel espace métrique, car ne dépendant pas des contraintes de distance présentes dans ces problèmes.

Nous avons ensuite présenté, un $(2/5, 4)$-algorithme d'approximation centralisé pour BCCD, valable pour tout espace métrique. Ce résultat a été publié dans [BBDL08]. Parallèlement, nous avons proposé un $(7/3, 2)$-algorithme d'approximation pour BPCD, également valable dans tout espace métrique.

Rappelons que les Figures 3.1 et 3.2 résument les résultats obtenus dans ce chapitre pour BCCD et BPCD respectivement, en terme d'inapproximabilité et d'approximation avec augmentation de ressources dans un espace métrique quelconque.

Enfin, nous avons présenté comment les résultats que nous avons obtenu pour BPCD s'adaptent à KCAPA, uniforme et non uniforme, et nous permettent d'obtenir, en plus de résultats d'inapproximabilité intéressants, le premier algorithme d'approximation connu pour KCAPA, assurant une approximation de 4 pour une augmentation de ressources sur le nombre de centres utilisés de 7/3.

Dans les chapitres suivants, nous restreignons nos études à des espaces métriques particuliers.

Ceci nous permet d'obtenir des résultats pour BCCD dans un environnement distribué, ce qui n'était pas possible dans un espace métrique quelconque. Nous obtenons également de meilleurs résultats d'approximation dans des espaces métriques destinés à représenter de façon fidèle les topologies des réseaux à grande échelle qui sont le domaine d'applications de nos problèmes.

Deuxième partie

Travaux dans des espaces métriques particuliers

Chapitre 4

Restriction à des espaces métriques spécifiques

Dans ce chapitre et le suivant, nous étudions les possibilités d'adaptation et d'extension de nos résultats pour BCCD et BPCD dans des espaces métriques particuliers. Plus spécifiquement, dans ce chapitre, nous nous intéressons aux espaces métriques suivants :

- $\mathbb{R}^D$ muni de la norme L_∞ pour définir les distances,
- le plan muni de la norme L_∞ (ou d'une norme quelconque),
- les espaces métriques induits par les arbres.

Dans chaque cas on considère que l'on connait un plongement explicite des éléments d'une instance dans l'espace considéré, *i.e.* pour $\mathbb{R}^D$, chaque élément est doté de coordonnées dans $\mathbb{R}^D$, dans le plan, de coordonnées dans $\mathbb{R}^2$, et quand nous étudions les arbres, nous considérons disposer d'un arbre dont font partie les éléments de notre instance. Nous n'abordons pas la question de comment trouver un tel plongement à partir de la seule donnée de la fonction de distance.

Lorsqu'on travaille dans $\mathbb{R}^D$ on souhaite que la distance entre deux éléments connus puisse être calculée en temps polynomial. Cependant la différence de deux éléments réels non rationnels (π et $3e\sqrt{2}$ par exemple) peut ne pas être calculable en temps polynomial. On préfère alors à l'utilisation de nombres réels l'utilisation de nombres rationnels à dénominateur borné. On s'assure ainsi que toutes les opérations classiques d'arithmétique se font en temps polynomial. Dans la suite nous utilisons $\mathbb{R}$ sans considération pour cette remarque en gardant à l'esprit qu'une extrême rigueur nous obligerait à restreindre les espaces métriques à ne pas utiliser n'importe quels réels.

On parlera dans la suite de métrique d'arbre (utilisée dans [ABK+07] par exemple), ou de métrique induite par un arbre. Un espace métrique (S, d) est une métrique d'arbre si il existe un arbre T, dont les arêtes sont pondérées de poids positifs, tel que $S \subseteq T$ et $d(u, v) = d_T(u, v)$ pour tout $u, v \in S^2$. L'arbre T peut contenir des noeuds additionnels, que nous appelons "noeuds virtuels", que nous considérons de poids nul lors de l'application de nos algorithmes.

En premier lieu, nous rappelons dans le Paragraphe 4.1 les résultats d'inapproximabilité concernant BCCD et BPCD et examinons dans quelle mesure ils peuvent s'appliquer dans des espaces métriques particuliers.

Ensuite, dans le Paragraphe 4.2, nous proposons plusieurs résultats, avec et sans augmentation de ressources, lorsque les éléments d'une instance de BCCD sont plongés dans un plan muni respectivement de la norme L_∞ ou d'une norme quelconque.

Nous présentons également, dans le Paragraphe 4.3, un $(1/3, 4)$-algorithme d'approximation distribué pour BCCD. Cet algorithme est dédié aux instances de BCCD dans lesquelles l'espace métrique contenant les éléments est $\mathbb{R}^D$, pour une dimension D fixée (faible), muni de la norme L_∞ pour définir les distances.

Dans le Paragraphe 4.5 nous présentons un $(1/3, 2)$-algorithme d'approximation pour BCCD, qui fonctionne lorsque les éléments d'une instance de BCCD sont plongés dans un arbre.

Dans le Paragraphe 4.6 nous étudions BPCD lorsque les instances utilisent comme espace métrique le plan euclidien.

Enfin nous présentons dans le Paragraphe 4.7 un $(\frac{5}{2}, 1)$-algorithme d'approximation pour BPCD, qui fonctionne lorsque les éléments d'une instance de BPCD sont plongés dans un arbre.

Dans le Chapitre 5, nous nous intéressons à l'application de ces problèmes à des espaces métriques ayant pour but de modéliser fidèlement les réseaux à grande échelle, qui sont le domaine d'applications de BCCD et BPCD. En particulier l'une des modélisations que nous étudions place les éléments dans un espace métrique qui est une généralisation du plan euclidien. C'est pour cela que nous nous sommes intéressés à l'étude de BCCD et de BPCD dans le plan, muni respectivement de la norme L_∞ et de la norme L_2. D'autres outils que nous évoquons dans le Chapitre 5 permettent de plonger Internet dans des espaces multidimensionnels, d'où l'intérêt de l'étude de $\mathbb{R}^D$. Comme la norme L_∞ et la norme L_2 sont équivalentes (les distances sont les mêmes à un facteur constant près), savoir qu'avec une des normes l'un ou l'autre des problèmes est approximable avec une certaine augmentation de ressources nous assure qu'avec L_2, il l'est aussi, à une modification du facteur d'approximation près. Enfin, nous nous intéressons également dans le Chapitre 5 à une modélisation d'Internet en utilisant des arbres, et appliquons à cette modélisation les résultats que nous obtenons dans ce chapitre.

4.1 NP-difficulté et inapproximabilité de BCCD et BPCD

4.1.1 Inapproximabilité de BCCD

Dans le Paragraphe 3.2, le Théorème 3.3 prouve que, quel que soit l'espace métrique utilisé dans les instances de BCCD, BCCD n'est pas (α, β)-approximable pour $\alpha > 1/2$ et pour n'importe quelle valeur de β.

L'autre théorème d'inapproximabilité concernant BCCD, le Théorème 3.2 démontré dans le Paragraphe 3.2.1, concerne l'inapproximabilité à facteur constant de BCCD en utilisant une augmentation de ressources sur le diamètre d'un facteur $(2 - \varepsilon)$, pour $\varepsilon > 0$. La preuve de ce théorème se fait via une réduction au problème de détection de l'existence d'une clique de taille fixée dans un graphe. Intuitivement, cette réduction ramène la difficulté de BCCD à celle de trouver, dans une instance $\mathcal{I}$, s'il existe des groupes valides.

Dans chacun des espaces métriques que nous étudions dans ce chapitre, il est possible de décider en temps polynomial de l'existence d'un groupe valide (*i.e.* un ensemble d'éléments de diamètre inférieur à $d_{\max}$ et de poids supérieur à 1), en en identifiant précisément un quand il

en existe. On pourrait donc penser que se restreindre à ces espaces métriques, en plus de rendre inapplicable ce dernier théorème d'inapproximabilité, va nous permettre d'obtenir des approximations plus fines de BCCD, par exemple ne nécessitant pas d'augmentation de ressources.

Plus généralement, la question qui se pose est : est-ce que, s'il est possible de détecter les groupes valides dans une instance en temps polynomial, BCCD devient "plus simple" dans un certain sens (approximable ? polynomialement solvable ?) ?

La réduction que nous allons présenter, depuis le problème de Couplage k-Dimensionnel Maximal (*Maximum k-Dimensional Matching*), ne nous donne pas de nouveaux résultats, mais nous propose une démonstration différente du Théorème 3.2, ainsi qu'une lumière nouvelle sur la difficulté de BCCD, dont nous discutons ensuite.

Theorème 4.1 (*Rappel du Théorème 3.2*)

$\forall \varepsilon > 0, \forall \alpha > 0$, *BCCD n'est pas* $(\alpha, (2-\varepsilon))$*-approximable.*

Définition 4.2 (*CkDM : Couplage k-Dimensionnel Maximal*)

Etant donnés un entier k *et un hypergraphe* $H = (V, F)$*, avec* $F \subseteq X_1 \times \cdots \times X_k$ *où* $X_1, \ldots, X_k$ *sont des sous-ensembles disjoints de* V*, et un entier* L*, existe-t-il un couplage de taille* L *pour* F*, i.e. un sous-ensemble* $M \subseteq F$*,* $|M| = L$ *et tel qu'aucune paire d'éléments de* M *n'ait de coordonnée commune ?*

Notons $\text{OPT}_{\text{CkDM}}(H)$ la taille du couplage k-dimensionnel maximal sur une instance H de CkDM. Hazan *et al.* ont prouvé [HSS06] que ce problème est impossible à approcher à un facteur $\Omega(\frac{\ln k}{k})$, à moins que $P = NP$.

Une instance de C3DM est proposée dans les Figures 4.1 et 4.2. Dans la Figure 4.2 est proposée une solution optimale sur cette instance, maximisant l'entier L pour lequel un couplage 3-dimensionnel est possible : ici $L = 2$. Tout comme dans une solution de BCCD, tous les sommets d'une instance de CkDM ne sont pas nécessairement utilisés.

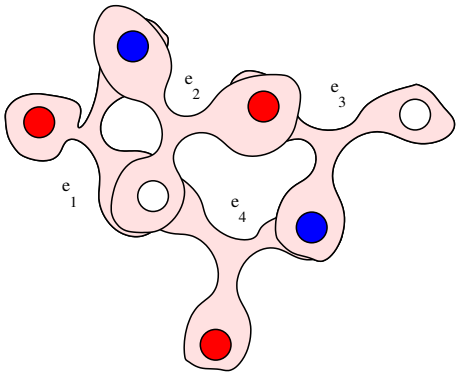


FIG. 4.1 – Une instance de C3DM. Chaque hyperarête (rose) est composée de trois éléments, un rouge, un bleu et un blanc.

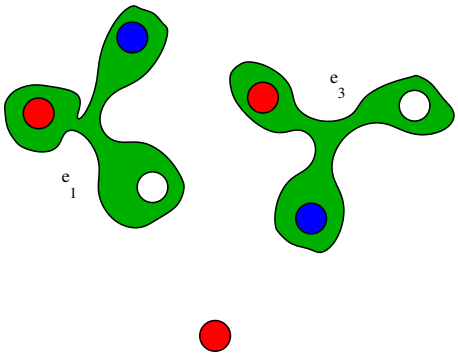


FIG. 4.2 – Une solution optimale sur l'instance de C3DM maximisant la taille L du couplage. Ici $L = 2$

Etant donnée une instance H de CkDM contenant n sommets, on peut, en utilisant la transformation I présentée ci-dessous, construire l'instance $I(H) = (S, d, w, 1)$ de BCCD équivalente (cf. exemple Figure 4.3) :

– chaque sommet de H est associé à un élément de S, dont le poids est $\frac{1}{n+1}$;

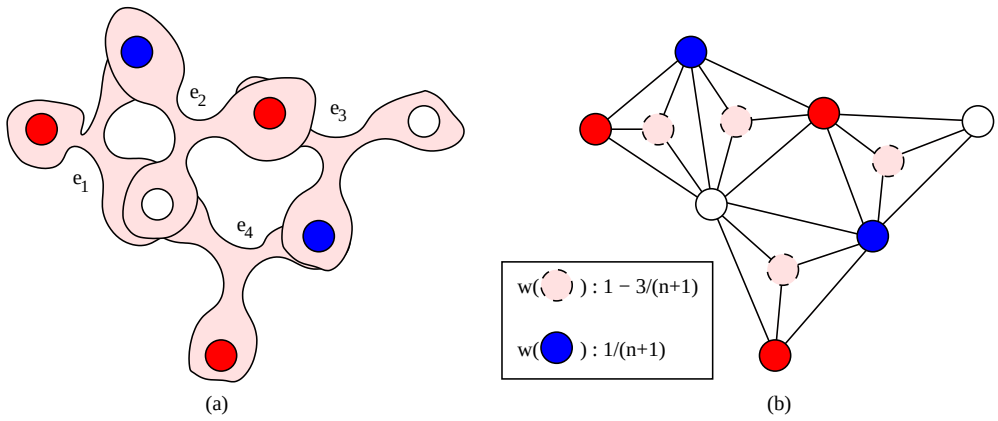


FIG. 4.3 – Un hypergraphe H et l'instance correspondante, $I(H)$. Chaque hyperarête (rose) est composée d'un élément rouge, d'un bleu et d'un blanc. Les hyperarêtes e_1 et e_3 composent une solution optimale à l'instance de C3DM de la Figure 4.3(a). L'instance de BCCD correspondante utilise un élément (rose) supplémentaire par hyperarête.

- chaque hyperarête de H est associée à un élément de S, appelé *centre*, dont le poids est $1 - \frac{k}{n+1}$;
- la fonction de distance d est définie comme la distance dans le graphe $G(H) = (S, E)$ où $(u, v) \in E \Leftrightarrow$ ils appartiennent à une même hyperarête dans H.

La transformation I s'applique en temps polynomial sur H pour obtenir $I(H)$.

On prouve le lemme suivant :

Lemme 4.3

Pour tout entier k, il existe un algorithme s'exécutant en temps polynomial qui, pour toute instance H de CkDM, construit une instance $I(H)$ de BCCD telle que $\text{OPT}_{\text{CkDM}}(H) = \text{OPT}_{BCCD}(I(H))$.

La démonstration de ce lemme utilise la propriété suivante :

Propriété 4.4

Soit H une instance de CkDM, et $I(H)$ l'instance de BCCD obtenue en appliquant la transformation I à H. Soit B un ensemble d'éléments de $I(H)$. Le diamètre de B est inférieur ou égal à 1 et B a un poids total supérieur à 1 si et seulement si B est constitué des $k + 1$ éléments correspondant à une hyperarête de H.

Démonstration : Par construction, l'ensemble des éléments correspondant à une hyperarête donnée de H est de poids 1 et de diamètre 1.

Réciproquement, soit B un ensemble d'éléments de poids supérieur à 1 et de diamètre inférieur à 1. Comme la distance entre deux centres est au moins 2, B contient au plus un centre. Par ailleurs, comme B a un poids supérieur à 1, et que le poids total des éléments de S qui ne sont pas des centres est strictement inférieur à 1, B doit contenir un centre, et en contient donc exactement un.

Pour atteindre un poids supérieur à 1, B doit être composé d'au moins k éléments qui ne sont pas des centres, en plus d'un centre c. Par construction, il y a exactement k

éléments non-centres à distance convenable de c : ceux correspondant à l'hyperarête de c dans H. B est donc composé de $k+1$ éléments correspondant à la même hyperarête. ■

Nous pouvons à présent prouver le Lemme 4.3, en utilisant la Propriété 4.4 :

Démonstration Lemme 4.3 : Soit une instance H de CkDM, et l'instance de BCCD correspondante $I(H)$. En utilisant le Lemme 4.4, si L groupes disjoints peuvent être construits dans $I(H)$, un couplage de taille L existe dans H, et réciproquement. En particulier, $\text{OPT}_{\text{CkDM}}(H) = \text{OPT}_{BCCD}(I(H))$. ■

Cette réduction de CkDM à BCCD nous permet de prouver le Théorème 3.2 d'une nouvelle façon.

Démonstration du Théorème 3.2 : Pour $\alpha > 0$, $\varepsilon > 0$, supposons qu'il existe un $(\alpha, (2-\varepsilon))$-algorithme d'approximation s'exécutant en temps polynomial pour BCCD, $\mathcal{A}$. Soit $|\mathcal{A}(I)|$ le nombre de groupes construit par $\mathcal{A}$ sur une instance I de BCCD.

A moins que $P = NP$, il est impossible [HSS06] de trouver un algorithme d'approximation pour le problème de CkDM assurant un facteur $\Omega(\frac{\ln k}{k})$, *i.e.* il existe une constante $c > 0$ telle que, pour n'importe quelle valeur de k suffisamment grande, il est impossible d'approcher le problème de CkDM à un facteur plus grand que $c\frac{\ln k}{k}$, à moins que $P = NP$.

Considérons alors le problème du CkDM pour k suffisamment grand (c'est à dire pour k tel que $c\frac{\ln k}{k} \leq \alpha$).

Soit H une instance de CkDM, et $I(H)$ l'instance correspondante de BCCD (cf. Lemme 4.3). Comme dans $I(H)$ le diamètre de n'importe quel ensemble d'éléments est entier, les groupes construits par $\mathcal{A}$ sur $I(H)$ ont aussi un diamètre inférieur à 1 (puisque inférieur à $(2-\varepsilon)$ et entier).

Ainsi $\mathcal{A}$ construit sur $I(H)$ une solution au problème du BCCD telle que $|\mathcal{A}(I(H))| \geq \alpha \text{OPT}_{BCCD}(I(H))$, sans augmentation de ressources.

$\mathcal{A}$ est donc un algorithme d'approximation assurant un facteur d'approximation de α pour BCCD sur les instances $I(H)$, où H est une instance quelconque de CkDM.

Via le Lemma 4.3, ceci implique qu'il existe un algorithme d'approximation s'exécutant en temps polynomial et assurant un facteur d'approximation de α pour CkDM, avec $\alpha \geq c\frac{\ln k}{k}$, ce qui est impossible à moins que $P = NP$. ■

Dans les instances de BCCD considérées par la réduction que nous venons de présenter, chaque groupe valide correspond à une hyperarête de l'instance correspondante de CkDM. Il est donc très simple d'identifier les groupes valides. Cependant le résultat obtenu montre bien qu'il ne suffit pas de savoir détecter ces groupes valides, mais que le problème de savoir choisir quels groupes construire parmi un ensemble de groupes qu'on sait virtuellement valides est encore compliqué.

Cette observation nous permet d'expliquer pourquoi, bien que dans les espaces métriques que nous allons considérer dans la suite les groupes valides soient facilement (*i.e.* en temps polynomial) identifiables, il reste difficile d'élaborer des algorithmes permettant d'y construire des solutions à BCCD en temps polynomial.

La réduction à CkDM que nous venons de proposer n'est pas transposable dans les espaces métriques que nous allons étudier, ainsi dans ces espaces métriques nous ne disposons pas de preuve formelle que BCCD n'est pas approximable en utilisant une augmentation de ressources strictement inférieure à 2 sur le diamètre des groupes. D'ailleurs, nous prouvons dans le Paragraphe 4.2 qu'il existe bien un algorithme d'approximation sans augmentation de ressources pour résoudre BCCD dans le plan muni de la norme L_∞.

Par contre, en ce qui concerne la version de ce problème dans les arbres, nous ne sommes pas en mesure de présenter un algorithme assurant un facteur d'approximation constant en ayant recours à une augmentation de ressources inférieur à 2. Comme nous venons de le dire, même si dans ce cas l'on dispose des groupes valides (ce qui est possible dans un arbre), il reste compliqué, de choisir lesquels construire. L'utilisation d'une augmentation de ressources de 2 permet d'obtenir un algorithme, présenté dans le Paragraphe 4.5, assurant un facteur d'approximation de $\frac{1}{3}$. Il est ouvert de savoir s'il est possible d'élaborer un algorithme fonctionnant en temps polynomial qui permettrait d'approcher BCCD dans les arbres à facteur constant.

4.1.2 Inapproximabilité de BPCD

Dans le Paragraphe 3.2, le Théorème 3.7 prouve que, quel que soit l'espace métrique utilisé dans les instances de BPCD, BPCD n'est pas (α, β)-approximable pour $\alpha < 3/2$ et pour n'importe quelle valeur de β.

Par ailleurs le Théorème 3.6 démontré dans le Paragraphe 3.2.2, concerne l'inapproximabilité à facteur constant de BPCD en utilisant une augmentation de ressources sur le diamètre d'un facteur $(2-\varepsilon)$, pour $\varepsilon > 0$. Il n'est plus vrai dans les espaces métriques utilisés dans ce chapitre. Sa démonstration reposait également sur des propriétés de l'espace métrique contenant les points, qui ne sont plus vérifiées dans les espaces métriques que nous considérons désormais. Nous prouvons d'ailleurs dans la suite qu'il existe bien un algorithme d'approximation sans augmentation de ressources pour résoudre BPCD dans le plan comme dans les arbres.

4.2 BCCD dans le plan

Nous allons dans un premier temps, dans le Paragraphe 4.2.1, présenter un algorithme qui assure un facteur d'approximation dépendant de la structure de l'espace considéré. Cet algorithme s'exécute en temps polynomial si, dans l'espace métrique dans lequel sont placés les éléments de BCCD, la détection et la construction de groupes valides de diamètre $d_{\max}$ peut se faire en temps polynomial. Nous prouvons alors, dans le Paragraphe 4.2.3, que dans le plan muni de la norme L_∞ un tel algorithme existe, ce qui nous permet d'obtenir un $(\frac{1}{11}, 1)$-algorithme d'approximation pour BCCD.

Dans le Paragraphe 4.2.4 nous présentons un résultat inspiré de cet algorithme, qui nécessite une augmentation de ressources d'un facteur 3 sur le diamètre des groupes. Si dans l'espace métrique considéré, un algorithme permettant la détection et la construction de groupes valides de diamètre $d_{\max}$ existe, ce nouvel algorithme s'exécute en temps polynomial et est un $(\frac{2}{5}, 3)$-algorithme d'approximation pour BCCD. Un algorithme permettant la détection et la construction de tels groupes existe quand l'espace métrique utilisé est le plan muni d'une norme quelconque. Nous le présentons dans le Paragraphe 4.2.2

4.2.1 Un algorithme d'approximation sans augmentation de ressources pour BCCD

Etant donné un espace métrique (S, d), définissons $\gamma(S)$ de la façon suivante :

Définition 4.5 ($\gamma(S)$)

Etant donné un espace métrique (S, d), on définit $\gamma(S)$ comme le plus petit entier suffisamment grand pour que n'importe quel ensemble de diamètre inférieur ou égal à 3 soit couvert par γ boules de rayon 1/2.

En particulier, pour tout espace vectoriel E de dimension fini, l'espace métrique $S(E)$ correspondant est tel que $\gamma(S(E))$ est fini.

Il est possible d'approcher la solution optimale de BCCD dans un espace métrique (S, d) dans lequel $\gamma(S)$ est fini sans utiliser d'augmentation de ressources. Le facteur d'approximation assuré par l'algorithme que nous présentons dépend de $\gamma(S)$.

Nous présentons l'Algorithme 7 qui, dans un espace métrique (S, d), est un $(\frac{1}{\gamma(S)+2}, 1)$-algorithme d'approximation, *qui ne s'exécute pas toujours en temps polynomial*, mais dont le temps d'exécution dépend de l'espace (S, d). Cet algorithme construit, tant qu'il en existe, des groupes de diamètre $d_{\max}$, et ce de façon gloutonne.

Par gloutonne on entend qu'on ajoute des éléments dans un groupe jusqu'à ce que le poids du groupe dépasse 1 (en respectant la contrainte de distance), et qu'une fois que la contrainte de poids est satisfaite, plus aucun élément n'y est ajouté. Ceci a pour conséquence que tous les groupes ainsi construits ont un poids strictement inférieur à 2 (puisque tous les éléments considérés ont un poids strictement inférieur à 1).

C'est cette capacité à savoir détecter l'existence d'un groupe qui diffère selon l'espace métrique. Dans certains cas, comme nous l'évoquons dans le Paragraphe 4.2.2, il est possible de détecter l'existence d'un groupe en temps polynomial, ce qui rend le temps d'exécution de l'Algorithme 7 polynomial.

Algorithme 7 $(\frac{1}{\gamma(S)+2}, 1)$-algorithme d'approximation glouton pour BCCD

1: $U \leftarrow S$ // Eléments qui n'appartiennent à aucun groupe
2: **tant que** il existe un groupe valide de diamètre $d_{\max}$ **faire**
3: construire un tel groupe de façon gloutonne, de poids strictement inférieur à 2,
4: supprimer de U les éléments appartenant au groupe construit
5: **fin tant que**

Theorème 4.6

L'Algorithme 7 appliqué à une instance où l'espace métrique est (S, d) est un $(\frac{1}{\gamma(S)+2}, 1)$-algorithme d'approximation pour BCCD, s'exécutant en temps polynomial si un algorithme de détection et de construction d'un groupe valide de diamètre $d_{\max}$ s'exécutant en temps polynomial existe quand les éléments d'une instance de BCCD sont dans (S, d).

Démonstration : Clairement, si un algorithme de détection et de construction de groupes valides de diamètre $d_{\max}$ existe dans (S, d), l'Algorithme 7 s'exécute en temps polynomial.

La preuve du facteur d'approximation de cet algorithme se fait en examinant le nombre de groupes qu'une solution optimale aurait pu créer à la place de celles construites par l'exécution de l'Algorithme 7.

On choisit une solution optimale, notée OPT_{BCCD}. On note $\mathcal{B}$ l'ensemble des groupes construits par l'Algorithme 7.

On réutilise la notion de zone étendue introduite dans la preuve du Théorème 3.14 du Paragraphe 3.3, illustrée dans la Figure 4.4(a). Comme dit précédemment, tout groupe construit par une solution optimale est inclus dans la zone étendue d'un groupe $B \in \mathcal{B}$, puisque au moins un de ses éléments appartient à un groupe $B \in \mathcal{B}$. Ainsi, tous ses éléments sont à distance au plus $d_{\max}$ de ce groupe B. On définit le *poids perdu* dans la zone étendue d'un groupe $B \in \mathcal{B}$, $w_{\text{perdu}}(B)$, comme le poids dans la zone étendue de B non utilisé par l'Algorithme 7. Ce poids a pu être utilisé par OPT_{BCCD}. Remarquons que le poids de B n'est pas pris en compte dans cette valeur.

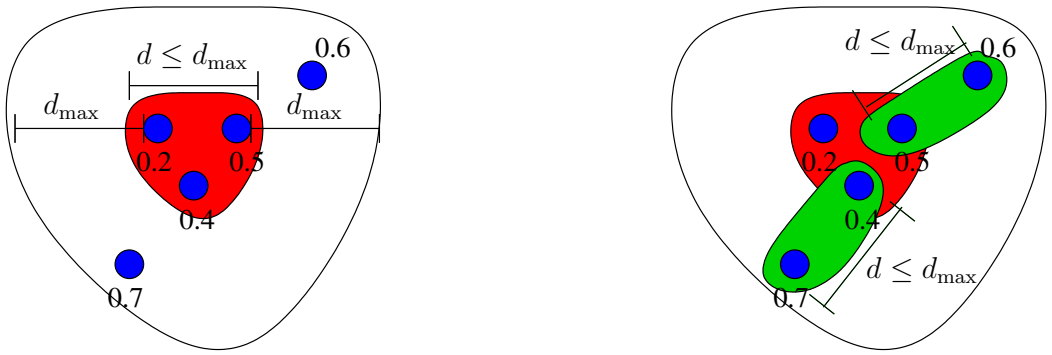


FIG. 4.4 – (a) La zone étendue d'un groupe valide dans le plan euclidien, de diamètre inférieur à $3d_{\max}$. (b) Tout groupe d'une solution optimale intersecte un groupe construit par l'Algorithme 7, et est donc inclus dans sa zone étendue.

La terminaison de l'Algorithme 7 nous assure que dans chacune des zones étendues il n'existe pas d'ensemble d'éléments de poids supérieur à 1 et de diamètre inférieur à $d_{\max}$. En s'appuyant sur $\gamma(S)$ on a le lemme suivant :

Lemme 4.7

Pour chaque groupe B construit par l'Algorithme 7, $w_{perdu}(B) < \gamma(S)$.

Démonstration : Supposons que dans une zone étendue, le poids résiduel soit supérieur à $\gamma(S)$. Remarquons d'abord que la zone étendue d'un groupe est de diamètre au plus $3d_{\max}$ ($d_{\max}$ de chaque "côté" du groupe, lui même de diamètre au plus $d_{\max}$, voir Figure 4.4(b)). Cette zone étendue, quand le diamètre du groupe central est maximal et égal à $d_{\max}$, est couverte par au plus $\gamma(S)$ boules de diamètre $d_{\max}$.

Prenons une telle couverture composée de $\gamma(S)$ boules de diamètre au plus $d_{\max}$. Comme le poids de cette zone étendue est supérieur à $\gamma(S)$, une de ces boules a nécessairement un poids supérieur à 1, et constitue donc un groupe valide, ce qui est en contradiction avec la fin de l'exécution de l'Algorithme 7. ■

Comme chaque groupe construit a un poids strictement inférieur à 2, on peut récapituler et en déduire que la solution optimale OPT_{BCCD} construit au plus $(\gamma(S)+2)|\mathcal{B}|$ groupes. ■

4.2.2 Détection d'un groupe valide en temps polynomial

Nous présentons à présent un algorithme permettant la détection et la construction d'un groupe valide dans le plan muni d'une norme quelconque. L'utilisation de cet algorithme permet à l'Algorithme 7 de s'exécuter en temps polynomial dans le plan muni de la norme L_∞. Lors de l'étude de ce résultat nous utilisons la notion de *lentille* liée à une paire d'éléments dans l'espace.

Définition 4.8 (*Lentille de* $\mathbf{x}$ *et* $\mathbf{y}$)

Soient deux points $\mathbf{x}$ *et* $\mathbf{y}$ *dans un espace métrique. La lentille (symétrique) de* $\mathbf{x}$ *et de* $\mathbf{y}$, *dénotée* $Le(\mathbf{x},\mathbf{y})$, *consiste en l'ensemble des points* $\mathbf{z}$ *de l'espace tels que* $d(\mathbf{x},\mathbf{z}) \leq d(\mathbf{x},\mathbf{y})$ *et* $d(\mathbf{y},\mathbf{z}) \leq d(\mathbf{x},\mathbf{y})$.

L'algorithme 8 examine chacune des lentilles de diamètre inférieure à $d_{\max}$, identifiables par chaque paire d'éléments $(\mathbf{x},\mathbf{y})$ d'une instance. Au sein de chaque lentille est construit le complémentaire du graphe de compatibilité des éléments, liant deux éléments d'une lentille s'ils sont à plus de $d_{\max}$ l'un de l'autre. Ce graphe est biparti, car chaque demi-lentille (ensemble d'éléments d'une lentille situés du même côté du segment $[\mathbf{xy}]$, voir Figure 4.5) est de diamètre inférieur à $d(\mathbf{x},\mathbf{y})$, donc inférieur à $d_{\max}$. Dans un tel graphe biparti, on peut chercher un ensemble indépendant de poids maximum en temps polynomial. Si un tel ensemble est de poids supérieur à 1, alors un groupe valide de diamètre au plus $d(\mathbf{x},\mathbf{y})$ existe dans cette lentille. Le mécanisme de recherche d'un ensemble indépendant de poids maximum dans un graphe biparti est largement inspirée de la recherche d'un ensemble indépendant de cardinalité maximum dans un tel graphe, bien connu en algorithmique classique (qui se ramène, via le Théorème de König [LP86], à un problème de couplage dans un graphe biparti, qui se résoud en utilisant le Théorème de Hall).

Dans la suite, étant donné un graphe $G=(V,E)$, on note $\overline{G}=(V,\overline{E})$ son complémentaire, dans lequel $\overline{E}=\{(u,v)\in V^2,(u,v)\notin E\}$.

Algorithme 8 Algorithme de détection de l'existence d'un groupe

1: **pour** chaque paire de noeuds $(\mathbf{x},\mathbf{y})$ telle que $d(\mathbf{x},\mathbf{y}) \leq d_{\max}$ **faire**
2: identifier tous les éléments appartenant à $Le(\mathbf{x},\mathbf{y})$
3: considérer le graphe $\overline{G}_{\mathbf{x},\mathbf{y}} = \overline{\text{Comp}(\mathcal{I}, d_{\max}) \cap Le(\mathbf{x},\mathbf{y})}$
4: **si** dans $\overline{G}_{\mathbf{x},\mathbf{y}}$, un ensemble indépendant de poids maximum est de poids supérieur à 1 **alors**
5: mettre un par un les éléments de cet ensemble dans un même groupe, jusqu'à ce qu'il dépasse 1. Il est de poids supérieur à 1 et de diamètre inférieur à $d_{\max}$. Renvoyer "groupe construit".
6: **sinon**
7: renvoyer "plus de groupes"
8: **fin si**
9: **fin pour**

Lemme 4.9

L'Algorithme 8 permet de décider en temps polynomial s'il est possible de créer un groupe valide, de poids supérieur à 1 *et de diamètre inférieur à* $d_{\max}$, *dans le plan muni d'une norme*

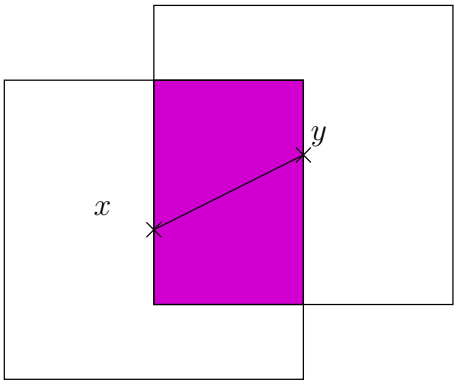


FIG. 4.5 – Deux points $\mathbf{x}$ et $\mathbf{y}$ dans le plan muni de la norme L_∞ et la lentille (colorée) $Le(\mathbf{x}, \mathbf{y})$. Deux points dans une même demi-lentille (chacun des côtés du segment $[\mathbf{xy}]$) sont à distance inférieure à $d(\mathbf{x}, \mathbf{y})$.

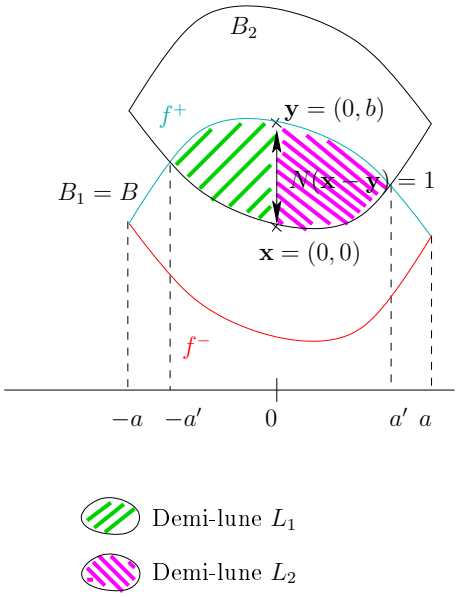


FIG. 4.6 – Deux points $\mathbf{x}$ et $\mathbf{y}$ dans le plan muni de la norme définie par la boule B. f^- suit le bord inférieur de B, f^+ en suit le bord supérieur. $-a$ est l'abscisse de l'extrémité gauche de B, et a est l'abscisse de l'extrémité droite. La lentille $Le(\mathbf{x}, \mathbf{y})$ est l'union des demi-lentilles L_1 et L_2, l'intersection de B_1 et de B_2 par ailleurs.

quelconque.

Pour prouver le Lemme 4.9, on utilise l'Algorithme 8, inspiré du travail de Aggarwal *et al.* dans [AIKS91], qui permet de décider s'il existe un groupe constructible. En particulier, nous utilisons le lemme suivant :

Lemme 4.10

Pour toute paire $(\mathbf{x}, \mathbf{y})$ du plan muni d'une norme quelconque, le graphe $\overline{G}_{\mathbf{x},\mathbf{y}} = \overline{Comp(\mathcal{I}, d_{\max}) \cap Le(\mathbf{x}, \mathbf{y})}$ est biparti.

Démonstration : Considérons l'espace métrique $(\mathbb{R}^2, N)$ pour une norme N quelconque. La donnée d'un convexe fermé, borné, symétrique par rapport à 0 et d'intérieur non vide B, centré en 0, définit une unique norme dont B est la boule unité.

Sans perte de généralité, considérons deux points de $\mathbb{R}^2$ de coordonnées respectivement $\mathbf{x} = (0, 0)$ et $\mathbf{y} = (0, b)$, tels que $N(\mathbf{x}-\mathbf{y}) = 1$. On note $B_1 = B = B(\mathbf{x}, 1)$ et $B_2 = B(\mathbf{y}, 1)$, ce qui implique que $\mathbf{x}$ est sur le bord de B_2 et $\mathbf{y}$ est sur le bord de B_1.

On définit la lentille de $\mathbf{x}$ et $\mathbf{y}$ comme $Le(\mathbf{x}, \mathbf{y}) = B_1 \cap B_2$. Le segment $[\mathbf{xy}]$ coupe $Le(\mathbf{x}, \mathbf{y})$ en deux parties, L_1 et L_2. Un exemple de lentille est présenté dans la Figure 4.5, dans le plan muni de la norme L_∞. Le cas plus général est présenté dans la Figure 4.6, qui est muni des notations que nous utilisons dans cette démonstration.

On souhaite tout d'abord démontrer que pour tout $\mathbf{z}$ et $\mathbf{t}$ dans L_1, $N(\mathbf{z}-\mathbf{t}) \leq 1$ (la preuve sera valide pour L_2 par symétrie des arguments).

Posons $a = \max\{x : \exists(x,y) \in B_1\}$. On a alors $-a = \min\{x : \exists(x,y) \in B_1\}$. a est l'abscisse de "l'extrémité droite" de B_1, et $-a$ est l'abscisse de "l'extrémité gauche" de B_1.

Considérons les deux fonctions

$$f^-(x) = \min\{y : (x,y) \in B_1\}$$

et

$$f^+(x) = \max\{y : (x,y) \in B_1\},$$

qui suivent respectivement le bord inférieur et le bord supérieur de B_1. Ces deux fonctions sont symétriques par rapport à l'origine, et donc $f^+(-x) = -f^-(x)$. Par ailleurs f^- est convexe, et f^+ est concave.

Caractérisons à présent L_1 : on a d'après les définitions de B_1, B_2, f^+ et f^- :

$$B_1 = \{(x,y) : -a \leq x \leq a, f^-(x) \leq y \leq f^+(x)\},$$
$$B_2 = \{(x,y) : -a \leq x \leq a, f^-(x)+b \leq y \leq f^+(x)+b\}.$$

On a donc

$$L_1 = \{(x,y) : 0 \leq x \leq a, f^-(x)+b \leq y \leq f^+(x)\}.$$

La fonction $f^+(x) - f^-(x)$ est concave, et paire car $f^+(-x) - f^-(-x) = -f^-(x) + f^+(x)$. Ainsi $f^+ - f^-$ est décroissante sur $[0,a]$. On définit alors

$$a' = \min\{x : f^+(x) - (f^-(x)+b) \leq 0\} \cup \{a\},$$

i.e. a' est l'abscisse du point d'intersection des bords de B_1 et B_2 dans L_1 s'il existe, a sinon (au cas où le bord de la boule comprendrait un segment vertical).

Utilisant cette notation, on peut remarquer que pour un élément $(x,y) \in L_1$, $x \leq a'$, ce qui nous donne une nouvelle caractérisation de L_1 : $L_1 = \{(x,y) : 0 \leq x \leq a', f^-(x)+b \leq y \leq f^+(x)\}$.

Considérons à présent l'ensemble $L_1 - L_1$ composé des éléments dont les coordonnées résultent de la différence des coordonnées de deux éléments de L_1. Nous prouvons que

$$L_1 - L_1 \subseteq \{(x,y) : -a' \leq x \leq a', f^-(x) \leq y \leq f^+(x)\}.$$

Clairement, étant donnés deux points (x,y) et (x',y') tous deux dans L_1, $-a' \leq x - x' \leq a'$.

Pour x fixé dans $[-a', a']$, on regarde l'ensemble suivant : $\{y : (x,y) \in L_1 - L_1\}$. Cet ensemble est un intervalle, de par la convexité de $L_1 - L_1$ (qui découle de la convexité de L_1). Pour chaque y de cet ensemble, il existe deux points de L_1 de coordonnées respectivement (α, β) et $(\alpha + x, \beta + y)$ dont la différence est (x,y).

On sait donc que :

$$\beta \in [f^-(\alpha)+b, f^+(\alpha)],$$
$$\beta + y \in [f^-(\alpha+x)+b, f^+(\alpha+x)].$$

Ceci implique l'inégalité suivante :

$$f^-(\alpha+x)+b-f^+(\alpha) \leq y \leq f^+(\alpha+x) - (f^-(\alpha)+b).$$

Examinons cet encadrement de y.

Quand $\alpha = 0$, $f^-(\alpha + x) + b - f^+(\alpha) = f^-(x)$, car $f^+(0) = b$, et quand $\alpha = -x$, $f^-(\alpha + x) + b - f^+(\alpha) = -f^+(-x)$, car $f^-(0) = -b$.

Or $f^-(x) = -f^+(-x)$. Comme $f^- - f^+$ est convexe, $f^-(\alpha + x) + b - f^+(\alpha)$ est croissante (comme fonction de α) pour $0 \leq \alpha \leq a'$, donc atteint son minimum quand $\alpha = 0$, donc $y \geq f^-(x)$,

Par ailleurs, quand $\alpha = 0$, $f^+(\alpha + x) - (f^-(\alpha) + b) = f^+(x)$, car $f^-(0) = -b$, et quand $\alpha = -x$, $f^+(\alpha + x) - (f^-(\alpha) + b) = -f^-(-x)$, car $f^+(0) = b$.

Or $f^+(x) = -f^-(-x)$. Comme $f^+ - f^-$ est concave, $f^+(\alpha + x) - (f^-(\alpha) + b)$ est décroissante (comme fonction de α) pour $0 \leq \alpha \leq a'$, donc atteint son maximum quand $\alpha = 0$, donc $y \leq f^+(x)$.

Ce qui, en récapitulant, nous donne $f^-(x) \leq y \leq f^+(x)$. Comme ceci est vrai pour tout $x \in [-a', a']$, on a bien

$$L_1 - L_1 \subseteq \{(x, y) : -a' \leq x \leq a', f^-(x) \leq y \leq f^+(x)\} \subseteq B_1$$

.

Ainsi, pour toute paire d'éléments e_1 et e_2 de L_1, $N(e_1, e_2) \leq 1$. En suivant un raisonnement symétrique pour L_2, on en déduit que tous les points de $Le(e_1, e_2)$ se trouvant dans la même demi-lentille (L_1 ou L_2) forment un ensemble indépendant de $\overline{G}_{e_1,e_2}$. En considérant également la demi-lentille opposée, on a un graphe biparti. ■

Démonstration du Lemme 4.9 : Construire un ensemble indépendant de poids maximum dans un graphe biparti $G = (L \cup R, E)$, où L et R sont les deux composantes du graphe biparti G, se fait en temps polynomial de façon exacte. Pour cela on utilise un algorithme de calcul d'une $s - t$-coupe de capacité minimum. Une fois la coupe obtenue, on pourra récupérer l'ensemble indépendant de poids maximum dans G.

Définition 4.11 (*Problème de coupe minimum*)

Etant donné un graphe $G = (V, E)$ avec une pondération positive des arêtes $w \in [0; 1[$, trouver une $(s - t)$-coupe (S, T) de G de capacité $c(S, T)$ minimum, i.e. un sous-ensemble d'arêtes $D \subseteq V$ qui partitionne G en deux sous-ensembles S et T tels que $s \in S$ et $t \in T$, tel que $\forall (u, v) \in D$, $u \in S$ et $v \in T$ ou $v \in S$ et $u \in T$; et tel que la somme des poids des arêtes de D est minimum.

On va utiliser un algorithme de calcul de $(s - t)$-coupe minimum dans le graphe $G' = (L \cup R \cup \{s, t\}, E')$, dans lequel une source s et une cible t sont ajoutées, et où E' est l'union des arêtes de E et des arêtes (s, L) et (R, t), respectivement entre la source s et chaque élément de L, et entre chaque élément de R et la cible t.

On définit les capacités c des arêtes de G' de la façon suivante :
- pour toute arête (s, u), $u \in L$, $c(s, u) = w(u)$,
- pour toute arête (v, t), $v \in R$, $c(v, t) = w(v)$,
- pour toute arête $(u, v) \in E$, $c(u, v) = \infty$.

L'étude de la coupe minimum d'un graphe biparti G nous permet de résoudre le problème de couverture de sommets dans le graphe G, défini ci-dessous.

Définition 4.12 (*Problème de couverture de sommets*)

Etant donné un graphe $G = (V, E)$, trouver un sous-ensemble de sommets $S \subseteq V$ de cardi-

nalité minimum, tel que pour chaque arête $(u, v) \in E$, $u \in S$ ou $v \in S$.

Rappelons qu'un ensemble S de sommets est une couverture de sommets si son complémentaire $\overline{S} = V \backslash S$ est un ensemble indépendant. Ainsi une couverture de sommets de poids minimal a pour complémentaire un ensemble indépendant de poids maximum. Pour obtenir l'ensemble indépendant de poids maximum, il nous faut donc résoudre le problème de couverture de sommets sur le graphe biparti G et en prendre le complémentaire. Et pour résoudre le problème de couverture de sommets, il nous suffit de trouver la coupe minimum dans G', comme énoncé dans le Lemme suivant :

Lemme 4.13

La capacité de la coupe minimum dans G' est égale au poids de la couverture de sommets de poids minimum dans G.

Démonstration : Prouvons tout d'abord qu'à une couverture de sommets de G, C, correspond une $(s-t)$-coupe $(S, \overline{S})$ dans G' de capacité $w(C)$:

Posons $S = \{s\} \cup (L \backslash C) \cup (R \cap C)$. Alors,

- pour tout $u \in L \cap C$, (s, u) est une arête de la coupe de capacité $w(u)$,
- pour tout $v \in R \cap C$, (v, t) est une arête de la coupe de capacité $w(v)$,
- tout autre arête partant de s ou arrivant en t ne fait pas partie de la coupe,
- pour toute arête $(u, v) \in E$ telle que $u \in S$, $v \in S$ aussi, car C est une couverture de sommets.

Ainsi, la capacité de la $(s-t)$-coupe $(S, \overline{S})$ est bien égale au poids de la couverture de sommets, $w(C)$.

Réciproquement, prouvons qu'à une $(s-t)$-coupe $(S, \overline{S})$ de G' de capacité finie correspond une couverture de sommets C de G, de poids $c(S, \overline{S})$.

D'abord, une telle coupe ne possède pas d'arête dans E, puisque ces arêtes ont une capacité infinie. Prenons alors $C = (L \backslash S) \cup (R \cap S)$. Alors,

- pour tout sommet $u \in L \backslash S$, (s, u) est une arête de la coupe de capacité $w(u)$,
- pour tout sommet $v \in R \cap S$, (v, t) est une arête de la coupe de capacité $w(v)$,
- aucune autre arête ne fait partie de la coupe,
- toutes les arêtes de G ont une extrémité dans C, car les arêtes de G ont leur deux extrémités dans le même ensemble, S ou T.

Ainsi, pour construire une couverture des sommets de G de poids minimal, il suffit de trouver une $(s-t)$-coupe de capacité minimum dans le graphe G' correspondant, et, pour chaque arête (s, u) (resp. (v, t)) de cette $(s-t)$-coupe, mettre le sommet u (resp. v) dans la couverture de sommets. ■

Pour obtenir l'ensemble indépendant de poids maximum dans G, il suffit de prendre le complémentaire dans V de la couverture précédemment obtenue.

Pour terminer la preuve du Lemme 4.9, il convient de noter que l'Algorithme 8 s'exécute bien en temps polynomial.

Par ailleurs, cet algorithme teste les lentilles de chaque paire de sommets. Pour chaque paire ainsi testée, le groupe pouvant avoir cette paire de sommets comme diamètre est donc testée. Ainsi on teste bien toutes les groupes possibles. ■

Notons au passage que dans le plan, la norme L_1 est identique à la norme L_∞ à une rotation près, et que donc les résultats sont identiques qu'on utilise L_1 ou L_∞.

4.2.3 Conséquence dans le plan muni de la norme L_∞, sans augmentation de ressources

En reprenant le Théorème 4.14 et le Lemme 4.9, on obtient la preuve de l'existence d'un algorithme d'approximation pour BCCD dans le plan.

Theorème 4.14

L'Algorithme 7 appliqué à une instance où l'espace métrique est le plan muni de la norme L_∞ est un $(\frac{1}{11}, 1)$-algorithme d'approximation s'exécutant en temps polynomial pour BCCD.

Démonstration : Dans le plan muni de la norme L_∞, les boules sont des carrés. Dans cet espace, $\gamma = 9$.

Lemme 4.15

Dans le plan muni de la norme L_∞, noté $\mathcal{P}_\infty$, $\gamma(\mathcal{P}_\infty) = 9$.

Démonstration : Considérons une boule B de $\mathcal{P}_\infty$ de diamètre 3. Définissons :

- $x_g = \inf\{x_i, (x_i, y_i) \in B\}$,
- $x_d = \sup\{x_i, (x_i, y_i) \in B\}$,
- $y_b = \inf\{y_i, (x_i, y_i) \in B\}$,
- $x_h = \sup\{y_i, (x_i, y_i) \in B\}$.

On a en particulier $d((x_g, y_b), (x_d, y_h)) \leq \max((x_g - x_d), (y_h - y_b)) \leq 3$, et de même pour $d((x_g, y_h), (x_d, y_b))$. Soit le point c_B de coordonnées $c_B = (\frac{x_g + x_d}{2}, \frac{y_b + y_h}{2})$, la boule $B(c, 1.5)$ contient les éléments de coordonnées $(x_g, y_b), (x_d, y_h), (x_d, y_b), (x_g, y_b)$, et contient donc tous les éléments de B. Couvrir cette boule par des boules de rayon 1/2 dans $\mathcal{P}_\infty$ se fait trivialement en utilisant exactement 9 boules. ■

L'application du Théorème 4.14 nous fournit le facteur d'approximation, et le Lemme 4.9 appliqué au plan muni de la norme L_∞ nous assure que le temps d'exécution de cet algorithme est polynomial. ■

4.2.4 Avec augmentation de ressources, et une norme quelconque

Contrairement à l'étude de BCCD dans un espace métrique quelconque, travailler avec de l'augmentation de ressources dans le plan n'est pas nécessaire. Cependant, elle nous permet d'obtenir de meilleurs résultats d'approximation.

En particulier, il est possible d'adapter l'Algorithme 1 du Paragraphe 3.3 en utilisant l'Algorithme 8 pour créer des groupes de diamètre $d_{\max}$, puis $3d_{\max}$ (au lieu de $2d_{\max}$ et $4d_{\max}$ respectivement), et obtenir ainsi l'Algorithme 9.

Dans une première phase cet algorithme construit, de façon économe, des groupes de diamètre $d_{\max}$ de telle sorte qu'au terme de cette phase il n'existe plus de groupe valide de diamètre $d_{\max}$ à

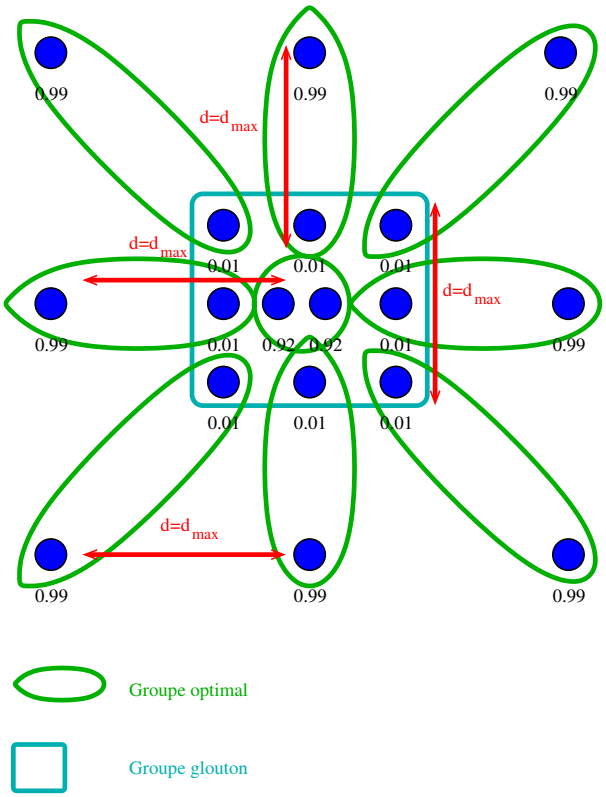


FIG. 4.7 – Un facteur d'approximation de 9 est atteint dans cet exemple.

partir des éléments non utilisés. Dans une seconde phase, il construit des groupes de diamètre $3d_{\max}$ de telle sorte qu'au terme de son exécution il n'existe plus de groupe valide de diamètre $3d_{\max}$.

Theorème 4.16

L'Algorithme 9 appliqué à une instance où l'espace métrique est le plan muni d'une norme quelconque est un $(\frac{2}{5}, 3)$-algorithme d'approximation s'exécutant en temps polynomial pour BCCD.

Démonstration : La preuve est exactement la même que celle de l'Algorithme 1 du Paragraphe 3.3. Le Lemme 4.9 nous assure que l'Algorithme 8 permet d'identifier un groupe valide s'il en existe, quelle que soit la norme qu'on utilise. La seule chose qui change est l'augmentation de ressources nécessaire à la construction exhaustive des groupes de diamètre $d_{\max}$, puis $3d_{\max}$. Ici, aucune augmentation de ressources supplémentaire n'est nécessaire à cette exhaustivité, tandis que dans un graphe quelconque il était nécessaire de construire des groupes de diamètre $2d_{\max}$ pour être certains de construire tous les groupes de diamètre $d_{\max}$. Il en est de même pour les groupes de diamètre $3d_{\max}$ qu'on peut construire sans avoir recours à une augmentation de ressources de 4. Le facteur d'approximation reste donc le même, seule l'augmentation de ressources est diminuée. ■

Ce résultat s'étend à tout espace métrique dans lequel il est possible de construire en temps polynomial des groupes de diamètre $d_{\max}$ jusqu'à ce qu'il n'existe plus de groupe valide d'un tel

Algorithme 9 $(\frac{2}{5}, 3)$-algorithme d'approximation glouton pour BCCD dans le plan muni d'une norme quelconque.

1: $U \leftarrow S$ // Eléments qui n'appartiennent à aucun groupe

Phase 1 :

1: **tant que** l'Algorithme 8 examinant les lentilles de diamètre $d_{\max}$ renvoie "groupe construit" **faire**
2: exécuter l'Algorithme 8 pour construire un groupe B
3: **tant que** il existe $e \in B$ tel que $w(B) - w(e) \geq 1$ **faire**
4: supprimer e de B
5: **fin tant que**
6: supprimer de U les éléments appartenant à un des groupes construits.
7: **fin tant que**

Phase 2 :

1: appliquer la Phase 1 en examinant les lentilles de diamètre $3d_{\max}$.

diamètre, puis de faire la même chose avec des groupes de diamètre $3d_{\max}$. Il suffit donc que dans un tel espace métrique un algorithme permettant la détection et la construction de groupes valides s'exécutant en temps polynomial existe.

En particulier dans les espaces dans lesquels la résolution du problème de trouver une clique de poids maximum sur le graphe $\mathrm{Comp}(\mathcal{I}, d)$, $d > 0$ se fait en temps polynomial, ce résultat s'applique. Par exemple, si $\mathrm{Comp}(\mathcal{I}, d)$ fait partie d'une classe de graphes où ce problème est solvable. Dans un tel cas, tant qu'une telle clique dans $\mathrm{Comp}(\mathcal{I}, d_{\max})$ est de poids supérieur à 1, construire un groupe composé de certains éléments de cette clique, de poids inférieur à 2, puis recommencer de même sur $\mathrm{Comp}(\mathcal{I}, 3d_{\max})$. Le facteur d'approximation reste le même.

Il convient cependant de prendre garde au fait que sur de tels espaces métriques, les résultats d'inapproximabilité ne sont plus toujours vrais. Ainsi le résultat précédent nous assure de l'existence d'un $(\frac{2}{5}, 3)$-algorithme d'approximation dans ce type d'espace, mais aucun résultat, *a priori*, ne nous apprend que BCCD n'est pas approximable à facteur constant dans ce cadre.

4.3 Un $(1/3, 4)$-algorithme d'approximation distribué pour BCCD dans $\mathbb{R}^D$ muni de la norme L_∞

Dans ce paragraphe nous considérons l'espace $\mathbb{R}^D$ muni de la norme L_∞. L'objectif est, ayant restreint l'espace métrique considéré, d'adapter l'algorithme centralisé présenté dans le Paragraphe 3.3 pour obtenir un algorithme distribué de complexité raisonnable. L'algorithme obtenu est un $(\frac{1}{3}, 4)$-algorithme d'approximation. Il s'appuie sur une structure de skip-graph, dont les performances sont étudiées dans le Paragraphe 4.4, nous assurant que notre algorithme distribué, dans le cadre du modèle défini dans le Paragraphe 1.2.3, s'exécute en $O(\log n)$ étapes avec forte probabilité.

4.3.1 Structurer nos éléments

La skip-list

On utilise une structure de *skip-list*. Une skip-list [Pug90, MPS92] est une structure de données ordonnée basée sur la succession de listes chainées ayant un nombre d'éléments décroissant géométriquement. Il existe des variantes déterministes [MPS92] et aléatoires [Pug90] des skip-lists. La version déterministe a des performances garanties, tandis que la version randomisée n'offre que des performances avec forte probabilité. L'effet d'équilibrage de charge introduit avec la variante nommée *skip-graph* [AS03, AS07] n'est valable que pour la version probabiliste. Nous utilisons dans la suite des skip-lists randomisées, puis des skip-graphs. On suppose qu'il est possible de maintenir une telle structure dans un environnement distribué. La construction d'une skip-list d'une manière distribuée se fait d'une façon similaire à celle d'un skip-graph, et nous invitons le lecteur intéressé par le sujet de la construction et du maintien d'une structure de données de ce type dans un environnement distribué à consulter l'article de Aspnes et Shah qui étudie ce type de problématique pour les skip-graphs [AS07]. En ce qui concerne la structure d'une skip-list, on en donne ici un bref aperçu, ainsi qu'un exemple dans la Figure 4.8. Une définition plus précise de cette structure est donnée dans le Paragraphe 4.4, lorsque nous étudions certaines propriétés du skip-graph.

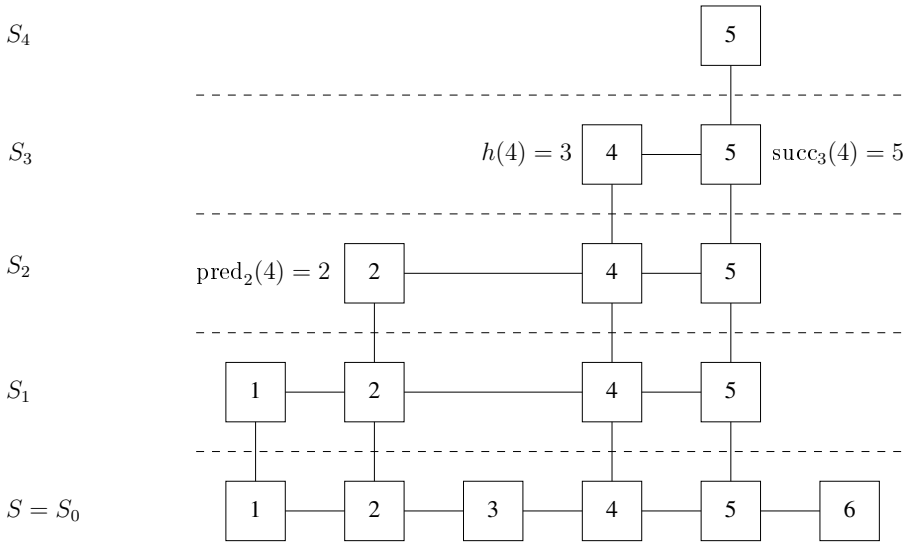


FIG. 4.8 – Une skip-list construite à partir d'un ensemble de 6 éléments.

Etant donné un ensemble S d'éléments totalement ordonnés, une skip-list $\mathcal{S}$ est définie par une suite décroissante $S = S_0 \supseteq S_1 \supseteq \cdots \supseteq S_{L(\mathcal{S})}$ d'ensembles finis d'éléments de S, où L est le *niveau* de la skip-list $\mathcal{S}$. Chacun des S_i est ici représenté comme une liste doublement chainée. On note $h(x)$ l'indice maximal de la liste S_i dans laquelle x apparait, c'est son niveau dans $\mathcal{S}$. On utilise ensuite la notation $\mathrm{pred}_{S_i}(x)$ pour indiquer le prédécesseur de x au niveau i, et $\mathrm{succ}_{S_i}(x)$ pour indiquer son successeur à ce niveau, pour chaque niveau $i \leq h(x)$.

Dans les skip-lists que nous utilisons ici, on associe indépendamment à chaque élément $x \in S$ une variable aléatoire géométrique de paramètre $1/2$, G_x. L'élément x apparait alors dans les G_x premiers ensembles $S_0, \ldots, S_{G_x - 1}$.

Chemins de recherche dans les skip-lists Le *chemin de recherche* pour un élément x dans une skip-list $\mathcal{S}$, noté $P_{\mathcal{S}}(x)$, représente l'ensemble des cellules à examiner lors de la recherche de x. Sa longueur est notée $|P_{\mathcal{S}}(x)|$.

Pour chaque niveau $0 \leq i \leq L(\mathcal{S})$ et chaque élément $x \in S_i$, la cellule de S_i pour l'élément $x = x_m$ est représentée par un noeud de coordonnées (m, i). On ajoute des noeuds sentinelles $(0, i)$ et $(n+1, i)$ à chaque niveau. Ainsi, il existe des pointeurs depuis chaque noeud (m, i) vers $(m, i-1)$ et (m', i) où m' est le rang dans S_0 de $\text{succ}_{S_i}(x_m)$, *i.e.* , $\text{succ}_{S_i}(x_m) = x_{m'}$. La Figure 4.9 représente une skip-list construite à partir d'un ensemble de 6 éléménts, à laquelle ont été ajouté les noeuds sentinelles 0 et $n+1$ à chaque niveau.

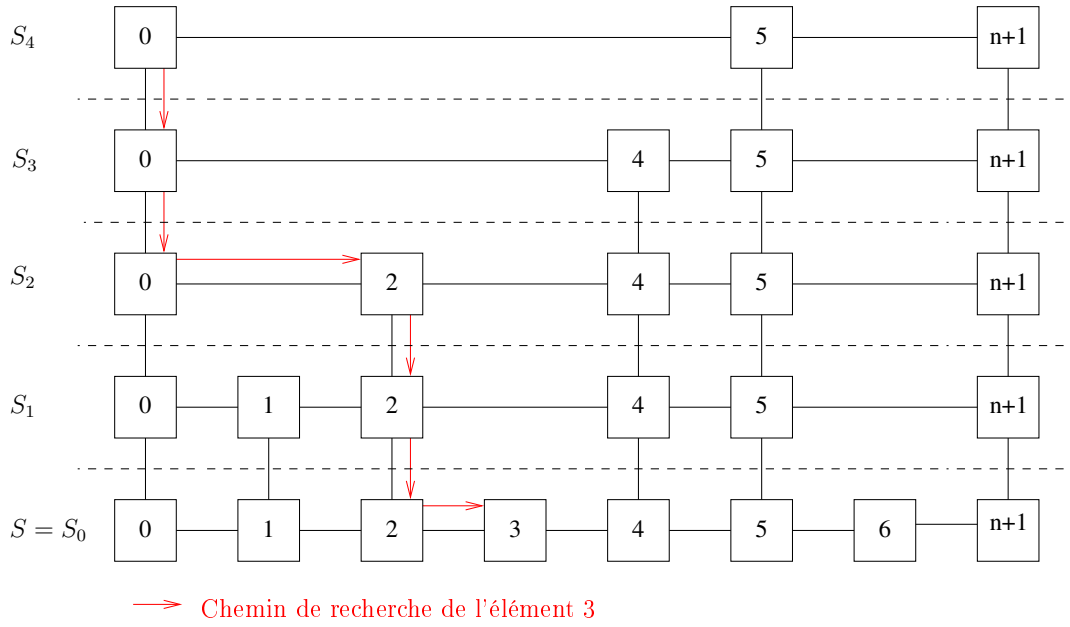


FIG. 4.9 – La skip-list de la figure 4.8 à laquelle ont été ajoutés les noeuds sentinelles.

Le chemin de recherche d'un élément x dans une skip-list $\mathcal{S}$ part de $(0, L(\mathcal{S}))$ et se termine à l'élément x de S_0 (cf. Figure 4.9 pour un exemple). Si le i-ème noeud est (m, l), le noeud suivant (m', l') est défini comme suit :

- si $\text{succ}_{S_l}(x_m) > x$, $(m', l') = (m, l-1)$ (à moins que $l = 0$, auquel cas le noeud suivant est $(m+1, 0)$ et le chemin se termine) ;
- sinon, $l' = l$ et m' est le rang dans S_0 de $\text{succ}_{S_l}(x_m)$.

La longueur maximale d'un chemin de recherche dans une skip-list $\mathcal{S}$, appelée hauteur de la skip-list, est notée $H(\mathcal{S})$. Pour réutiliser les notations de Devroye, nous appellons E_i la longueur de la fin du chemin de recherche vers x_i, qui commence au premier noeud de coordonnée positive.

Devroye a prouvé dans [Dev92] le Théorème suivant :

Theorème 4.17 (*D'après [Dev92]*)

La hauteur H_n d'une skip-list aléatoire de paramètre p est telle que $\frac{H_n}{\log_{1/p} n} \to c$ en probabilité, i.e. quand n tend vers l'infini, pour tout $\varepsilon > 0$, $\mathbb{P}(|\frac{H_n}{\log_{1/p} n} - c| > \varepsilon) = 0$, où c est l'unique

solution positive de l'équation

$$x - 1 + (x - 1)\log_{1/p}(x - 1) - x\log_{1/p} x = 0,$$

Arbre associée à une skip-list Etant donnée une skip-list $\mathcal{S}$, l'arbre associé à $\mathcal{S}$ possède un noeud de coordonnées (m, i) pour chaque niveau i dans lequel chaque élément $x = x_m$ apparait. Pour chaque niveau $0 \leq i \leq L(\mathcal{S})$, le noeud $(a, i), a = \min\{m, x_m \in S_i\}$ et le noeud $(b, i), b = \max\{m, x_m \in S_i\}$ sont dupliqués (virtuellement) et servent de noeuds sentinelles, de coordonnées respectivement $(0, i)$ et $(n + 1, i)$ à chaque niveau.

Chaque noeud (m, i) a alors pour fils les noeuds de coordonnées $(p, i-1)$ tels que $(p, i-1)$ se trouve entre $(m, i-1)$ (compris) et $\text{succ}_{S_i}(x_m)$ (exclus) dans le Z-ordre des éléments. On peut trouver un exemple d'un tel arbre dans la Figure 4.10, associé à la skip-list de la Figure 4.8.

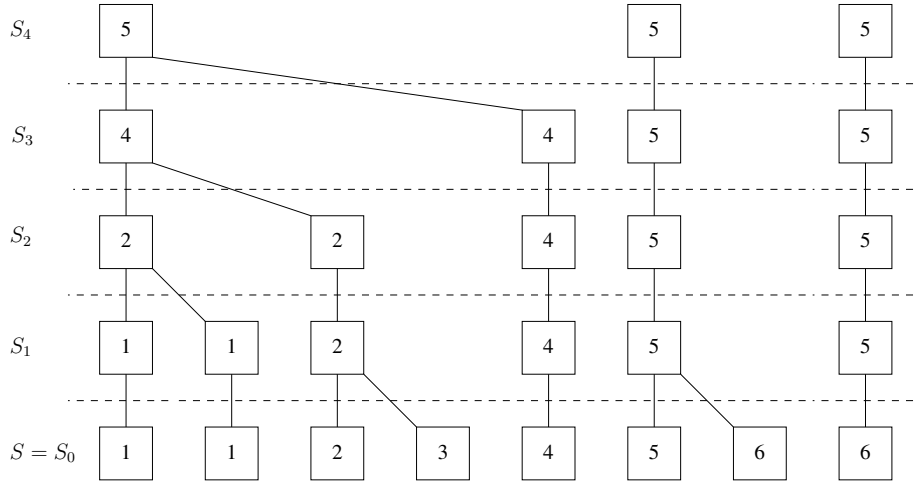


FIG. 4.10 – L'arbre associé à la skip-list de la Figure 4.8.

Ordonner nos éléments : le Z-ordre

Pour utiliser une skip-list, il est nécessaire de travailler à partir d'un ensemble totalement ordonné d'éléments. Comme nous considérons que les éléments sont placés dans un espace D-dimensionnel, nous les ordonnons selon le *Z-ordre* [Ore90] (ou ordre de Morton) de leurs coordonnées. Notons que tout ce qui est dit dans la suite est également vrai si on utilise un ordre de Hilbert pour ordonner les éléments, ou tout ordre vérifiant la Propriété 4.19 présentée ci-dessous.

Par ailleurs, par souci de simplicité des algorithmes et des explications que nous présentons, nous représentons les coordonnées des points de façon binaire, dans $[0, 1[^D$.

Dans l'espace considéré on peut voir une boule de rayon r comme le produit d'intervalles $[a_1 - r, a_1 + r] \times \cdots \times [a_D - r, a_D + r]$. L'espace qu'on considère peut alors être pavé par de telles boules d'intérieurs disjoints.

Définition 4.18 (*Z-ordre*)

Considérons un ensemble d'éléments de coordonnées dans $[0, 1[^D$, pour D fixé. Soient deux

éléments $\mathbf{x} = (x_1, \ldots, x_D)$ *et* $\mathbf{y} = (y_1, \ldots y_D)$ *de cet ensemble tels que :*
- $\forall\ 1 \leq k \leq D, (x_k^{(i)})_{i \geq 1}$ *est le développement binaire de* x_k *où* $\forall\ i \geq 1, x_k^{(i)} \in \{0,1\}$,
- *de même,* $\forall\ 1 \leq k \leq D, (y_k^{(i)})_{i \geq 1}$ *est le développement binaire de* y_k *où* $\forall\ i \geq 1, y_k^{(i)} \in \{0,1\}$,

On définit φ *la fonction d'entrelacement des coordonnées des éléments* $\mathbf{x}$ *de* $[0,1[^D$ *telle que* $\varphi(\mathbf{x}) = (x_1^{(1)}, x_2^{(1)}, \ldots, x_D^{(1)}, x_1^{(2)}, \ldots)$. *On dit que* $\mathbf{x}$ *est inférieur à* $\mathbf{y}$ *suivant le Z-ordre de leurs coordonnées si et seulement si* $\varphi(\mathbf{x}) \underset{lex}{\leq} \varphi(\mathbf{y})$ *(où* $\underset{lex}{\leq}$ *est la relation d'ordre lexicographique classique).*

Le Z-ordre est obtenu en entrelaçant les représentations binaires des D coordonnées de chaque point, et en utilisant l'ordre lexicographique des chaînes ainsi obtenues (voir Figure 4.13). On utilise principalement la Propriété 4.19 concernant la préservation de la localité du Z-ordre.

Propriété 4.19

Si $\mathbf{x} = (x_1, \ldots, x_D)$ *est un point dont les coordonnées sont de la forme* $x_i = a_i/2^k$ *pour des entiers* $a_i < 2^k$, *alors le produit des intervalles* $[x_1, x_1 + 2^{-k}[\times \cdots \times [x_D, x_D + 2^{-k}[$ *est un intervalle dans le Z-ordre.*

Démonstration : Etant donné $\mathbf{x} = (x_1, \ldots, x_D)$ un point dont les coordonnées sont de la forme $x_i = a_i/2^k$ pour des entiers a_i, le produit des intervalles $[x_1, x_1 + 2^{-k}[\times \cdots \times [x_D, x_D + 2^{-k}[$ est composé exactement des D-uplets de coordonnées $y_i, 1 \leq i \leq D$ dont les k premiers bits sont les mêmes que ceux des x_i, donc l'entrelacement des coordonnées de ces points ont leurs kD premiers bits qui sont ceux de l'entrelacement des coordonnées de $\mathbf{x}$. Ils forment donc un intervalle suivant la relation de Z-ordre. ■

Définition des boules utilisées

Etant donné $\mathbf{x} = (x_1, \ldots, x_D)$ un point dont les coordonnées sont de la forme $x_i = a_i/2^k$ pour des entiers a_i, le produit des intervalles $[x_1, x_1 + 2^{-k}[\times \cdots \times [x_D, x_D + 2^{-k}[$ est appelé une *boule de niveau* k (voir la Figure 4.12). Le point $\mathbf{x}$ est appelé le *coin* de cette boule.

Remarquons qu'une boule de niveau k est en fait, au bord près, une boule de $[0,1[^D$, de centre le point de coordonnées $\mathbf{x} + 2^{-(k+1)} = (x_1 + 2^{-(k+1)}, \ldots, x_D + 2^{-(k+1)})$ et de rayon $2^{-(k+1)}$. Le bord d'une boule de niveau k n'est pas totalement inclus dans celle-ci, contrairement au bord de la boule $B(\mathbf{x} + 2^{-(k+1)}, 2^{-(k+1)})$. Pour pouvoir utiliser la Propriété 4.19, nous souhaitons que les boules que nous définissons nous permettent de travailler sur des intervalles du Z-ordre, d'où la non-inclusion des bords dans celles-ci. Intuitivement cependant il est plus clair de parler de boules centrées, dont on sous-entend l'exclusion d'une partie du bord.

Par définition, chaque boule de niveau k est l'union (disjointe) de 2^D boules de niveau $(k+1)$ (voir Figure 4.11). En conséquence, $[0,1[^D$ est l'union disjointe de toutes les 2^{kD} boules de niveau k.

Etant donné un entier k, définissons une *boule décalée de niveau* k comme une boule de rayon $2^{-(k+1)}$ dont le centre est de la forme $(a_i/2^{k+1})_{1 \leq i \leq D}$ pour des entiers a_i, et une *boule doublement décalée de niveau* k comme une boule de rayon $2^{-(k+1)}$ dont les coordonnées du centre sont de

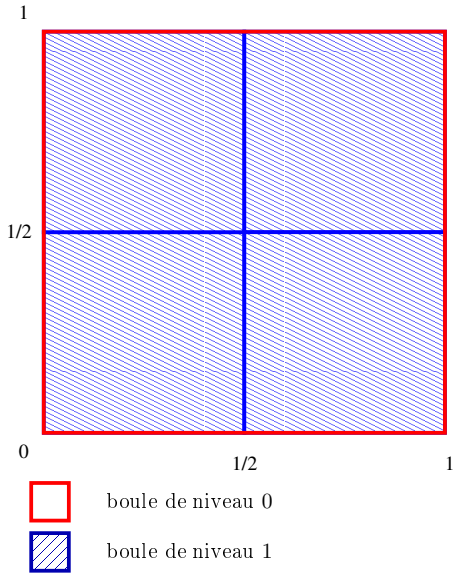


FIG. 4.11 – Une boule de niveau 0 est l'union disjointe de 2^D boules de niveau 1 (ici $D = 2$). $[0, 1[^2$ est l'union disjointe de 2^2 boules de niveau 1.

la forme $b_i/2^{k+2}$, pour des entiers b_i (voir aussi la Figure 4.12). Les boules de niveau k sont exactement les boules décalées de niveau k pour lesquelles tous les a_i sont impairs, et les boules décalées de niveau k sont exactement les boules doublement décalées de niveau k pour lesquelles les b_i sont pairs.

Remarquons par ailleurs qu'une boule décalée de niveau k est toujours l'union de 2^D boules de niveau $(k+1)$, et, ainsi, dans le Z-ordre, l'union d'au plus 2^D intervalles disjoints. De façon similaire, une boule doublement décalée de niveau k est l'union de 4^D boules de niveau $(k+2)$, et, ainsi, l'union d'au plus 4^D intervalles disjoints du Z-ordre.

De plus, l'ensemble des points à distance au plus $2^{-(k+1)}$ d'une boule de niveau k forme une boule décalée de niveau $(k-1)$, et l'ensemble des points à distance au plus $2^{-(k+1)}$ d'une boule décalée de niveau k forme une boule doublement décalée de niveau $(k-1)$.

Dans la suite nous considérons $d_{\max} \leq 1$ de la forme $d_{\max} = \frac{a}{2^k}$ pour des entiers a et k. Nous utilisons le lemme suivant pour prouver la correction de l'algorithme que nous présentons :

Lemme 4.20

Soit un ensemble $E \in [0, 1[^D$ de diamètre inférieur à $d = a/2^k$ pour un entier a tel que $2^h \leq a < 2^{h+1}$ pour un entier h, alors il existe une boule de niveau $(k - 1 - h)$ qui contient E.

Démonstration : Posons $E = \{X^{[i]} = (x_1^{(i)}, \ldots, x_D^{(i)})\}$.

Posons $y_1 = \min_{1 \leq i \leq |E|} x_1^{(i)}, \ldots, y_D = \min_{1 \leq i \leq |E|} x_D^{(i)}$. Pour tout i et tout j on a alors $y_j \leq x_j^{(i)} \leq y_j + d$.

Pour tout i, posons $y'_i = \frac{\lfloor 2^k y_i \rfloor}{2^k}$. Le point $\mathbf{y}'$ de coordonnées $(y'_1, \ldots, y'_D)$ est le coin d'une boule de niveau $(k - 1 - h)$. Cette boule contient E. Clairement dans un premier temps, tout élément $\mathbf{x}$ de E est tel que pour tout i, $x_i \geq y'_i$. Par ailleurs, pour tout i, on

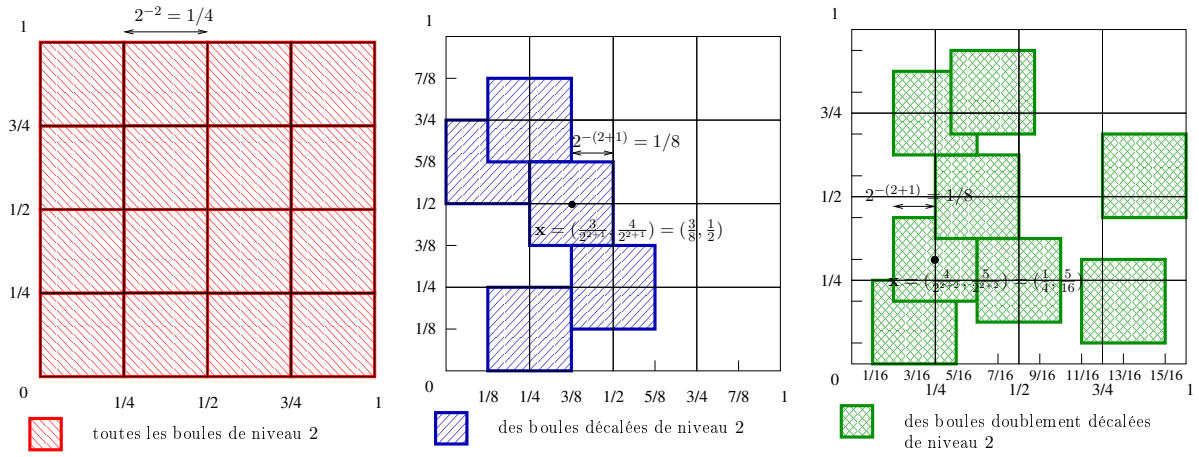


FIG. 4.12 – Exemples de boules et de boules décalées

a :

$$y'_i \geq \frac{2^k y_i - 1}{2^k} \geq y_i - \frac{1}{2^k}$$

Donc

$$x_i \leq y_i + \frac{a}{2^k} \leq y'_i + \frac{1}{2^k} + \frac{a}{2^k}$$

Comme $a < 2^{h+1}$ et que a et h sont entiers, $a + 1 \leq 2^{h+1}$, d'où :

$$x_i \leq y'_i + \frac{1}{2^{k-(h+1)}}$$

et donc tout élément $\mathbf{x}$ est inclus dans la boule de niveau $(k-1-h)$ et de coin $\mathbf{y}'$. ■

4.3.2 Une adaptation distribuée du $(2/5, 4)$-algorithme d'approximation

Pour construire une version distribuée de l'Algorithme 1 présenté dans le Paragraphe 3.3, il suffit de trouver un moyen de construire de façon gloutonne, tant qu'il en existe, les groupes de poids supérieur à 1 dans toutes les boules décalées de niveau $(k-1)$ (Phase 1), puis, tant qu'il en existe, dans toutes les boules doublement décalées de niveau $(k-2)$ (Phase 2), où $d_{\max} = 2^{-k}$. La Phase 0 de l'algorithme correspond à une préphase qui permet la construction de groupes dans chaque boule de niveau k, qui sont des intervalles du Z-ordre, ce qui y simplifie leur construction. Une fois que plus aucun groupe valide n'existe dans une boule de niveau k, il reste à gérer les éléments restant dans chacune de ces boules, ce qui se fait de manière distribuée en Phase 1, puis en Phase 2.

A ceci près qu'on n'effectue pas, à l'intérieur de chaque groupe construit, la phase de suppression des éléments inutiles effectué par l'Algorithme 1, l'Algorithme 11 suit un déroulement similaire et construit des groupes au sein duquel l'un des éléments est nommé comme chef, pour, par exemple, une gestion ultérieure du groupe lui-même. Nous démontrons que l'Algorithme 11 est un $(1/3, 4)$-algorithme d'approximation distribué pour BCCD.

Algorithme 10 Construction gloutonne de groupes dans un intervalle $[a, b)$

Pour chaque élément x :

si $p(\text{pred}_{S_0}(x)) < a$ **alors**
  calculer la hauteur maximale $H(a, b)$ dans l'intervalle $[a, b)$ en utilisant l'Algorithme 12
  changer la hauteur de x dans la skip-list pour la valeur $H(a, b)$
fin si
$\ell \leftarrow -1$, $c \leftarrow x$, $W \leftarrow w(x)$
pour i **de** 0 **à** $h(x)$ **faire**
  si $h(\text{succ}_{S_i}(x)) = i$ **et** $p(\text{succ}_{S_i}(x)) < b$ **alors**
    /* Rappelons que $h(y)$ est la hauteur maximale à laquelle y apparait dans la skip-list, ainsi en général $h(\text{succ}_{S_i}(x)) \geq i$ */
    recevoir le message $(\textsc{Poids}, i, y, w)$ de $\text{succ}_{S_i}(x)$
    $c \leftarrow y$, $W \leftarrow W + w$, $\ell \leftarrow \ell + 1$
    si $W \geq 1$ **alors**
      envoyer le message $(\textsc{ChefdeGroupe}, W)$ à l'élément y
      appeler la procédure $\textsc{CréerGroupe}(y, i)$
    fin si
  fin si
fin pour
envoyer le message $(\textsc{Poids}, h(x), c, W)$ à $\text{pred}_{S_{h(x)}}(x)$
recevoir le message $(\textsc{Groupe}, y, h(x))$ et appeler la procédure $\textsc{CréerGroupe}(y, h(x))$

Procédure $\textsc{CréerGroupe}(y, i)$:

si x n'appartient pas encore à un groupe **alors**
  envoyer le message $(\textsc{GrouperElement}, x, w(x))$ à y
fin si
pour j de ℓ à i **faire**
  si $h(\text{succ}_{S_j}(x)) = j$ **et** $p(\text{succ}_{S_j}(x)) < b$ **alors**
    envoyer le message $(\textsc{Groupe}, y, j)$ à $\text{succ}_{S_j}(x)$
  fin si
fin pour
$\ell \leftarrow i + 1$, $W \leftarrow 0$

A la réception d'un message $(\textsc{ChefdeGroupe}, W)$:

attendre les messages $(\textsc{GrouperElement}, x, w)$ jusqu'à atteindre un poids total des w égal à W ; les x correspondant sont les éléments du groupe.

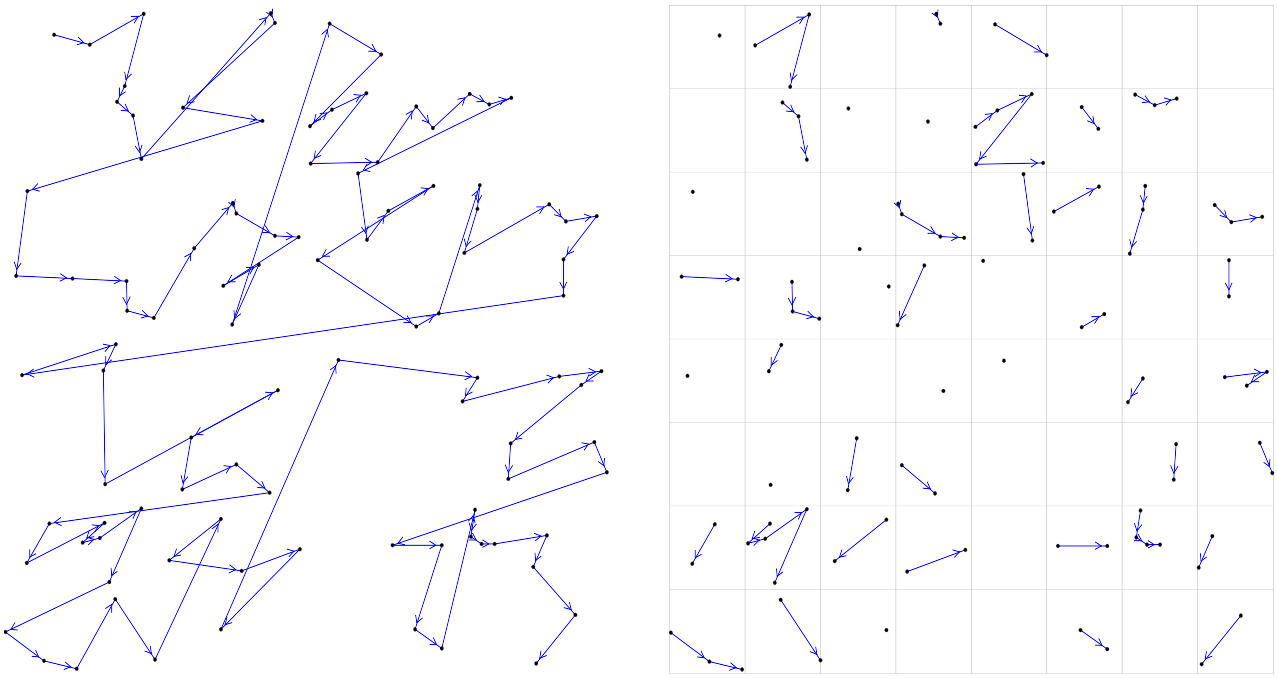


FIG. 4.13 – Figure de gauche : exemple de Z-ordre dans le plan. Figure de droite : boules de niveau 3 : chacune de ces boules correspond à un intervalle du Z-ordre.

Algorithme 11 $(\frac{1}{3}, 4)$-algorithme d'approximation distribué pour BCCD

Phase 0 :

1: **pour** chaque boule de niveau k **faire**
2: construire des groupes de façon gloutonne en utilisant l'Algorithme 10 sur l'intervalle du Z-ordre correspondant à cette boule.
3: **fin pour**

Phase 1 :

1: **pour** chaque boule décalée de niveau $(k-1)$ **faire**
2: construire des groupes de façon gloutonne en utilisant l'Algorithme 10 sur l'intervalle correspondant à cette boule décalée.
3: **fin pour**

Phase 2 :

1: **pour** chaque boule doublement décalée de niveau $(k-2)$ **faire**
2: construire des groupes de façon gloutonne en utilisant l'Algorithme 10 sur l'intervalle correspondant à cette boule doublement décalée.
3: **fin pour**

Détail de la Phase 0

Chaque élément x calcule les coordonnées de la boule de niveau k (donc, de diamètre $d_{\max}$) à laquelle il appartient. Ensuite, chaque élément applique l'Algorithme 10 de façon à construire des groupes dans chaque boule de niveau k.

Cet algorithme utilise la structure de skip-list pour construire des groupes gloutons dans cette boule ; comme on travaille à l'intérieur d'une boule, la contrainte de distance n'a plus à être prise en compte. Chaque élément x, de poids $w(x)$, possède des coordonnées dans $\mathbb{R}^D$, ainsi qu'une hauteur $h(x)$ dans la skip-list (le niveau maximal parmi les listes dans lesquelles il apparait), et une position $p(x)$ correspondant à son indice dans la liste ordonnée (selon le Z-ordre) sur laquelle est construite la skip-list. On suppose qu'il détient ces informations à propos de chacun des voisins qu'il possède à chaque niveau de la skip-list, *i.e.* pour tous les éléments dans $\{\text{pred}_{S_i}(x), i \leq h(x)\} \cup \{\text{succ}_{S_i}(x), i \leq h(x)\}$.

Dans la suite de notre raisonnement, on utilise l'arbre associé à la skip-list considérée. Dans l'Algorithme 10, chaque élément remonte l'arbre en maintenant le poids total W des éléments non groupés dans son sous-arbre, associé à un niveau ℓ sous lequel tous les éléments de son sous-arbre font déjà partie d'un groupe. La plupart des messages utilisés par l'algorithme sont en fait envoyés depuis un élément vers un de ses voisins dans l'arbre.

Il y a deux types de messages :

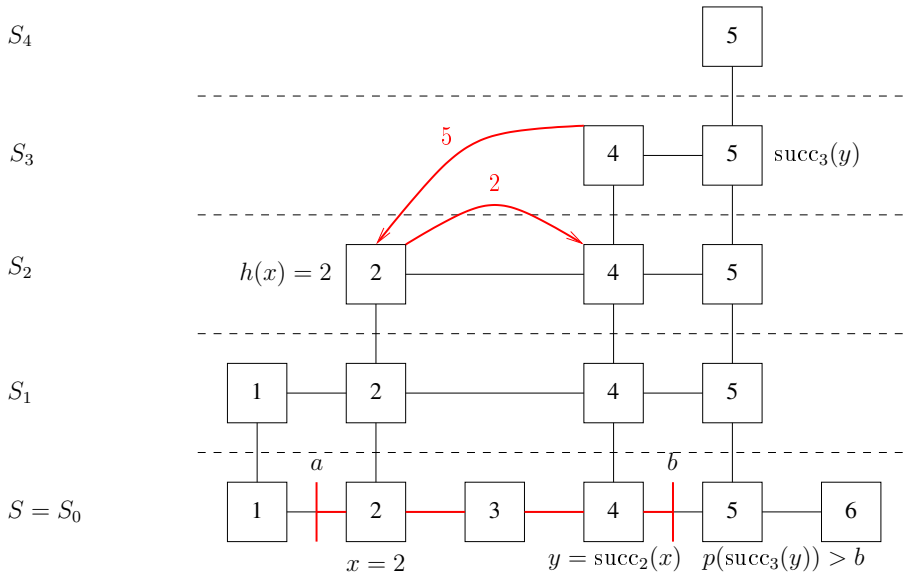
- Les messages de type POIDS, qui indiquent un niveau i, un candidat y, et un poids w. Le sens d'un tel message est que dans l'arbre de hauteur i enraciné à l'émetteur, il y a un poids total d'éléments non groupés d'une valeur w. L'élément y est l'un des éléments non groupés, et est utilisé comme chef de groupe si ce poids est utilisé.
- Les messages de type GROUPE, qui indiquent un niveau i et un chef de groupe y. Ils sont envoyés en réponse aux messages de type POIDS, et signifient que tous les éléments non groupés dans l'arbre de hauteur i enraciné au récepteur du message sont membres d'un groupe commun, qui a y pour chef.

Durant cette phase, des groupes vont être construits dans chaque boule de niveau k. A la fin de cette phase, le premier élément de l'intervalle correspondant à chaque boule connait la valeur du poids qui n'a pas été groupé dans cette boule. Cet élément est appelé le *coordinateur* de cette boule.

Algorithme 12 Calcul de la hauteur maximale dans un intervalle $[a, b)$

L'élément x est supposé être le premier de l'intervalle $[a, b)$, et s'envoie un message (HAUTEUR, x, b) pour démarrer l'exécution.
Pour chaque élément y, lors de la réception d'un message (HAUTEUR, x, b) :

1: **si** $p(\text{succ}_{S_{h(y)}}(y)) \geq b$ **alors**
2: envoyer le message (RÉSULTAT-HAUTEUR, $b, h(y)$) à l'élément x
3: **sinon**
4: envoyer le message (HAUTEUR, x, b) à l'élément $\text{succ}_{S_{h(y)}}(y)$
5: **fin si**



Exécution :

1 : début : $h(x) = 2$ et son successeur à ce niveau, y, est dans $[a, b)$,

2 : x envoie (HAUTEUR, x, b) à y, son successeur au niveau 2,

3 : y a une hauteur plus élevée, $h(y) = 3$,

4 : le successeur de y au niveau 3 n'est pas dans $[a, b)$,

5 : alors y envoie (RÉSULTAT-HAUTEUR, $b, h(y)$) à x.

FIG. 4.14 – Exécution de l'Algorithme 12

Détail de la Phase 1

Durant cette phase, toutes les boules décalées de niveau $(k-1)$ vont être parcourues à la recherche de poids restant. Comme tout ensemble de diamètre inférieur à $d_{\max}$ est inclus dans une de ces boules décalées, il ne reste aucun groupe valide d'éléments non groupés à la fin de cette phase.

Définissons une boule de niveau k et une boule décalée de niveau $(k-1)$ associée à un point donné :

Définition 4.21 (*Boules associées à une boule de niveau k*)

Etant donné un point $\mathbf{x} = (x_1, \ldots, x_D)$ *dont les coordonnées sont de la forme* $x_i = a_i/2^k$ *pour des entiers* a_i *:*

- *la boule de niveau* k *associée à* $\mathbf{x}$ *est l'unique boule de niveau* k *dont* $\mathbf{x}$ *est le coin,*
- *l'unique boule décalée de niveau* $(k-1)$*,* $B(\mathbf{x} + 2^{-k}, 2^{-k})$*, où* $\mathbf{x} + 2^{-k}$ *est le point* $(x_1 + 2^{-k}, \ldots, x_D + 2^{-k})$*, est sa boule décalée de niveau* $(k-1)$ *associée.* $\mathbf{x}$ *en est le coin.*
- *l'unique boule doublement décalée de niveau* $(k-2)$*,* $B(\mathbf{x} + 2^{-(k-1)}, 2^{-(k-1)})$*, où* $\mathbf{x} + 2^{-(k-1)}$ *est le point* $(x_1 + 2^{-(k-1)}, \ldots, x_D + 2^{-(k-1)})$*, est sa boule doublement décalée de niveau* $(k-2)$ *associée.* $\mathbf{x}$ *en est le coin.*

Cette définition est illustrée dans la Figure 4.15 où un point $\mathbf{x} = (\frac{1}{4}, \frac{1}{4})$, sa boule associée

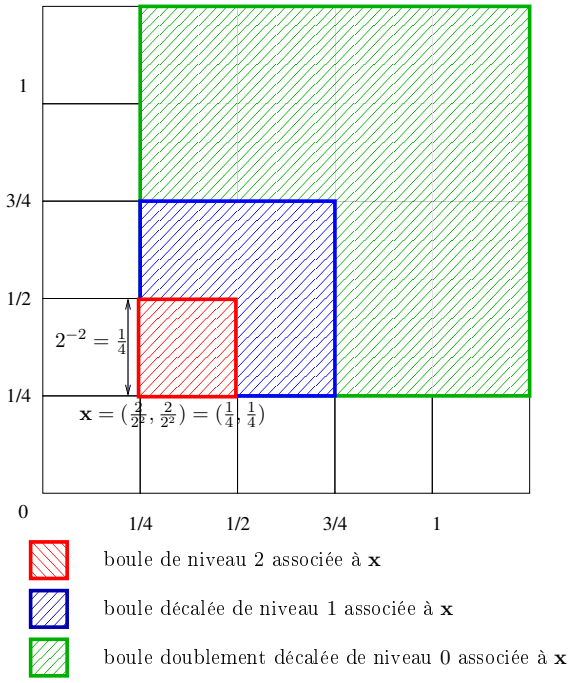


FIG. 4.15 – Un élément $\mathbf{x} = (\frac{1}{4}, \frac{1}{4})$, sa boule de niveau 2 associée, sa boule décalée de niveau 1 associée, et sa boule doublement décalée de niveau 0 associée.

de niveau 2, sa boule décalée de niveau 1 et sa boule doublement décalée de niveau 0 sont représentées. On peut, en utilisant cette définition, associer à chaque boule de niveau k une boule décalée unique de niveau $(k-1)$ et une boule doublement décalée de niveau $(k-2)$.

Remarque 4.22

Par définition, les boules décalées associées aux points $\mathbf{x}(a_1, \ldots, a_D) = (x_1 + a_1 * 2^{-(k-1)}, \ldots, x_D + a_D * 2^{-(k-1)})$, *pour* $a_i \in \mathbb{Z} \cap [(-x_i)2^{k-1}, (1-x_i)2^{k-1}[, 1 \leq i \leq D$ *ne s'intersectent pas (la contrainte sur les* a_i *correspond juste à* $\mathbf{x} \in [0,1[^D$*). L'ensemble de boules ainsi défini contient au plus* $2^{(k-1)D}$ *boules décalées qui ne s'intersectent pas, car* $\mathbf{x} \in [0,1[^D$.

De même, les boules doublement décalées associées aux points $\mathbf{x}(b_1, \ldots, b_D) = (x_1 + b_1 * 2^{-(k-2)}, \ldots, x_D + b_D * 2^{-(k-2)})$, *pour* $b_i \in \mathbb{Z} \cap [(-x_i)2^{k-2}, (1-x_i)2^{k-2}[, 1 \leq i \leq D$ *ne s'intersectent pas (la contrainte sur les* b_i *correspond juste à* $\mathbf{x} \in [0,1[^D$*). L'ensemble de boules ainsi défini contient au plus* $2^{(k-2)D}$ *boules décalées qui ne s'intersectent pas, car* $\mathbf{x} \in [0,1[^D$.

Définition 4.23 (*Vecteur de parité de niveau k*)

Soit $\mathbf{x} = (x_1, \ldots, x_D)$ *tel que pour tout* i, $x_i = a_i/2^k$ *pour des entiers* a_i*, on définit le vecteur de parité* $vp(\mathbf{x})$ *par* $vp(\mathbf{x}) = (p_1, \ldots, p_D)$ *où pour tout* i, $p_i = 0$ *si* a_i *est pair, et* $p_i = 1$ *si* a_i *est impair.*

Un point $\mathbf{x} = (x_1, \ldots, x_D)$ a même vecteur de parité de niveau k que le point $\mathbf{y} = (x_1 + 2/2^k, \ldots, x_D) = (x_1 + 2^{-(k-1)}, \ldots, x_D)$ mais pas le même que $\mathbf{z} = (x_1 + 1/2^k, \ldots, x_D)$.

On peut alors définir un ensemble de points ayant même vecteur de parité de niveau k. Soit

un vecteur de parité vp de niveau k, on définit $c(vp) = \{\mathbf{x} \in [0,1[^D, vp(\mathbf{x}) = vp\}$. Suivant cette définition, $c(vp)$ contient $2^{(k-1)D}$ points. $c(vp)$ est l'ensemble des $2^{(k-1)D}$ coins de boules de niveau k de parité vp. En particulier, et en utilisant la remarque 4.22, ces coins sont également des coins de boules décalées de niveau $(k-1)$ ne s'intersectant pas. On peut donc définir, via l'utilisation des vecteurs de parité, 2^D ensembles de boules décalées de niveau $(k-1)$, de telle sorte qu'au sein de chacun de ces ensembles les $2^{(k-1)D}$ boules ne s'intersectent pas deux à deux.

Pour chaque vecteur de parité $vp = (vp_1, \ldots, vp_D)$ on note $id(vp) = vp_1 + 2vp_2 + \cdots + 2^{D-1}vp_D$. On crée une partition des boules décalées de niveau $(k-1)$ en 2^D ensembles $S_0, \ldots, S_{2^D-1}$ de façon à ce que chaque ensemble S_i contienne toutes les boules décalées de niveau $(k-1)$ ayant pour vecteur de parité de niveau k le vecteur vp tel que $id(vp) = i$.

Deux boules du même sous-ensemble ne se chevauchent pas. Chaque boule de niveau k est par ailleurs incluse dans exactement 2^D boules décalées de niveau $(k-1)$, une dans chacun des sous-ensembles S_i. La Phase 1 examine alors toutes les boules décalées de niveau $(k-1)$ en examinant en parallèle chacune des boules contenues dans chacun des S_i. Chaque ensemble S_i sera traité l'un après l'autre, puisque les boules s'intersectent d'un ensemble à l'autre, nécessitant 2^D exécutions du même type de tâche sur chaque ensemble S_i.

Plus précisément l'idée de la Phase 1 est alors de centraliser toutes les informations concernant les poids des éléments d'une boule décalée de niveau $(k-1)$ en un élément particulier. Cet élément peut décider quels groupes peuvent être construits. Par ailleurs on peut désigner dans chaque boule décalée un *coordinateur* qui reçoit et traite les informations correspondantes aux éléments de la boule[4]. Il est alors possible d'examiner le contenu de toutes les boules décalées de niveau $(k-1)$ de façon distribuée.

Pour cela, cette phase va examiner tous les 2^D ensembles S_i les uns après les autres. Pour un ensemble S_i donné, tous les chefs de boules de niveau k vont transmettre le poids restant dans leur boule au coordinateur de la boule de niveau $(k-1)$ de l'ensemble S_i à laquelle ils appartiennent. Si le poids total est supérieur à 1, ce coordinateur construit un ou plusieurs groupes (chacun de poids strictement inférieur à 2) et les messages GROUPE correspondant à ces constructions sont envoyés. Notons que le poids restant dans chaque boule n'est jamais fractionné : soit il est utilisé en totalité, soit il ne l'est pas. En particulier, on ne peut pas avoir recours au rafinement utilisé dans l'Algorithme 1 qui consiste à supprimer de chaque groupe les éléments qui en sont inutiles. On se contente d'assurer que le poids de chaque groupe ne dépasse pas 2, ce qui signifie que quand on ajoute les éléments restant dans une boule à un groupe, ni le groupe auquel on les ajoute, ni la somme des éléments eux-mêmes, n'a un poids supérieur à 1 (sinon on aurait déjà construit un groupe).

Le coordinateur peut être atteint en utilisant une procédure de recherche standard dans la skip-list, puisque sa position est connue au sein d'une boule décalée donnée (comme dit précédemment, il peut s'agir par exemple du premier élément de la plus petite boule de niveau k qu'elle contient). Effectuer toutes les recherches en parallèle peut se réveler couteux pour les éléments se trouvant aux plus hauts niveaux de la skip-list. C'est pourquoi nous utilisons le skip-graph [AS07] évoqué plus haut, qui assure un bon équilibrage de charges dans ce genre de situation.

[4] Un bon exemple pourrait être l'élément le plus petit dans le Z-ordre parmi les éléments qu'elle contient.

Détail de la Phase 2

Cette phase est très similaire à la Phase 1. Pour chaque point $\mathbf{x}$ on définit son vecteur de double parité :

Définition 4.24 (*Vecteur de double parité de niveau k*)

Soit $\mathbf{x} = (x_1, \ldots, x_D)$ tel que pour tout i, $x_i = a_i/2^k$ pour des entiers a_i, on définit le vecteur de double parité $vp(\mathbf{x})$ par $vp(\mathbf{x}) = (p_1, \ldots, p_D)$ où pour tout i, p_i est le reste de la division entière de x_i par 4.

Un point $\mathbf{x} = (x_1, \ldots, x_D)$ a même vecteur de parité de niveau k que le point $\mathbf{y} = (x_1 + 4/2^k, \ldots, x_D) = (x_1 + 2^{-(k-1)}, \ldots, x_D)$ mais pas le même que $\mathbf{z} = (x_1 + 2/2^k, \ldots, x_D)$.

On peut alors définir un ensemble de points ayant même vecteur de double parité de niveau k. Soit un vecteur de parité vp de niveau k, on définit $c(vp) = \{\mathbf{x} \in [0, 1[^D, vp(\mathbf{x}) = vp\}$. Suivant cette définition, $c(vp)$ contient $2^{(k-2)D}$ points. $c(vp)$ est l'ensemble des $2^{(k-2)D}$ coins de boules de niveau k de double parité vp. En particulier, et en utilisant la remarque 4.22, ces coins sont également des coins de boules doublement décalées de niveau $(k-2)$ ne s'intersectant pas. Ainsi on définit de la même façon que précédemment 4^D ensembles de boules doublement décalées de niveau $(k-2)$, de telle sorte qu'au sein de chacun de ces ensembles les $2^{(k-2)D}$ boules ne s'intersectent pas deux à deux.

Pour chaque vecteur de parité $vp = (vp_1, \ldots, vp_D)$ on note $id(vp) = vp_1 + 4vp_2 + \cdots + 4^{D-1}vp_D$. De la même façon que dans la Phase 1, on crée une partition des boules doublement décalées de niveau $(k-2)$ en 4^D ensembles $S_0, \ldots, S_{4^D-1}$ de façon à ce que chaque ensemble S_i contienne toutes les boules doublement décalées de niveau $(k-2)$ ayant pour vecteur de double parité de niveau k le vecteur vp tel que $id(vp) = i$. Deux boules du même sous-ensemble ne se chevauchent pas.

A présent, en s'appuyant sur le Lemme 4.20, on sait que tout groupe de diamètre inférieur à $3d_{\max}$ est inclus dans une boule décalée de niveau $(k-2)$, pour $d_{\max} = 2^{-k}$.

On examine donc toutes les boules doublement décalées de niveau $(k-2)$ en considérant en parallèle chacune des boules contenues dans chacun des ensembles S_i. Chaque ensemble S_i sera traité l'un après l'autre, puisque les boules s'intersectent d'un ensemble à l'autre, nécessitant 4^D exécutions du même type de tâche sur chaque ensemble S_i.

La tâche effectuée sur chaque boule suit le même principe qu'en Phase 1, de telle sorte qu'au terme de l'exécution de l'Algorithme 11 il n'existe pas d'ensemble d'éléments non groupés de poids total supérieur à 1 et de diamètre inférieur à $3d_{\max}$.

4.3.3 Analyse de complexité

Theorème 4.25

L'Algorithme 10, exécuté en parallèle sur un nombre m d'intervalles disjoints, utilise $O(n + m \log n)$ messages et, dans un modèle d'exécution synchrone dans lequel chaque message prend un temps unitaire, $O(\log n)$ cycles, en moyenne et avec forte probabilité.

Démonstration : Durant l'exécution de l'Algorithme 10, chaque élément envoie un seul message de type POIDS. Chaque message de type GROUPE d'un élément x à un élément

y est précédé d'un message de type POIDS de y à x, donc le nombre total de messages de type GROUPE est aussi au plus n.

Le nombre de messages utilisés durant chaque exécution de l'Algorithme 12, et également nécessaires au changement de hauteur résultant de cette exécution, est majoré par la hauteur renvoyée, qui est $O(\log n)$ en moyenne et avec forte probabilité. Ainsi, le nombre total de messages échangés durant l'exécution de l'Algorithme 10 sur m intervalles disjoints est $O(n + m \log n)$, en moyenne et avec forte probabilité.

Dans chaque intervalle, les messages de type POIDS sont relayés le long d'un niveau de la skip-list, jusqu'à atteindre un élément appartenant à un niveau plus élevé de la skip-list. Le chemin que parcourt chacun de ces messages correspond à un chemin de recherche dans une skip-list. La longueur de ces chemins de recherche a été étudiée par Devroye dans [Dev92], qui a obtenu une caractérisation très précise de leur longueur (on y revient dans le Paragraphe 4.4). Ici, il suffit de savoir qu'ils ont une longueur de $O(\log n)$ avec forte probabilité pour conclure. ■

Durant les phases ultérieures, très peu de messages sont envoyés, exceptés les messages de recherche des coordinateurs de boules dans la skip-list. La complexité moyenne (et, ici, le nombre de messages nécessaires) d'une recherche dans une skip-list de taille n est en $O(\log n)$, et on effectue $O(m)$ recherche, pour un total de $O(m \log n)$ messages ; la plus longue des recherches étant de longueur $O(\log n)$ d'après Devroye [Dev92]. Ainsi le Théorème 4.25 s'étend à l'Algorithme 11 plus global de construction des groupes de manière distribuée.

4.3.4 Analyse du facteur d'approximation

Theorème 4.26

L'Algorithme 11 est un $(\frac{1}{3}, 4)$-algorithme d'approximation distribué pour BCCD, qui fonctionne, dans un modèle d'exécution synchrone dans lequel chaque message prend un temps unitaire, en $O((4^D) \log n)$ cycles et utilise $O((4^D) n \log n))$ messages en moyenne et avec forte probabilité.

Démonstration : Le Paragraphe 4.3.3 permet de conclure quant à la complexité de l'Algorithme 11. En effet, le nombre d'intervalles disjoints sur lesquels l'Algorithme 10 est exécuté en parallèle est majoré par 4^D, le nombre d'ensembles de boules doublement décalées nécessaires à l'exécution de la Phase 2.

Concernant l'analyse du facteur d'approximation, il est possible de reprendre la preuve du Théorème 3.14. En reprenant les mêmes notations et les mêmes raisonnements, on choisit une solution optimale économe, notée OPT_{BCCD}. Une telle solution optimale existe toujours, puisqu'il suffit d'en prendre une qui n'est pas économe, et d'ôter de chaque groupe ses éléments inutiles.

On note $\mathcal{B}^{(k)}$, $k = 1, 2$ l'ensemble des groupes construits durant la Phase k, avec $b_k = |\mathcal{B}^{(k)}|$.

Dans la suite, partitionnons les groupes $B_i^{(1)}$, $1 \leq i \leq b_1$ de $\mathcal{B}^{(1)}$ en deux sous-

ensembles $\mathcal{L}$ et $\mathcal{M}$, en fonction du nombre d'éléments de chaque groupe :

$$\begin{cases} \mathcal{L} = \{B_i^{(1)} \in \mathcal{B}^{(1)},\ \ |B_i^{(1)}| = 2\},\ \ l = |\mathcal{L}| \text{ et } \textsc{Opt}_L \text{ correspond aux groupes} \\ \text{de } \textsc{Opt}_{BCCD} \text{ qui intersectent des groupes de } \mathcal{L}, \\ \mathcal{M} = \mathcal{B}^{(1)} \setminus \mathcal{L},\ m = b_1 - l \text{ et } \textsc{Opt}_M \text{ correspond aux groupes de} \\ \textsc{Opt}_{BCCD} \setminus \textsc{Opt}_L \text{ qui intersectent des groupes de } \mathcal{M}. \end{cases}$$

On réutilise alors l'équation (4.1) : comme tous les groupes de $\textsc{Opt}_{BCCD}$ intersectent un groupe de Phase 1, on a

$$|\textsc{Opt}_{BCCD}| \leq |\textsc{Opt}_L| + |\textsc{Opt}_M|. \tag{4.1}$$

Par ailleurs, comme tous les groupes de $\mathcal{L}$ ont exactement deux éléments, et que les groupes de la solution optimale sont disjoints, alors chaque groupe de $\mathcal{L}$ intersecte au plus deux groupes de $\textsc{Opt}_L$. D'où $|\textsc{Opt}_L| \leq 2l$.

Ce qui est différent, c'est que le Lemme 3.18 ne s'applique plus, puisque la phase de suppression des éléments inutiles de chaque groupe n'a pas lieu dans l'Algorithme 11.

On peut cependant définir le poids perdu de la même façon que dans la preuve du Théorème 3.14 dans le but de majorer la cardinalité de $\textsc{Opt}_M$:

Définition 4.27 (*Poids perdu*)

Etant donnée instance $\mathcal{I} = (S, d, w, d_{\max})$ de BCCD, et une solution construite sur $\mathcal{I}$ par l'Algorithme 11, w_{perdu} est le poids total des éléments qui n'appartiennent à aucun groupe de cette solution, mais qui appartiennent à un groupe de $\textsc{Opt}_M$.

On utilise pour la suite de la preuve le Lemme 4.28

Lemme 4.28

$w_{perdu} < m$

Démonstration : Dans la zone étendue de chaque groupe de $\mathcal{M}$, le poids total des éléments qui n'appartiennent à aucun des groupes construits par l'Algorithme 11 est strictement inférieur à 1. En effet, si ce n'était pas le cas, un autre groupe aurait été construit en Phase 2. Comme tout élément appartenant à un groupe de $\textsc{Opt}_M$ se trouve dans la zone étendue d'un des groupes de $\mathcal{M}$, $w_{\text{perdu}} < m$. ■

Par ailleurs on a l'équation suivante régissant la cardinalité de $\textsc{Opt}_M$:

$$|\textsc{Opt}_M| \leq w(\textsc{Opt}_M) \leq w(\mathcal{M}) + w(\mathcal{B}^{(2)}) + w_{\text{perdu}} \tag{4.2}$$

Cependant, comme chaque groupe B de $\mathcal{M}$ a un poids strictement inférieur à 2, on a l'équation suivante :

$$|\textsc{Opt}_M| < 2m + 2b_2 + m = 3m + 2b_2$$

Et donc,

$$|\textsc{Opt}_{BCCD}| < 3m + 2b_2 + 2l < 3(b_1 + b_2),$$

ce qui permet nous permet d'obtenir le facteur d'approximation annoncé. ■

4.4 Amélioration de l'efficacité du $(1/3, 4)$-algorithme d'approximation en utilisant un skip-graph

Dans le Paragraphe 4.3, nous présentons un algorithme distribué pour résoudre de manière approchée BCCD. Cet algorithme utilise une skip-list. La complexité de l'algorithme présenté dépend fortement de la hauteur de la skip-list. Cette hauteur H_n est la longueur maximale d'un chemin de recherche vers n'importe quelle clé depuis le sommet de la skip-list (ici on parle de "clé" pour plus de généralité, mais dans notre application chaque élément est associé à une clé unique). Devroye a étudié précisément cette hauteur dans [Dev92] et a prouvé qu'elle est d'ordre $\log n$ (voir le Paragraphe 4.4.1 pour de plus amples détails).

Dans ce paragraphe nous nous intéressons à une autre structure de données : le skip-graph. Cette structure, introduite par Aspnes et Shah dans [AS03, AS07], est une variante de la skip-list, prévue pour fournir de meilleurs résultats que celle-ci dans un environnement distribué. Dans un skip-graph, l'intégralité de la structure de donnée peut être distribuée entre un grand nombre de noeuds. Par ailleurs cette structure offre de bonnes propriétés d'équilibrage de charge ainsi que de tolérance aux pannes. Cette structure a été bien étudiée [AW05, AS07], et plusieurs variantes du skip-graph ont été examinées, comme les skip-nets [HJS$^+$03], les skip-webs [AEG05] ou les *rainbow* skip-graphs [GNS06]. Afin de se représenter le skip-graph comme une structure de données distribuée, on peut considérer chaque clé comme un processeur d'un système distribué, ou bien que chaque processeur stocke un petit nombre de clés et les pointeurs associés. Dans notre cas, chaque clé unique correspond à un élément unique d'une instance de BCCD, qui correspond lui-même à un noeud unique du réseau à grande échelle dans lequel est appliqué notre algorithme. Le skip-graph fait office de réseau de communication, dans lequel chaque noeud a pour voisin ses voisins dans le skip-graph.

Notre intérêt pour cette structure de données peut paraître assez naturel : puisque nous essayons de construire un algorithme distribué pour BCCD, pourquoi ne pas utiliser une variante de la structure de données utilisée, adaptée à ce type d'environnement ? Le problème est que, comme dit précédemment, la complexité de cet algorithme repose fortement sur la hauteur d'une skip-list. Ainsi la complexité du même algorithme utilisant un skip-graph en lieu et place d'une skip-list dépendrait fortement de la hauteur dudit skip-graph.

Dans un skip-graph de taille n (qui s'appuie sur un ensemble de n clés distinctes), n'importe quelle clé peut être trouvée au sein de n skip-lists différentes (mais pas indépendantes), entrainant un aussi grand nombre de chemins de recherche. On définit alors la hauteur H'_n d'un skip-graph aléatoire comme la longueur maximale parmi tous les chemins de recherche dans le skip-graph. Le résultat que nous présentons est une première borne supérieure sur la longueur maximale d'un chemin de recherche dans un skip-graph, en étendant aux skip-graphs les résultats démontrés par Devroye sur les skip-lists.

On peut assez simplement observer que la hauteur H'_n d'un skip-graph est $O(\log^2 n)$, puisque le nombre de niveaux d'un skip-graph est d'ordre $\log n$ avec forte probabilité et que le nombre d'éléments à un niveau donné sur un chemin de recherche donné peut être d'ordre $\log n$. Le résultat que nous démontrons assure que cette hauteur est en fait d'ordre $\log n$.

Dans la suite, nous présentons tout d'abord, dans le Paragraphe 4.4.1, le modèle que nous utilisons pour décrire les skip-lists et les skip-graphs. Nous présentons ensuite une extension d'un résultat de majoration de Devroye pour les skip-lists aléatoires de paramètre $p = 1/2$, dans

le Paragraphe 4.4.2. Enfin nous appliquons ce résultat pour obtenir une majoration avec forte probabilité de la hauteur d'un skip-graph, dans le Paragraphe 4.4.3, et comparons nos résultats théoriques à quelques simulations dans le Paragraphe 4.4.4.

4.4.1 Modèles et notations

Skip-lists

Nous rappelons ici en la complétant la définition d'une skip-list, que nous avons décrit dans le Paragraphe 4.3.1 concernant la structuration des éléments pour l'élaboration d'un $(1/3, 4)$-algorithme d'approximation dans un environnement distribué.

Une skip-list est définie par une suite décroissante $S_0 \supseteq S_1 \supseteq \cdots \supseteq S_L$ d'ensembles finis de clés issues d'un univers totalement ordonné.

Dans une skip-list, chacun des S_i est représenté comme une liste chainée (simplement ou doublement) ; chaque noeud de S_i stocke un pointeur vers l'élément correspondant de S_{i-1}. Le niveau maximal L d'une skip-list $\mathcal{S}$ est noté $L(\mathcal{S})$. On utilise la notation $\mathrm{succ}_S(x)$ pour indiquer le successeur de l'élément x dans la liste chainée S.

Dans une skip-list *aléatoire* de paramètre p, basée sur un ensemble de clés $S_0 = \{x_1 \leq x_2 \leq \cdots \leq x_n\}$, i est le rang de x_i. On associe indépendamment à chaque clé $x \in S_0$ une variable aléatoire géométrique de paramètre $1-p$, G_x. La clé x apparait dans les G_x premiers ensembles $S_0, \ldots, S_{G_x - 1}$.

Rappelons enfin que la longueur maximale d'un chemin de recherche dans une skip-list $\mathcal{S}$, appelée hauteur de la skip-list, est notée $H(\mathcal{S})$. Pour réutiliser les notations de Devroye, nous appellons E_i la longueur de la fin du chemin de recherche vers x_i, qui commence au premier noeud de coordonnée positive.

A ce sujet, Devroye a prouvé dans [Dev92] le Théorème suivant :

Theorème 4.29 (*D'après [Dev92]*)

La hauteur H_n d'une skip-list aléatoire de paramètre p est telle que $\frac{H_n}{\log_{1/p} n} \to c$ en probabilité, i.e. quand n tend vers l'infini, pour tout $\varepsilon > 0$, $\lim_{n\to\infty} \mathbb{P}(|\frac{H_n}{\log_{1/p} n} - c| > \varepsilon) = 0$, où c est l'unique solution positive de l'équation

$$x - 1 + (x-1)\log_{1/p}(x-1) - x \log_{1/p} x = 0,$$

Skip-graphs

Dans un skip-graph basé sur un ensemble de clés $S_0 = \{x_1 \leq \cdots \leq x_n\}$, on associe à chaque clé x un mot binaire uniforme (potentiellement infini), *i.e.* chaque lettre de w_x est obtenue en tirant à pile ou face (sur une pièce honnêtement calibrée !). Pour chaque mot fini w tel qu'au moins une clé ait w comme préfixe de son mot binaire, on considère la liste chainée S^w de ces clés (la liste S^ε est juste la liste chainée S_0 de toutes les clés du skip-graph). Avec probabilité 1, chaque mot w_x a un plus petit préfixe fini qui le sépare de tous les autres mots w_{x_i} ; la longueur

de ce préfixe est le *niveau* de x dans le skip-graph, noté $L(x)$. En pratique, le skip-graph dépend uniquement de ces préfixes, mais pour notre analyse, chaque w_x est considéré comme une chaîne binaire infinie uniforme.

Le skip-graph basé sur l'ensemble de clés S_0 est composé de toutes les listes chainées non vides S^w. Chaque clé $x \in S_0$ apparait dans $1+L(x)$ listes. En pratique, chaque clé doit conserver un pointeur vers ses successeurs dans chacune des listes dans lesquelles il apparait.

Par conséquent, la suite de listes $S_0(x) = S^\varepsilon, S_1(x), \ldots, S_{L(x)}(x)$ dans lesquelles x apparait forme une skip-list (cette skip-list n'est pas distribuée comme une skip-list aléatoire comme définie précédemment ; sa distribution exacte est précisée plus tard). La Figure 4.16 est un exemple d'une instance de skip-graph construit à partir d'un ensemble de 6 éléments.

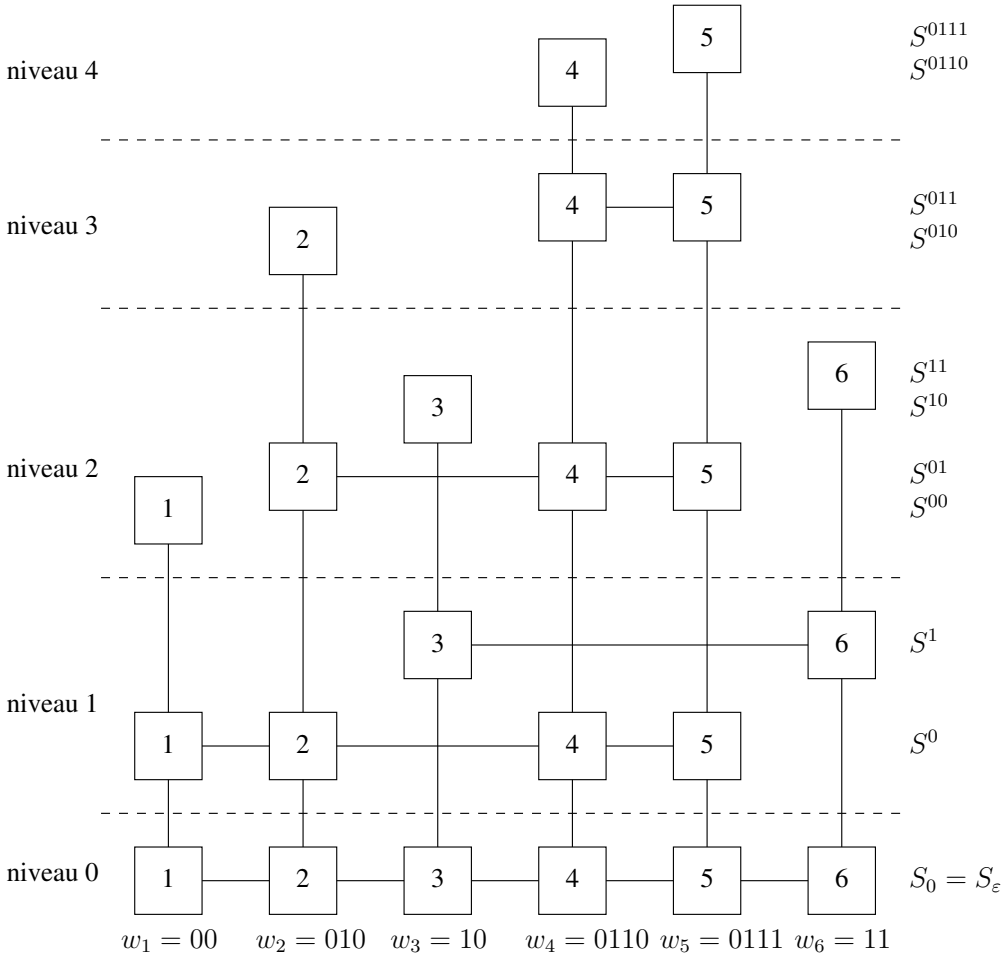


FIG. 4.16 – Un skip-graph construit à partir d'un ensemble de 6 éléments.

Aspnes et Shah ont prouvé dans [AS07] que le niveau maximal d'un skip-graph est en moyenne $O(\log n)$. Ils ont également prouvé que le temps moyen de recherche dans un skip-graph est $O(\log n)$. Dans ce qui suit, nous donnons une majoration du plus long temps de recherche, que nous prouvons être également d'ordre $\log n$.

Chemins de recherche dans les skip-graphs Contrairement au cas de la skip-list, les chemins de recherche dans un skip-graph $\mathcal{G}$ sont définis par une clé de départ x et une clé cible y (dans un environnement distribué où chaque clé correspond à un seul noeud de calcul, ou où chaque noeud se voit associer un petit nombre de clés, x serait une des clés associées au noeud initiant la recherche). Le chemin de recherche $P_{\mathcal{G}}(x, y)$ est simplement le chemin de recherche vers y dans la skip-list $\mathcal{S}(x) = (S_0(x), S_1(x), \ldots, S_{L(x)}(x))$.

La *hauteur* d'un skip-graph $\mathcal{G}$, notée $H'(\mathcal{G})$, est la longueur maximale, parmi toutes les paires de clés (x, y), des chemins de recherche de x à y dans $\mathcal{G}$.

Remarque 4.30

Le lecteur attentif a noté que notre définition d'un chemin de recherche dans un skip-graph n'est pas cohérente avec celle de [AS07], où les skip-graphs sont décrits en utilisant des listes doublement chainées, dans lesquelles la recherche se fait vers l'arrière quand $y < x$, tandis que nous utilisons uniquement une recherche vers l'avant dans une liste simplement chainée. Cela ne fait aucune différence dans la mesure où chercher vers l'avant dans la portion de "clés plus petites que x" dans une skip-list est équivalent à chercher vers l'arrière dans la skip-list basée sur une liste ordonnée dans l'autre sens, et ces deux skip-lists ont la même distribution.

Dans la suite, nous prouvons le théorème suivant, offrant une majoration sur la hauteur d'un skip-graph :

Theorème 4.31

Pour tout ε positif, la hauteur H'_n d'un skip-graph construit à partir d'une liste de n éléments est telle que $\mathbb{P}(H'_n \leq (2c+\varepsilon)\log_2 n) = 1-o(1)$, où $c \simeq 4,403$ est l'unique solution de l'équation

$$x - 1 - x\log_2(x) + (x-1)\log_2(x-1) = 0.$$

4.4.2 Majoration de la hauteur d'une skip-list

Pour prouver le Théorème 4.31, nous prouvons tout d'abord le théorème suivant, qui est une extension des résultats de Devroye dans [Dev92] (Théorème 4.29) :

Theorème 4.32

Pour tout $\alpha \geq 1$ et tout $c > \alpha c_p$, où c_p est l'unique solution positive de l'équation

$$x - 1 + (x-1)\log_{1/p}(x-1) - x\log_{1/p} x = 0,$$

la hauteur H_n d'une skip-list aléatoire (de paramètre p) à n éléments est telle que $\mathbb{P}(H_n > c\log_{1/p} n) = o(n^{1-\alpha})$.

Remarque 4.33

Le cas $\alpha = 1$ du Théorème 4.32, associé à une borne inférieure correspondante, est prouvé dans [Dev92].

Démonstration du Théorème 4.32 : En suivant le schéma de preuve de Devroye, on majore la probabilité d'avoir une grande hauteur. Pour tout k et l, soit le niveau de

$\mathcal{S}$ est plus grand que l, soit l'un des n chemins de recherche est plus grand que k. Comme chaque $E_i, 1 \leq i \leq n$ est stochastiquement inférieur à E_n, on a :

$$\begin{aligned} \mathbb{P}\{H_n > k\} &\leq \mathbb{P}\{L(\mathcal{S}) > l\} + n\mathbb{P}\{E_n > k, L(\mathcal{S}) \leq l\} && (4.3) \\ &\leq \mathbb{P}\{L(\mathcal{S}) > l\} + n\mathbb{P}\left\{\sum_{j=0}^{l} N_j > k\right\}, && (4.4) \end{aligned}$$

où N_j est le nombre de noeuds au niveau j dans le chemin de recherche pour x_n.

On commence par prendre une valeur de l telle que la probabilité $\mathbb{P}(L(\mathcal{S}) > l)$ soit $o(n^{1-\alpha})$: comme le nombre de niveaux d'une skip-list est le maximum de n variables aléatoires géométriques indépendantes de paramètre $1-p$,

$$\mathbb{P}(L(\mathcal{S}) > l) \leq n.p^l,$$

ainsi il est suffisant d'avoir $l = \alpha \log_{1/p} n + \omega(1)$. On prend

$$l = \alpha \log_{1/p} n + \sqrt{\log_{1/p} n}.$$

On va prendre $k = \lceil \theta \log_{1/p} n \rceil$, en choisissant θ de telle sorte que

$$\mathbb{P}\{E_n > k, L(\mathcal{S}) \leq l\} = o(n^{-\alpha}).$$

Soit N_j le nombre de noeuds au niveau j sur le chemin de recherche $P_{\mathcal{S}}(x_n)$. Les variables N_j sont collectivement stochastiquement inférieures à des variables aléatoires géométriques de paramètre p, indépendantes et identiquement distribuées : $\mathbb{P}\{N'_j = i\} = (1-p)^i p$, $i \geq 1$. Ainsi le second terme de l'équation (4.4) peut être majoré comme suit :

$$\mathbb{P}\left\{\sum_{j=0}^{l} N_j > k\right\} \leq \mathbb{P}\left\{\sum_{j=0}^{l} N'_j > k\right\}$$

On utilise à présent la méthode de majoration exponentielle classique de Chernoff : pour tout $0 < t < \ln 1/(1-p)$,

$$\begin{aligned} \mathbb{P}\left\{\sum_{j=0}^{l} N'_j > k\right\} &\leq \frac{\mathbb{E}\left(e^{t\sum_{j=0}^{l} N'_j}\right)}{e^{tk}} \\ &= \frac{\left(\mathbb{E}(e^{tN'_0})\right)^{l+1}}{e^{tk}} \\ &= e^{-tk}\left(\frac{pe^t}{1-(1-p)e^t}\right)^{l+1}. \end{aligned}$$

Pourvu que $\theta > \alpha$, on peut prendre t tel que $e^t = \frac{k-l}{k(1-p)}$. Ce qui nous donne, en posant $u = \frac{l}{k}$:

$$\begin{aligned}\mathbb{P}\left\{\sum_{j=0}^{l} N_j > k\right\} &\leq \left(\frac{k-l}{k(1-p)}\right)^{-k}\left(\frac{p(k-l)}{(1-p)l}\right)^{l+1} \\ &\leq \frac{p(1-u)}{(1-p)u}\left(\left(\frac{p}{u}\right)^{u}\left(\frac{1-p}{1-u}\right)^{1-u}\right)^{k}\end{aligned}$$

Notons que $u \to \frac{\alpha}{\theta}$ quand $n \to \infty$. On a donc

$$\begin{aligned}n\mathbb{P}\left\{\sum_{j=0}^{l} N_j > k\right\} &\leq n\frac{p(1-u)}{(1-p)u}\left(\left(\frac{p}{u}\right)^{u}\left(\frac{1-p}{1-u}\right)^{1-u}\right)^{k} \\ &= \frac{p(1-u)}{(1-p)u} n^{1+\theta(-u\log_{1/p}u + u\log_{1/p}p - (1-u)\log_{1/p}(1-u) + (1-u)\log_{1/p}(1-p)) + o(1)}\end{aligned}$$

Pour obtenir une borne supérieure en $o(n^{1-\alpha})$, il suffit d'avoir (en remplaçant u par sa limite),

$$(\theta-\alpha)\log_{1/p}\left(\frac{\theta-\alpha}{1-p}\right) - \theta\log_{1/p}\theta + \alpha\log_{1/p}\alpha > 0. \tag{4.5}$$

En posant $f_{\alpha,p}(x) = (x-\alpha)\log_{1/p}\left(\frac{x-\alpha}{1-p}\right) - x\log_{1/p}x + \alpha\log_{1/p}\alpha$, on trouve que $f_{\alpha,p}(\alpha x) = \alpha f_{1,p}(x)$, et on peut facilement vérifier que $f_{1,p}$ a un seul zéro positif, et est positive pour $x > c_p$. Ainsi, l'équation (4.5) est équivalente à $\theta > \alpha c_p$. ■

4.4.3 Lien entre skip-lists aléatoires et skip-graphs aléatoires

Pour analyser proprement les skip-graphs aléatoires, il est important de bien comprendre à quoi ressemble l'ensemble des listes $\mathcal{S}(x_i)$ contenant une clé x_i d'un skip-graph. Un exemple est présenté dans la Figure 4.17, montrant la skip-list correspondante à la clé 5 dans le skip-graph de la Figure 4.16.

Ces listes forment une skip-list qui ressemble beaucoup à une skip-list aléatoire de taille n (de paramètre $p = 1/2$), mais la distribution de probabilités la régissant n'est pas exactement la même. En effet, la liste "vue" par l'élément x_i est distribuée exactement comme une skip-list "classique" (de paramètre $p = 1/2$) de taille $n-1$, dans laquelle on aurait inséré une clé de rang i, apparaissant dans un nombre de niveaux égal au nombre de niveaux de la skip-list plus 1. Pour vérifier l'exactitude de cette affirmation, considérons un élément particulier x_i et son mot binaire associé, w_{x_i}. Pour chaque autre élément x_j, la longueur (augmentée de 1) du plus long préfixe commun entre w_{x_i} et w_{x_j} est distribuée géométriquement avec paramètre $1/2$, et ces longueurs sont indépendantes ; ainsi le nombre de listes de $\mathcal{S}(x_i)$ dans lesquelles x_j apparait est distribué géométriquement.

Lemme 4.34

Etant données une paire de skip-lists $(\mathcal{S},\mathcal{S}')$, telle que $\mathcal{S}$ est une skip-list arbitraire de taille

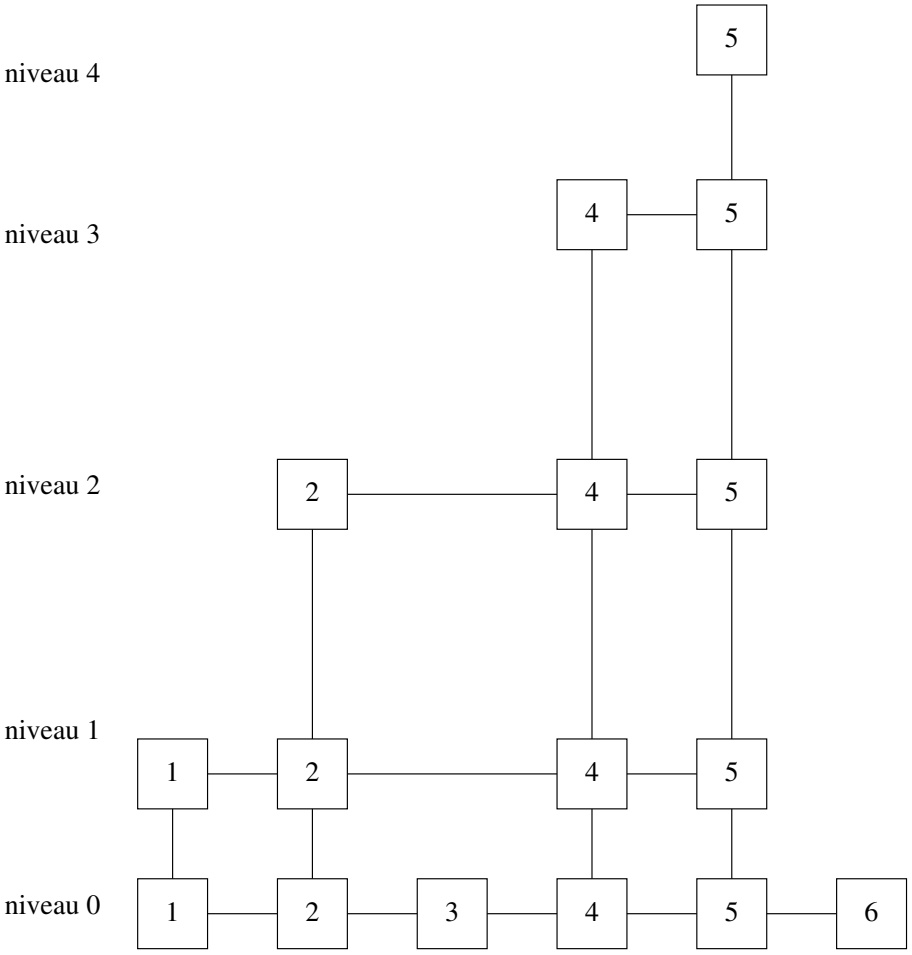


FIG. 4.17 – La skip-list "vue" par l'élément 5 dans le skip-graph de la Figure 4.16

$n-1$, et $\mathcal{S}'$ est obtenue à partir de $\mathcal{S}$ en insérant une n-ème clé x_i à la position i dans la liste de base, apparaissant à un nombre de niveaux égal au nombre de niveaux h de $\mathcal{S}$ plus 1, pour toute clé x, les longueurs des chemins de recherche pour x dans $\mathcal{S}$ et $\mathcal{S}'$ vérifient

$$|P_{\mathcal{S}'}(x)| \leq 2 + |P_{\mathcal{S}}(x)|.$$

Démonstration : Considérons les différentes positions relatives possibles entre x et x_i. Pour simplifier les notations, on associe à x_i la position semi-entière $i - 1/2$ dans $\mathcal{S}'$, de sorte que les clés d'indice plus grand conservent la même position dans $\mathcal{S}$ et dans $\mathcal{S}'$.

Si $x < x_i$, alors les chemins de recherche dans $\mathcal{S}$ et dans $\mathcal{S}'$ sont exactement les mêmes, sauf que le chemin dans $\mathcal{S}'$ commence à un niveau supplémentaire, et on a $|P_{\mathcal{S}'}(x)| = 1 + |P_{\mathcal{S}}(x)|$.

Si $x = x_i$, le chemin de recherche pour x dans $\mathcal{S}'$ est réduit à un noeud par niveau, ce qui est certainement moins que n'importe quel chemin de recherche dans $\mathcal{S}$ plus 1.

Si $x > x_i$, considérons le premier noeud (i', k) de rang $i' \geq i$ dans le chemin de

recherche $P_{\mathcal{S}}(x)$. Alors le chemin de recherche dans $\mathcal{S}'$ est simplement

$$(0, L(\mathcal{S}) + 1), (i - 1/2, L(\mathcal{S}) + 1), (i - 1/2, L(\mathcal{S})), \ldots, (i - 1/2, k)$$

suivi de la fin du chemin de recherche dans $\mathcal{S}$, commençant en (i', k). Encore une fois, le segment initial de ce chemin de recherche utilise seulement un noeud par niveau entre les niveaux k et $L(\mathcal{S})$ (le chemin de recherche dans $\mathcal{S}$ en utilise au moins autant), et deux noeuds au niveaux $L(\mathcal{S}) + 1$, d'où le terme en $+2$ de ce lemme ainsi démontré. ■

Armés de ce Lemme 4.34 et du Théorème 4.32, nous sommes prêts à prouver notre résultat principal.

Démonstration du Théorème 4.31 : Soit $\mathcal{G}$ un skip-graph aléatoire à n clés. Pour que $H'(\mathcal{G})$ soit plus grande que k, au moins une des n skip-lists $\mathcal{S}(x_i)$ doit avoir une hauteur supérieure à k. Si on prend $k \geq c \log_2(n)$ avec $c > 2c_{1/2}$, la combinaison du Lemme 4.34 et du Théorème 4.32 nous assure que, pour tout $1 \leq i \leq n$,

$$\mathbb{P}\{H(\mathcal{S}(x_i)) > k\} = o(1/n),$$

(pour tout i), et donc, par l'inégalité de Boole,

$$\mathbb{P}\left\{H'(\mathcal{G}) > k\right\} = o(1).$$

■

4.4.4 Evaluation expérimentale

Dans le paragraphe précédent, nous avons obtenu une majoration avec forte probabilité sur la hauteur maximale d'un skip-graph de n clés par une constante c fois $\log n$. Dans ce paragraphe nous présentons quelques simulations ayant pour but d'estimer la précision de cette majoration.

La Figure 4.18 montre les fonctions de répartition empirique des hauteurs normalisées de skip-lists et de skip-graphs de tailles différentes ; pour chaque taille d'instance, les simulations ont été effectué sur 100 skip-lists et autant de skip-graphs.

Pour une skip-list aléatoire de paramètre $p = 1/2$, la constante $c_{1/2} \simeq 4,403$ correspond au "sommet" des courbes de la Figure 4.18, même si la taille des instances testées reste relativement petite. Le sommet des courbes relatives aux skip-graphs semble correspondre à une constante plus proche de $5,5$ que de notre borne supérieure théorique de $2c_{1/2} \simeq 8,807$.

Pour les skip-lists, la borne supérieure est associée à une borne inférieure équivalente [Dev92], de telle sorte que la hauteur d'une skip-list H_n, normalisée par $\log_2 n$, tend vers $c_{1/2}$ en probabilité. Nos travaux ne nous ont pas permis de trouver une borne inférieure correspondante pour les skip-graphs (autre que celle existante pour les skip-lists), et nos résultats expérimentaux semblent indiquer que notre majoration n'est pas fine. La question de la convergence en probabilité de la hauteur normalisée d'un skip-graph vers une certaine constante reste donc ouverte. Les techniques que nous avons utilisées n'ont pas grandes chances de donner de meilleurs résultats, et une analyse plus fine des relations de dépendance entre les différentes skip-lists d'un skip-graph semble nécessaire.

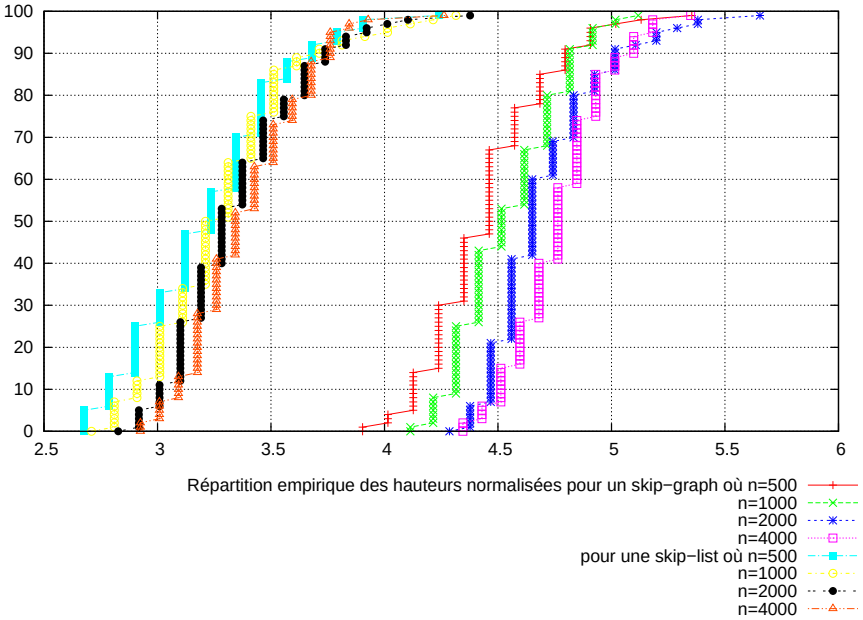


FIG. 4.18 – Fonctions de répartition empirique des hauteurs normalisées de chaque structure (skip-list ou skip-graph)

Cependant ce résultat reste intéressant car il prouve de façon non triviale que la hauteur d'un skip-graph est d'ordre $\log n$ (et non $\log^2 n$ comme on pouvait le penser auparavant). Il est en particulier intéressant dans notre cadre puisqu'il permet de prouver que l'utilisation d'un skip-graph pour structurer les éléments dans le cadre de l'utilisation du $(1/3, 4)$-algorithme d'approximation présenté dans le Paragraphe 4.3 n'engendre pas de surcoût en terme de complexité par rapport à l'utilisation d'une skip-list. En effet la complexité de l'Algorithme 11 dépend de la hauteur de la structure utilisée. La hauteur d'un skip-graph est $O(\log n)$, du même ordre que celle d'une skip-list. Comme cette structure est conçue pour être utilisée dans un environnement distribué, dans le cadre des réseaux à grande échelle dans lequel on se place dans le Paragraphe 4.3, on s'attend à ce que l'utilisation d'un skip-graph offre de meilleures performances générales que celle d'une skip-list.

4.5 BCCD dans un arbre

Dans ce paragraphe nous étudions les résultats qu'il est possible d'obtenir lorsque les éléments d'une instance de BCCD sont dans une métrique d'arbre. Nous présentons dans ce paragraphe un $(\frac{1}{3}, 2)$-algorithme d'approximation pour BCCD dans les arbres, *i.e.* pour les instances $\mathcal{I} = (S, d, w, d_{\max})$ de BCCD pour laquelle un plongement dans un arbre T est donné. Tout élément de S est un noeud de l'arbre T, mais tout noeud de l'arbre T n'est pas nécessairement un élément de S. Ces noeuds "virtuels" se voient affecter un poids nul, pour simplifier la description de l'algorithme que nous présentons. La distance d est la distance dans cet arbre T. Les arêtes de cet arbre sont pondérées, et les valeurs de ces arêtes sont utilisées pour calculer les distances entre les éléments.

Il est important de remarquer que ce résultat utilise une augmentation de ressources d'un

facteur 2 alors qu'aucun résultat d'inapproximabilité ne nous permet de nous assurer qu'il est impossible d'approcher une solution optimale à BCCD dans les arbres sans avoir recours à une quelconque augmentation de ressources. On sait par contre, comme on l'a détaillé dans le Paragraphe 4.1, que quelle que soit l'augmentation de ressources utilisée, il est impossible d'obtenir un facteur d'approximation plus grand que $\frac{1}{2}$.

Etant donné un noeud x de l'arbre, on note pere(x) son père, et fils(x) l'ensemble de ses fils.

Cet algorithme est récursif. On lance son exécution en appelant la procédure ConstruiredansSousArbre sur la racine. La condition d'arrêt de cet algorithme est que l'appel de cette procédure ait lieu sur une feuille. Chaque feuille x de l'arbre, correspondant à un élément d'une instance $\mathcal{I} = (S, d, w, d_{\max})$ de BCCD, va renvoyer les informations contenant son poids à son père pere(x), si celui-ci se trouve à une distance inférieure à $d_{\max}$ de lui (sinon, il est trop loin et ne sera pas utilisé dans un groupe). Quand le noeud pere(x) aura reçu toutes les informations en provenance de ses fils, il construira de façon gloutonne des groupes de poids strictement inférieur à 2, en commençant par utiliser ses fils les plus loin de lui.

Cette construction gloutonne de groupes sans considérer les distances séparant chacun des fils entre eux, mais en considérant simplement la distance de chaque fils à leur père commun va engendrer la construction de groupes de diamètre au plus $2d_{\max}$. Quand plus aucun groupe n'est constructible, pere(x) renvoie à son propre père pere(pere(x)) les informations concernant ses fils qu'il n'a pas pu placer dans un groupe, si ceux-ci se trouvent à une distance inférieure à $d_{\max}$ de pere(pere(x)). pere(x) transmet ainsi à son père, pour chacun de ses fils, son poids et la distance à laquelle il se trouve. Une remontée récursive de l'arbre en construisant les groupes de cette façon suffit à nous assurer la facteur d'approximation annoncé.

Un exemple de la fusion exécutée ligne 12 de l'Algorithme 13 est présenté dans la Figure 4.19.

Un exemple d'exécution est présenté dans la Figure 4.20 présentant un arbre dont la racine et les feuilles exclusivement sont pondérées. Les noeuds virtuels sont de poids nul. Deux groupes sont construits par l'Algorithme 13, tandis qu'une solution optimale ne construit qu'un seul groupe (vert).

Theorème 4.35

L'Algorithme 13 est un $(\frac{1}{3}, 2)$*-algorithme d'approximation pour BCCD dans les arbres.*

Démonstration : Etant donnée une instance $\mathcal{I} = (S, d, w, d_{\max})$ de BCCD dans laquelle les éléments sont plongés dans un arbre T, de racine r, on choisit une solution optimale, notée Opt_{BCCD}. On note $\mathcal{B}$ l'ensemble des groupes construits par l'Algorithme 13 sur $\mathcal{I}$.

Tout d'abord, observons que chaque groupe créé a un diamètre inférieur à $2d_{\max}$, d'où l'augmentation de ressources utilisée. En effet deux éléments sont placés dans un même groupe s'ils sont tous deux à une distance inférieure à $d_{\max}$ du noeud qui les place dans un groupe, et, donc, ils sont à distance au plus $2d_{\max}$ l'un de l'autre.

Considérons un groupe B d'une solution de BCCD. On note $r(B)$ le plus petit ancêtre commun aux éléments de B, qu'on nomme sa racine. En particulier, tout élément de B est à distance au plus $d_{\max}$ de $a(B)$.

Notons que si deux groupes de deux solutions différentes utilisent le même élément, alors la racine de l'un des deux est l'ancêtre de la racine de l'autre.

On associe chaque groupe O de Opt_{BCCD} au groupe B de $\mathcal{B}$ l'intersectant, et dont la racine est la plus éloignée de la racine de l'arbre T. Si la distance maximale depuis la

Algorithme 13 $(\frac{1}{3}, 2)$-algorithme d'approximation pour BCCD dans un arbre

Procédure ConstruiredansSousArbre(x) :

si x est une feuille **alors**
  si $d(x, \text{pere}(x)) \leq d_{\max}$ **alors**
    retourner $(d(x, \text{pere}(x)), w(x), x)$.
  sinon
    retourner $\emptyset$.
  fin si
sinon
  pour chaque noeud $y \in \text{fils}(x)$ **faire**
    L_y =ConstruiredansSousArbre(y)
  fin pour
  fusionner les listes $L_y = \{(d_1, w_1, h_1), \ldots, (d_k, w_k, h_k)\}, \forall y \in \text{fils}(x)$ en $L_x = \{(d'_1, w'_1, h'_1), \ldots, (d'_{k'}, w'_{k'}, h'_k)\}$ où $\forall i < j, d'_i \leq d'_j$, et $\forall i, w'_i = w'_{i+1} + w(y)$ où y est le noeud à distance d'_i de x dans $\bigcup_{y \in \text{fils}(x)} L_y$.
  /* Note : tous les d'_i sont inférieurs à $d_{\max}$*/
  tant que il existe dans L_x un i tel que $w'_i \geq 1$ **faire**
    prendre le i le plus grand vérifiant cette propriété,
    construire un groupe composé des éléments h'_j de tous les triplets $d'_j, w'_j, h'_j)$ de L_x tels que $j \geq i$,
    supprimer de L_x les triplets (d'_j, w'_j, h'_j) pour $j \geq i$.
  fin tant que
  si x est la racine **alors**
    retourner "fin"
  sinon
    pour chaque triplet $(d'_i, w'_i, h'_i) \in L_x$ **faire**
      $d'_i \leftarrow d'_i + d(x, \text{pere}(x))$
      si $d'_i > d_{\max}$ **alors**
        supprimer de L_x les triplets (d'_j, w'_j, h'_j) pour $j \geq i$.
      fin si
    fin pour
    retourner L_x.
  fin si
fin si

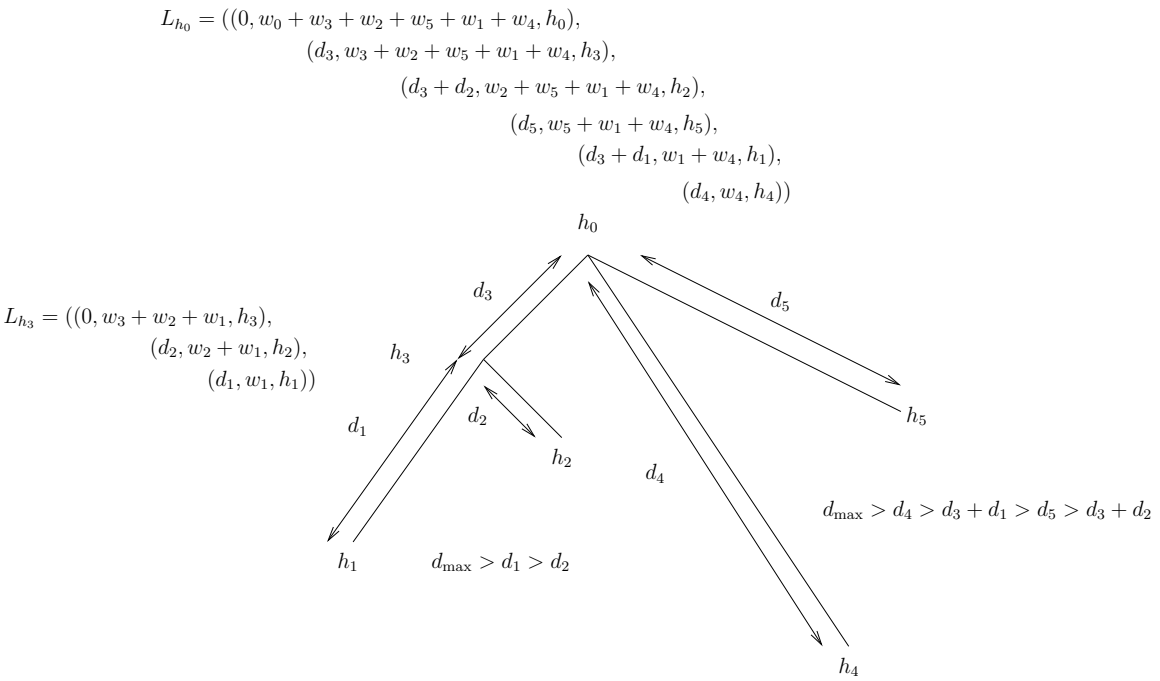


FIG. 4.19 – Exemple de fusion de listes d'éléments comme effectuée ligne 12 de l'Algorithme 13. $d_{\max}$ est, en l'occurence, plus grand que le diamètre de l'arbre. A chaque noeud interne une fusion a lieu, classant les éléments selon l'ordre croissant de leurs distances au noeud virtuel, et leur affectant un poids égal à la somme des poids des éléments se trouvant à une plus grande distance qu'eux du noeud virtuel sur lequel a lieu la fusion.

racine d'un des groupes de $\mathcal{B}$ intersectant O à la racine de T est la même pour plusieurs de ces groupes, on choisit celui qui a été construit en premier par l'Algorithme 13 pour l'associer à O. Au moins un groupe B de $\mathcal{B}$ intersecte chaque groupe O, puisque si aucun autre groupe n'a été construit auparavant, l'exécution de l'Algorithme 13 en $r(O)$ aurait abouti à la construction d'un groupe.

Dans un premier temps, remarquons que le noeud $r(O)$ est un ancêtre de $r(B)$ (ou $r(B)$ lui-même) si O est associé à B. En effet, supposons que ce ne soit pas le cas, alors $r(B)$ est un ancêtre strict de $r(O)$, puisqu'ils utilisent un même élément. L'exécution de l'Algorithme 13 implique qu'en $r(O)$ un groupe aurait été construit, le groupe O pouvant être construit avant la construction du groupe B (empêchant celle-ci). En effet, d'après la définition de l'association de O à B, aucun élément de O n'a encore été placé dans un groupe de $\mathcal{B}$ lors de l'examen de $r(O)$ ($r(B)$ est le "premier" élément à avoir utilisé des éléments de O pour construire un groupe).

Dans la suite nous comptons le poids total dont disposait OPT_{BCCD} pour construire des groupes. Pour cela, définissons w_{perdu} comme le poids des éléments de OPT_{BCCD} non utilisés dans des groupes de $\mathcal{B}$. Chaque élément dont le poids est ainsi décompté appartient à un groupe optimal. On peut donc associer chacun de ces éléments au groupe de $\mathcal{B}$ auquel est associé le groupe optimal dont il fait partie. On définit alors, étant donné un groupe $B \in \mathcal{B}$, le poids perdu associé à B, $w_{\text{perdu}}(B)$, comme le poids des éléments de groupes optimaux associés à B et non utilisés par des groupes de $\mathcal{B}$.

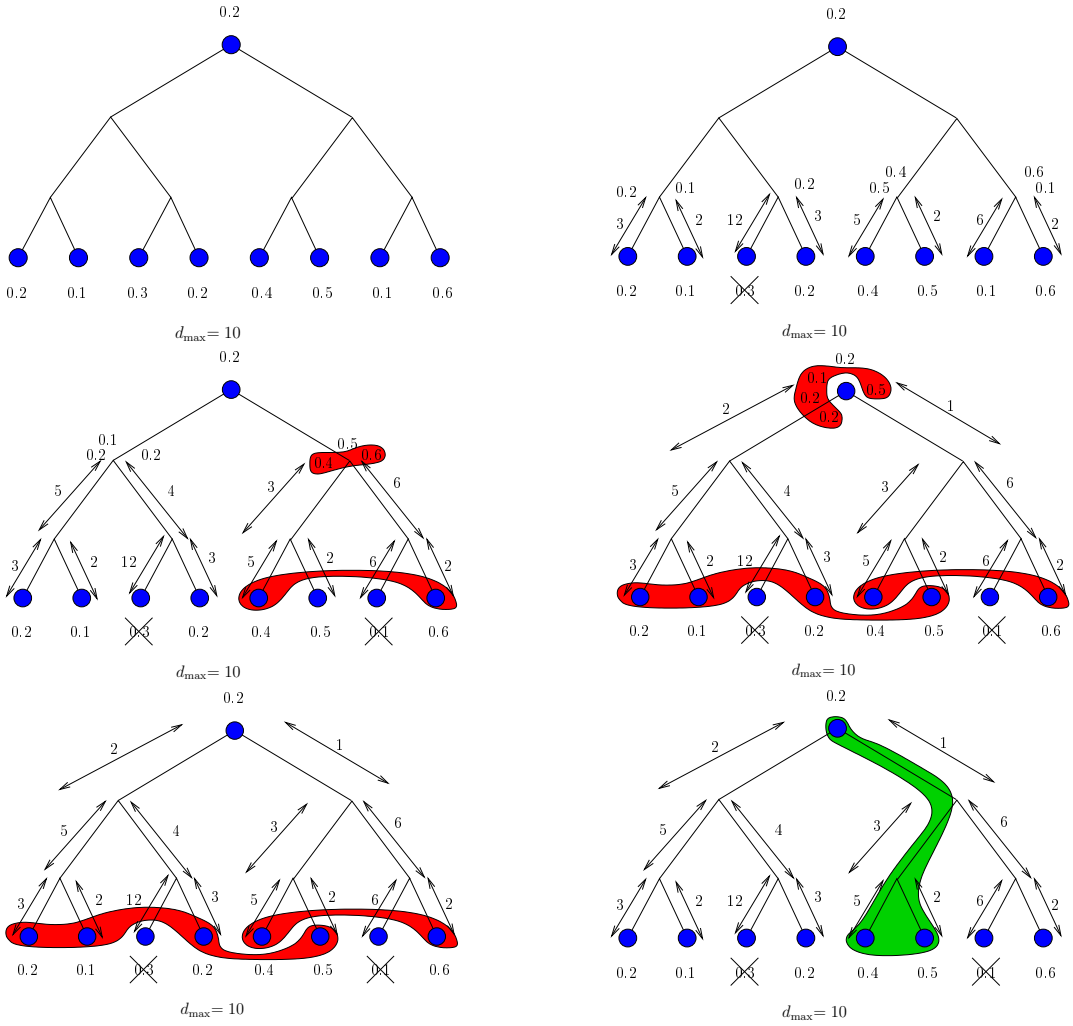


FIG. 4.20 – Exemple d'exécution de l'Algorithme 13 construisant 2 groupes dans un arbre dans lequel une solution optimale en construit 1 (cf. dernière figure). Dès le première étape, un élément est éliminé (deuxième figure). Aucun groupe ne peut être créé avant la troisème étape, au cours de laquelle un groupe est construit en un noeud virtuel, en utilisant en priorité les éléments les plus éloignés. Finalement un dernier groupe sera construit avec la racine.

Lemme 4.36

Pour tout groupe B de $\mathcal{B}$, $w_{perdu}(B) < 1$.

Démonstration : Soit d la distance de $r(B)$ à l'élément e_d de B, appartenant à un groupe O de OPT_{BCCD}, tel que $r(O)$ est le plus proche possible de la racine de T. Soit d' la distance de $r(B)$ à l'élément $e_{d'}$ d'un des groupes optimaux associés à B ne faisant pas partie de B, non utilisé dans un groupe de $\mathcal{B}$ et étant le plus éloigné de $r(B)$ (voir Figure 4.21). On a $d \geq d'$. En effet lors de la construction de groupe B en $r(B)$, B utilise les éléments en les prenant dans l'ordre décroissant de leur distance à lui-même. Il commence dont par utiliser les éléments les plus loin de lui. Il utilise e_d et pas $e_{d'}$, donc $d(r(B), e_d) \geq d(r(B), e_{d'})$.

Tous les groupes optimaux associés à B ont pour ancêtre un ancêtre de $r(B)$

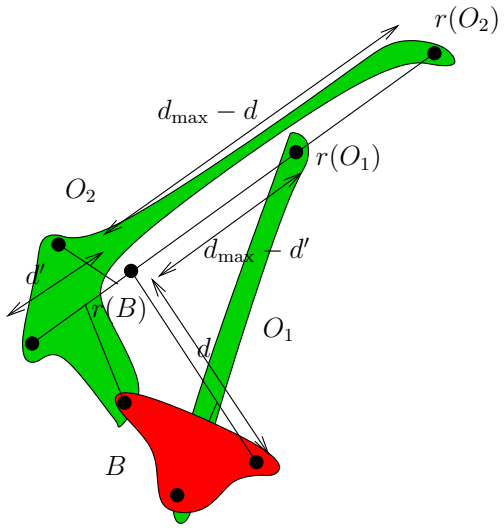


FIG. 4.21 – Exemple d'un groupe B de $\mathcal{B}$ et de deux groupes optimaux O_1 et O_2 dont les ancêtres respectifs $r(O_1)$ et $r(O_2)$ font partie du groupe O_1 et O_2 respectivement. $r(O_2)$ est à distance au plus $d_{\max}$ de tous les éléments comptés dans $w_{\text{perdu}}(B)$. Ces éléments se trouvent entre B et $r(O_2)$.

(ou lui-même). Il en existe au moins un qui soit l'ancêtre de tous ces ancêtres. S'il en existe plusieurs, prenons en un arbitrairement. Ce noeud, a, est à distance inférieure à $d_{\max} - d$ de $r(B)$, de par la définition de d. Ainsi il est à distance au plus $d_{\max} - d + d'$ de tous les éléments dont le poids est décompté dans $w_{\text{perdu}}(B)$, de par la définition de d'. Comme $d \geq d'$, il est à distance au plus $d_{\max}$ de tous ces éléments.

Ainsi, tous les éléments qui contribuent à $w_{\text{perdu}}(B)$ sont à distance au plus $d_{\max}$ de a. Par conséquent, $w_{\text{perdu}}(B)$ est inférieure à 1, sinon l'exécution de l'Algorithme 13 aurait abouti à la construction d'un groupe. ■

Le poids de chaque groupe de $\mathcal{B}$ est strictement inférieur à 2. On peut donc majorer la cardinalité d'une solution optimale par :

$$|\text{OPT}_{BCCD}| \leq w(\text{OPT}_{BCCD}) \leq 2|\mathcal{B}| + \sum_{B \in \mathcal{B}} w_{\text{perdu}}(B) \leq 3|\mathcal{B}|,$$

d'où le facteur d'approximation annoncé. ■

4.6 BPCD sans augmentation de ressources dans le plan euclidien

Dans ce paragraphe nous présentons une famille d'algorithmes centralisés assurant une $(\frac{5}{2} + \varepsilon, 1)$-approximation pour BPCD dans le plan euclidien, pour tout $\varepsilon > 0$. Ces algorithmes utilisent, de la même façon que le $(7/3, 2)$-algorithme d'approximation présenté dans le Paragraphe 3.4, les travaux effectués par Epstein et Levin sur BPAC dans [EL08]. En particulier

on réutilise un algorithme présenté dans leurs travaux, en le modifiant pour travailler sur une instance de BPCD. Cet algorithme utilise un calcul de couplage de poids maximum dans un graphe biparti :

Définition 4.37 (*Couplage de poids maximum*)

Etant donné un graphe $G = (V, E)$ et une fonction de pondération $w : E \to \mathbb{R}^+$ des arêtes, trouver un ensemble d'arêtes $C \subseteq E$ tel qu'aucune arête de C n'ait d'extrémité commune, et que le poids total des arêtes de C soit maximum.

Il existe un algorithme s'exécutant en temps polynomial et produisant une solution optimale au problème de couplage de poids maximum dans un graphe biparti. C'est un résultat connu de la littérature, qu'on peut trouver par exemple dans le livre de West [W+01].

On réutilise dans ce paragraphe le problème de la partition minimale en cliques, dont nous rappelons la définition (voir la Définition 3.9 dans le Paragraphe 3.2.2 pour plus de détails).

Définition 4.38 (*Partition Minimale en Cliques*)

Etant donné un graphe $G = (V, E)$, trouver une partition de V en sous-ensembles disjoints $V_1, \dots V_K$ de V de cardinalité minimale K, telle que $\forall i \leq K$, V_i soit une clique, i.e. un sous-graphe complet de G.

On note $\text{OPT}_{CliquePart}(G)$ (ou simplement $\text{OPT}_{CliquePart}$) une solution où la valeur de K est minimale pour un graphe G donné, et $|\text{OPT}_{CliquePart}(G)|$ cette valeur K minimale.

Chaque algorithme de la famille d'algorithmes que nous présentons consiste dans un premier temps à construire un graphe biparti à partir des éléments d'une instance de BPCD, et à y résoudre le problème de couplage de poids maximum.

Ce graphe biparti $G = (L \cup R, E)$ est composé de deux ensembles L et R définis comme suit :
- L est composé de l'ensemble des éléments de poids strictement supérieur à $1/2$,
- R est composé des autres éléments.

L'arête $(u, v) \in L \times R$ est dans E si et seulement si u et v forment un groupe valide, *i.e.* sont à moins de $d_{\max}$ l'un de l'autre, et la somme de leurs poids est inférieure à 1.

Etant donné ce graphe biparti G, un algorithme de la famille que nous définissons trouve un couplage maximum dans ce graphe en temps polynomial, et place chacune des paires d'éléments de ce couplage dans un groupe avant de supprimer cette paire de l'instance.

On construit ensuite, à partir des éléments restants, le graphe de compatibilité $\text{Comp}(\mathcal{I}, d_{\max})$ (dans lequel deux éléments sont connectés par une arête s'ils sont à moins de $d_{\max}$ l'un de l'autre, voir Paragraphe 2.3). On construit ensuite la plus petite partition en cliques possible de $\text{Comp}(\mathcal{I}, d_{\max})$. Enfin dans chaque clique de la partition obtenue, de diamètre au plus $d_{\max}$, on applique l'algorithme *First-Fit-Decreasing*, pour obtenir le facteur d'approximation annoncé dans le Théorème 4.40.

Lemme 4.39 (*Reformulation du Théorème 9 couplé au Lemme 3.22 du Paragraphe 3.4*)

Dans l'Algorithme 14, si la partition construite est de cardinalité inférieure à γOPT_{BPCD}, $\gamma \geq 1$, alors le nombre de groupes construits par l'Algorithme 14 est au plus égal à $(\frac{3}{2} + \gamma)\text{OPT}_{BPCD}$.

Algorithme 14 $(\frac{5}{2}+\varepsilon, 1)$-algorithme d'approximation pour BPCD dans le plan euclidien, pour tout $\varepsilon > 0$

1: définissons le graphe biparti suivant : l'un des ensembles indépendants de ce graphe est l'ensemble des éléments de poids strictement supérieur à 1/2, l'autre est l'ensemble des éléments restants. Une arête (a, b) existe entre deux éléments de poids $w(a) > 1/2, w(b) \leq 1/2$ si l'ensemble $\{a, b\}$ forme un groupe valide. Si une telle arête existe, elle est pondérée avec le poids étendu $e(b)$ pour l'entier j tel que $w(b) \in]\frac{1}{j+1}, \frac{1}{j}]$
2: trouver un couplage de poids maximum dans ce graphe biparti
3: placer chaque paire d'éléments couplés dans un groupe et l'en supprimer de l'instance
4: $\mathcal{C} = \text{Comp}(\mathcal{I}, d_{\max})$
5: construire une partition de $\mathcal{C}$ en cliques
6: appliquer l'algorithme *First-Fit-Decreasing* au sein de chacune des cliques.

Dans [EL08] les auteurs obtiennent ce résultat d'approximation sur des graphes de conflits parfaits, sur lesquels il est en particulier possible de construire une solution exacte au problème de coloration en temps polynomial. Dans ce cas, $\gamma = 1$ (dans l'énoncé équivalent pour BPAC) et ils obtiennent une $\frac{5}{2}$ approximation de BPAC.

Construire une solution au problème de coloration sur un graphe donné est équivalent à construire une partition en cliques sur le graphe complémentaire. Dans BPCD, nous travaillons sur les graphes complémentaires des graphes de conflits considérés dans [EL08]. Plus précisément, si dans une instance de BPCD on sait construire en temps polynomial une partition de cardinalité minimale en cliques, dans l'instance de BPAC correspondante, on sait construire une solution exacte au problème de coloration, et on peut donc appliquer le Théorème 9 d'Epstein et Levin.

C'est cette observation qui nous permet d'obtenir le théorème suivant :

Theorème 4.40

L'Algorithme 14 est un $(\frac{5}{2}+\varepsilon, 1)$-algorithme d'approximation pour BPCD restreint aux instances où les éléments sont dans le plan euclidien, pour tout $\varepsilon > 0$.

Démonstration : Quand l'espace métrique dans lequel sont plongés les éléments est le plan euclidien, le graphe de compatibilité $\text{Comp}(\mathcal{I}, d_{\max})$ associé à une instance $\mathcal{I} = (S, d, w, d_{\max})$ de BPCD est un graphe disque-unité [CCDC90, MBI+95] ("Unit-Disk Graph"). Ce type de graphe est bien étudié dans la littérature, et il existe, pour de nombreux problèmes dur à approcher dans le cas général, des algorithmes proposant des solutions exactes ou approchés en temps polynomial.

En particulier, il existe un PTAS (Schéma d'approximation en temps polynomial) pour le problème de partition en cliques de cardinalité minimale présenté par Dumitrescu dans [DP09] dans les graphes de disque-unité, *i.e.* une famille d'algorithmes $\mathcal{A}_\varepsilon$, pour tout $\varepsilon > 0$, qui assurent un facteur d'approximation de $1 + \varepsilon$ dans ce type de graphes.

Le Lemme 4.39 appliqué à l'utilisation de ce PTAS nous permet de conclure quant au résultat annoncé. ■

Ce résultat est notamment intéressant car il prouve que si les éléments d'une instance de BPCD sont placés dans le plan euclidien, il n'est pas nécessaire d'avoir recours à une augmentation de ressources pour obtenir une approximation à facteur constant de la solution optimale.

Enfin notons que l'Algorithme 14 peut être exécuté dans tout graphe appartenant à une classe de graphes sur laquelle une approximation à facteur constant du problème de partition en cliques de cardinalité minimale peut être construite en temps polynomial. Le Lemme 4.39 s'applique alors, indiquant le facteur d'approximation obtenu en fonction du facteur d'approximation lié à la construction de la partition en cliques.

4.7 BPCD dans un arbre

Dans ce paragraphe nous étudions les résultats qu'il est possible d'obtenir lorsque les éléments d'une instance de BPCD sont dans une métrique d'arbre. De la même façon que dans le Paragraphe 4.5, nous considérons qu'un plongement des éléments d'une instance $\mathcal{I} = (S, d, w, d_{\max})$ de BPCD dans un arbre T est donné. Tout élément de S est un noeud de l'arbre T, mais tout noeud de l'arbre T n'est pas nécessairement un élément de S. Précisons que toute feuille de T est un élément de S (si c'est un noeud virtuel, on peut le supprimer sans perturber le plongement). Ces noeuds "virtuels" se voient affecter un poids nul, pour simplifier la description de l'algorithme que nous présentons. La distance d est la distance dans cet arbre T.

Considérer un arbre constitué exclusivement d'éléments de S, ou un arbre contenant également des noeuds virtuels, n'engendre aucune augmentation du nombre de groupes nécessaires à la construction d'une solution à BPCD, comme le démontre le lemme suivant :

Lemme 4.41

Tout noeud virtuel v non-feuille peut être placé dans n'importe quel groupe B dont l'ancêtre commun a est un de ses ancêtres ou v lui-même, et tel que v soit ancêtre d'une des feuilles f de B.

Démonstration : Comme v est de poids nul, il ne change pas le poids d'un groupe par sa présence. Comme l'ancêtre d'un groupe est à distance inférieure à $d_{\max}$ de tous ses éléments, et que $d(f, a) = d(f, v) + d(v, a)$, alors il est à distance inférieure à $d_{\max}$ de tous les éléments du groupe. ■

Nous prouvons dans un premier temps que la graphe de compatibilité associé à une métrique d'arbre est un graphe triangulé.

Définition 4.42 (*Graphe triangulé*)

Un graphe triangulé est un graphe dans lequel il n'existe pas de cycle de taille strictement supérieure à 3 sans corde.

Lemme 4.43

Pour tout arbre construit sur un ensemble de sommets V, dont les arêtes sont pondérés avec des valeurs dans $\mathbb{R}^+$ utilisées pour définir les distances, pour toute contrainte de distance $d_{\max}$, le graphe de compatibilité Comp$(V, d_{\max})$ (connectant deux éléments de V à moins de $d_{\max}$ l'un de l'autre dans l'arbre) est un graphe triangulé.

Démonstration : Considérons un arbre $\mathcal{A}$ construit sur un ensemble V de sommets. Supposons que dans le graphe de compatibilité Comp$(V, d_{\max})$ il existe 4 éléments formant un cycle de taille 4 sans corde. Soient $a, b, c, d \in V^4$ tels que :

- $d(a,b) \leq d_{\max}$
- $d(b,c) \leq d_{\max}$
- $d(c,d) \leq d_{\max}$
- $d(d,a) \leq d_{\max}$
- mais $d(a,c) > d_{\max}$
- et $d(b,d) > d_{\max}$

Soit $r_{x,y}$ le plus petit ancêtre commun aux éléments x et y. Sans perte de généralité, considérons $r_{a,b} \leq r_{a,c} \leq r_{a,d}$ où $x \leq y$ si x est plus profond dans l'arbre. Comme on est dans un arbre, $r_{b,c} = r_{a,c}$ et $r_{b,d} = r_{c,d} = r_{a,d}$ (voir Figure 4.22).

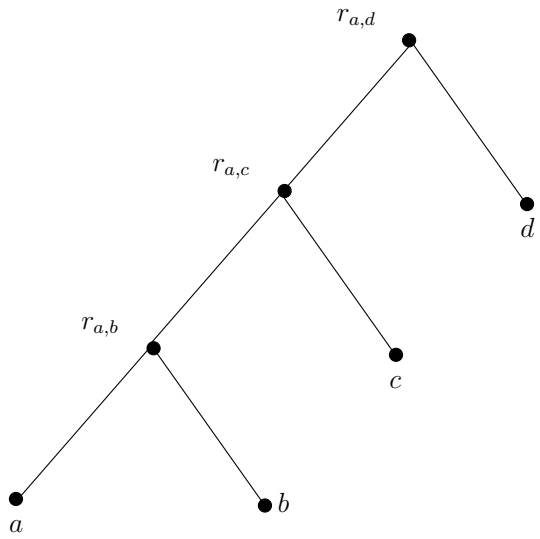


FIG. 4.22 – Un arbre avec les 4 points a, b, c et d placés comme dans notre preuve.

On a ainsi les inéquations suivantes :

- $d(a,b) = d(a,r_{a,b}) + d(r_{a,b},b) \leq d_{\max}$
- $d(b,c) = d(b,r_{a,c}) + d(r_{a,c},c) \leq d_{\max}$
- $d(c,d) = d(c,r_{a,d}) + d(r_{a,d},d) \leq d_{\max}$
- $d(d,a) = d(d,r_{a,d}) + d(r_{a,d},a) \leq d_{\max}$
- $d(a,c) = d(a,r_{a,c}) + d(r_{a,c},c) > d_{\max}$

On a alors :

$$\begin{aligned}
d(b,d) &= d(b,r_{a,d}) + d(r_{a,d},d) \\
&\leq d(b,r_{a,c}) + d(r_{a,c},ra,d) + d_{\max} - d(r_{a,d},a) \\
&\leq d_{\max} - d(r_{a,c},c) + d(r_{a,c},ra,d) + d_{\max} - d(r_{a,c},r_{a,d}) - d(r_{a,c},a) \\
&\leq d_{\max} + (d(a,r_{a,c}) - d_{\max}) + d_{\max} + (d(r_{a,c},c) - d_{\max}) \\
&\leq d(a,r_{a,c}) + d(r_{a,c},c) \\
&\leq d(a,c) \\
&\leq d_{\max}
\end{aligned}$$

Ce qui est en contradiction avec les hypothèses. Si on construit le graphe de compa-

tibilité d'un ensemble d'éléments dans un arbre, on obtient donc un graphe triangulé. ■

Dans la suite, nous utilisons des résultats qui s'appliquent aux graphes parfaits.

Définition 4.44 (*Graphe Parfait*)

Un graphe G est parfait si la cardinalité d'une solution optimale au problème de coloration de graphe sur tout sous-graphe induit de G (appelé son nombre chromatique) est égale à la cardinalité de la clique de taille maximale incluse dans ce sous-graphe.

Remarquons qu'un graphe triangulé est un graphe parfait [Ber60]. Comme nous l'avons décrit dans le Paragraphe 2.2.1, on peut appliquer sur une instance de BPCD un algorithme dédié à BPaCen considérant le complémentaire du graphe de compatibilité de l'instance de BPCD. Ainsi on peut appliquer un autre résultat du travail de Epstein et Levin présenté dans [EL08], s'appliquant aux graphes de conflits parfaits :

Lemme 4.45 (*Théorème 9 de [EL08]*)

Dans une instance de BPaC*, si le graphe de conflits est un graphe parfait, il existe un algorithme s'exécutant en temps polynomial assurant un facteur d'approximation de $\frac{5}{2}$.*

De ce théorème découle le suivant :

Theorème 4.46

Il existe un $(\frac{5}{2}, 1)$-algorithme d'approximation s'exécutant en temps polynomial pour BPCD quand la métrique utilisée est une métrique d'arbre.

Démonstration : Dans le cas de l'utilisation d'une métrique arborescente, le graphe de compatibilité est un graphe triangulé, donc parfait. Pour appliquer le Lemme 4.45, il faut que le graphe de conflits de l'instance de BPaC correspondante soit parfait. Ce graphe de conflits correspond au complémentaire du graphe de compatibilité. Or le complémentaire d'un graphe parfait est aussi parfait.

La construction du complémentaire du graphe de compatiblité $\overline{\text{Comp}(\mathcal{I}, d_{\max})}$ d'une instance $\mathcal{I}$ se fait en temps polynomial. On peut ensuite appliquer l'algorithme utilisé par Epstein et Levin pour prouver le Lemme 4.45. ■

4.8 Conclusion partielle

Dans ce chapitre nous avons étudié les possibilités d'extension de nos résultats pour BCCD et BPCD dans des espaces métriques particuliers. Nous avons obtenu les résultats suivants :

Pour BCCD : – un algorithme d'approximation à facteur constant ne nécessitant aucune augmentation de ressources dans le cas où les éléments sont plongés dans un espace vérifiant certaines propriétés (un espace métrique (S, d) tel que $\gamma(S)$ est borné, voir Paragraphe 4.2.1). Cet algorithme s'exécute en temps polynomial dans les espaces métriques dans lesquels la détection et la construction d'un groupe valide de diamètre $d_{\max}$ peut se faire en temps polynomial (en particulier, les espaces métriques dans lesquels le graphe

de compatibilité fait partie d'une classe de graphes dans laquelle le problème de trouver une clique de poids maximum),
- en utilisant cet algorithme dans le plan muni de la norme L_∞, on obtient un $(\frac{1}{11}, 1)$-algorithme d'approximation, ne nécessitant donc aucune augmentation de ressources,
- un $(\frac{2}{5}, 3)$-algorithme d'approximation dans le plan muni d'une norme quelconque,
- un $(\frac{1}{3}, 4)$-algorithme d'approximation distribué dans $\mathbb{R}^D$ muni de la norme L_∞. Ce résultat a été publié dans [BBDL08]. Cet algorithme utilise une structure de données appelée "skip-graph" pour organiser les éléments d'une instance. Nous avons prouvé dans [DL10] que la hauteur de cette structure de données, *i.e.* la longueur maximale d'un chemin de recherche d'un élément à un autre dans cette structure, est d'ordre $\log n$,
- un $(\frac{1}{3}, 2)$-algorithme d'approximation lorsque les éléments d'une instance sont plongés dans un arbre (contenant éventuellement des noeuds virtuels, ne correspondant pas à des éléments de l'instance).

Pour BPCD :
- une famille d'algorithmes assurant une $(\frac{5}{2} + \varepsilon, 1)$-approximation dans le plan euclidien, ne nécessitant donc aucune augmentation de ressources,
- un $(\frac{5}{2}, 1)$-algorithme d'approximation lorsque les éléments d'une instance sont plongés dans un arbre (du même type qu'auparavant, *i.e.* pouvant contenir des noeuds virtuels).

Il est par ailleurs intéressant de constater que les résultats d'inapproximabilité que nous avions obtenu dans le cas général n'étaient plus tous vérifiés dans les espaces métriques que nous étudions. En particulier, le résultat concernant l'inapproximabilité de ces problèmes à facteur constant, pour une augmentation de ressources de $(2 - \varepsilon)$ pour $\varepsilon > 0$, n'est plus vrai, ni pour BCCD ni pour BPCD. Cependant, quelque soit l'espace métrique considéré, BCCD n'est pas (α, β)-approximable pour $\alpha > 1/2$ et pour n'importe quelle valeur de β, et BPCD n'est pas (α, β)-approximable pour $\alpha < 3/2$ et pour n'importe quelle valeur de β.

Nous avons par ailleurs évoqué une réduction depuis CkDM à BCCD qui nous laisse penser que, même s'il est possible de détecter en temps polynomial l'existence d'un groupe valide dans une instance de BCCD (comme il est possible de le faire dans les espaces métriques auxquels nous nous sommes intéressés), il peut être compliqué de savoir lesquels construire pour maximiser le nombre total de groupes créés. Il serait intéressant, dans de futurs travaux, d'étudier plus précisément les résultats d'inapproximabilité qu'il serait possible d'obtenir en restreignant l'étude de ces problèmes aux espaces métriques que nous avons étudiés.

Dans le chapitre suivant nous présentons une série de simulations s'appuyant sur des outils de modélisations d'Internet connus. Nous étudions deux de ces outils, Vivaldi [DCKM04, CDK+04], qui plonge les noeuds d'Internet dans une variante du plan euclidien, et Sequoia [RMK+09, ABK+07], qui plonge les noeuds dans un arbre contenant des noeuds virtuels, comme ceux que nous avons étudiés dans ce chapitre. L'idée est de comparer ces deux modélisations d'Internet en examinant l'efficacité relative des algorithmes que nous avons conçus, en les adaptant à chacune des modélisations proposée par ces outils.

Chapitre 5

Vers une utilisation sur Internet

Dans ce chapitre nous étudions divers outils permettant de modéliser la structure du réseau d'Internet, qui fait partie des domaines d'applications de l'étude de BCCD et de BPCD. En particulier, nous nous intéressons à Vivaldi et à Sequoia, brièvement décrit respectivement dans les Paragraphes 5.1.1 et 5.1.2. Nous nous intéressons ensuite à l'étude de BCCD et de BPCD dans les espaces métriques générés par ces outils via des simulations utilisant des données réelles, présentées dans le Paragraphe 5.2.

5.1 Etat de l'art des outils de modélisation du réseau d'Internet

Dans ce paragraphe nous présentons brièvement plusieurs outils connus qui ont pour objectif de proposer des modélisations précises de l'espace des latences séparant les noeuds d'Internet. Plusieurs approches, parfois complètement différentes, existent.

Certaines approches commencent par identifier un nombre fixe, fini et relativement restreint de noeuds stables dans le réseaux, appelés "traceurs". Chaque noeud i du réseau choisit alors son traceur $t(i)$ le plus proche. On maintient une matrice de distances entre les traceurs, plus simple à maintenir qu'entre tous les éléments, car de taille beaucoup plus petite, et l'estimation de la distance entre deux éléments i et j se fait en fonction de leur traceur respectif $t(i)$ et $t(j)$: $d(i,j) = d(i,t(i)) + d(t(i);t(j)) + d(t(j),j)$. C'est le procédé qui est utilisé dans IDMaps [FJJ+01]. Ce principe revient à regrouper les noeuds en clusters dont on examine les distances relatives. Ce même principe est utilisé dans Internet Iso-bar [COK02].

De nombreux autres systèmes utilisent le principe des "landmarks", sortes de points de repères qu'on estime être des points fixes dans le réseau, qu'on peut rapprocher des "traceurs" d'IDMaps. Chaque noeud va calculer sa latence vers un ensemble fixe et prédéfini de tels points. Par triangulation il peut en déduire ses coordonnées dans un certain espace préalablement choisi, de telle sorte que la distance entre deux points dans cet espace, calculable à partir des coordonnées de ces deux points, soit une bonne approximation de la latence les séparant.

Par exemple GNP [NZ02, EZ04] utilise entre 5 et 20 points de repères par noeud, et plonge les noeuds du réseau dans un espace euclidien possédant une dizaine de dimensions environ.

Lighthouse [PCW+], une variante de ce type de système, propose d'utiliser le même principe avec un nombre arbitraire de points de repères, pour éviter des problèmes de contention aux

niveaux des noeuds agissant comme points de repères, ainsi que des problèmes de tolérance aux pannes de ceux-ci. De nombreuses variantes de ce type de système ont été étudiées [LHC03, TC03]. Mais le choix de ces points de repères reste un problème, quand il s'agit de ne pas en utiliser trop, et la dimension de l'espace visé dépend de ces choix qui peuvent s'avérer difficiles.

Netvigator [XSLB04] utilise également des points de repères, mais chaque noeud ne se contente plus simplement de garder la distance à ces repères, il en garde les "traceroutes", les chemins décrivant les routeurs utilisés pour rejoindre chacun de ces repères. Lors de l'estimation de la distance entre deux noeuds, on regarde alors les traceroutes de ces noeuds et on cherche le plus proche routeur commun.

Vivaldi [DCKM04, CDK+04] n'utilise pas de tels points de repères, mais modélise chaque lien entre deux noeuds par un ressort. Chaque fois qu'un noeud va effectuer une mesure de latence avec n'importe quel autre noeud du réseau, il va mettre à jour la tension du ressort les séparant, et ainsi mettre à jour ses coordonnées en utilisant des équations connues de mécanique newtonienne. Chaque noeud maintient ainsi ses coordonnées en calculant régulièrement la latence le séparant d'un noeud qui peut être choisi arbitrairement dans le réseau. Généralement, il affecte à chaque noeud des coordonnées dans un plan euclidien, en plus d'une hauteur à sommer lors de chaque estimation de distance à la distance euclidienne obtenue. En effet, en pratique, c'est ce type d'espace qui semble offrir le meilleur compromis entre la dimension de l'espace dans lequel sont plongés les éléments, et la qualité de la prédiction obtenue. Une variante de Vivaldi [LS06] plonge les noeuds dans un plan hyberbolique, en munissant également chaque noeud d'une hauteur. La distance entre deux noeuds devient alors la distance dans le plan hyperbolique entre ces deux noeuds, à laquelle on ajoute la somme de leurs hauteurs respectives.

Un autre type de travail intéressant est celui effectué avec King [GSG02]. Il a pour but l'estimation de la distance entre deux noeuds sans utiliser de plongement ou d'infrastructure supplémentaire, simplement en considérant que la distance entre les DNS utilisés par chacun des noeuds de la paire considérée est une bonne estimation de la distance entre ces noeuds. Ainsi aucun plongement dans un espace métrique n'est effectué dans ce cas, et on a recours à une nouvelle mesure à chaque fois qu'on souhaite estimer une distance. Mais ce procédé a l'avantage qu'une estimation précise de la distance entre deux noeuds distants est possible sans qu'il soit nécessaire qu'ils coopèrent (et donc sans passer par une mesure de latence effectué par l'un des noeuds de la paire vers l'autre).

Enfin, Sequoia [RMK+09, ABK+07] essaye de plonger l'espace des noeuds d'Internet dans un ou plusieurs arbres dans lesquels les noeuds du réseau sont les feuilles, et les noeuds internes de l'arbre sont des noeuds virtuels. L'objectif est que la distance entre deux points dans un tel arbre soit une bonne approximation de la latence les séparant.

5.1.1 Vivaldi

Dans Vivaldi [CDK+04, DCKM04], mis au point par Dabek *et al.*, les hôtes se voient attribuer une hauteur en plus de coordonnées dans le plan euclidien.

Cette idée d'associer deux types de coordonnées à chaque hôte, une hauteur et des coordonnées dans un plan, est inspirée du fait que le temps requis par les paquets pour traverser le lien d'accès entre le "coeur" d'Internet et l'hôte lui-même peut souvent être un point de ralentissement significatif, tandis qu'au sein de ce "coeur" d'Internet, il est attendu que les communications

soient rapides et bien approchées par la distance euclidienne sur Terre entre ces deux hôtes.

Dans l'espace généré par Vivaldi, un paquet allant d'un noeud à un autre doit "descendre" de la hauteur du noeud émetteur, puis traverser le plan euclidien entre les coordonnées dans le plan des deux noeuds en communication, puis finalement "monter" la hauteur du noeud cible. Ainsi même si deux noeuds ont la même hauteur, la distance totale les séparant est la distance euclidienne séparant leur projection dans le plan, additionnée à la somme de leurs hauteurs (il est impossible de voyager dans la troisième dimension).

La distance dans Vivaldi entre deux noeuds de coordonnées $(a_x, a_y, a_h) \in \mathbb{R}^2 \times \mathbb{R}^+$ et $(b_x, b_y, b_h) \in \mathbb{R}^2 \times \mathbb{R}^+$ est : $d(a,b) = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2} + a_h + b_h$.

Ce système permet une bonne estimation des latences entre les noeuds, puisque l'erreur relative médiane de l'estimation du temps d'aller retour entre deux noeuds ("Round Trip Time", RTT) est de 11%, comme l'ont présenté Dabek *et al.* dans [DCKM04].

5.1.2 Sequoia

L'idée de plonger le réseau des noeuds d'Internet dans un arbre provient de l'observation faite par Abraham *et al.* dans [ABK+07] que la structure d'Internet est relativement proche d'une métrique arborescente. Pour vérifier ceci, ils utilisent la notion de ε-4PC qui, étant donnés 4 points d'un espace métrique, permet de quantifier combien l'espace métrique restreint à ces 4 points est proche d'une métrique arborescente ($\varepsilon = 0$ correspondant aux métriques arborescentes, $\varepsilon = 1$ étant vérifié par n'importe quel espace métrique). Une majorité des quadruplets de noeuds pris dans Internet vérifie cette propriété pour une valeur de ε suffisamment petite pour qu'il soit intéressant d'essayer de plonger une métrique vérifiant la ε-4PC pour un ε assez faible pour tous les quadruplets dans un arbre dont les feuilles seraient les noeuds du réseau, et dans lequel les noeuds internes seraient virtuels. Dans [ABK+07] les auteurs prouvent que la distortion engendrée par un tel plongement est $(1+\varepsilon)^{(\Theta \log n)}$ dans le pire des cas, où n est le nombre de noeuds dans l'espace métrique d'origine.

Ramasubramanian *et al.* ont exploité ces travaux en implémentant ce plongement dans un outil appelé Sequoia [RMK+09]. Les feuilles de l'arbre construit par cet outil sont les noeuds du réseau (tout comme la racine), les autres sommets de l'arbre étant des noeuds virtuels. Une suite de calculs simples permet la construction de cet arbre. Pour chaque feuille, la distance à la racine est exacte. Les auteurs ont observé qu'il était nécessaire, pour obtenir de meilleurs résultats de prédiction de latence en pratique, d'utiliser plusieurs arbres, enracinés en des noeuds différents, puis, pour estimer la latence entre deux éléments, de prendre la valeur médiane des distances calculées dans chacun des arbres (l'erreur relative médiane passe ainsi d'environ 20% à 10%). Dans la suite nous considérons qu'un seul arbre est utilisé.

Nous présentons dans ce chapitre des simulations mettant en oeuvre les algorithmes que nous avons présentés dans les chapitres précédents, en utilisant des données réelles. Nous décrivons le processus expérimental dans le Paragraphe 5.2.1, et présentons les résultats comparatifs entre l'utilisation de Vivaldi, celle de Sequoia, et en s'appuyant sur des mesures exhaustives de RTTs séparant un ensemble de noeuds, dans les Paragraphes 5.2.2 et 5.2.3, concernant respectivement BCCD et BPCD.

5.2 Expérimentations

Dans ce paragraphe nous présentons les expérimentations que nous avons menées en vue de comparer les outils de modélisation du réseau des latences d'Internet que sont Vivaldi et Sequoia, présentés précédemment. Pour cela, nous avons appliqué les algorithmes développés durant nos travaux pour BCCD et BPCD à des instances de ces modélisations s'appuyant sur des jeux de données réels.

5.2.1 Protocole expérimental

Nous avons effectué des simulations en nous appuyant sur des jeux de données réelles. Pour chaque simulation, nous avons utilisé la même matrice de latence à partir de laquelle nous avons effectué, selon le cas, un plongement dans un arbre ou un plongement en utilisant Vivaldi. Cette matrice de latence est issue du projet Meridian [5] et contient les mesures de latences relatives à 2500 noeuds pris arbitrairement dans Internet. Dans cette matrice il est admis que la latence mesurée entre deux noeuds est la même quel que soit le noeud mesurant cette latence (*i.e.* la matrice est symétrique).

Cependant, de nombreuses violations de la propriété d'inégalité triangulaire existent dans ce jeu de données. En effet, l'espace des noeuds d'Internet muni de la latence pour définir les distances n'est pas un espace métrique (le lecteur intéressé pourra par exemple consulter [LFV08, LBSB09]). Pour nos expérimentations, nous avons travaillé sur une matrice ne contenant pas de telles violations (notamment car pour la construction d'un arbre Sequoia cela est nécessaire). Pour cela, nous créons une nouvelle matrice de latences dans laquelle la latence entre deux noeuds est la latence du plus court chemin entre ces deux noeuds dans le jeu de données d'origine. Pour obtenir le moins de distortions possible, nous avons modifié les valeurs nulles (qui correspondent à une absence de mesure) entre deux points distincts dans la latence d'origine, en les fixant à 500000 (μs). Nous avons choisi cette valeur car, après calcul des plus courts chemins, plus aucune latence entre deux noeuds n'avait pour valeur 500000, *i.e.* pour toute paire de noeuds pour laquelle on avait fixé la latence à 500000, il existait un plus court chemin de longueur inférieure.

Ce faisant nous avons observé une très grande distortion des latences entre la matrice d'origine et la matrice modifiée des plus courts chemins. Par exemple, la latence moyenne entre deux éléments passe de $75ms$ à $10ms$. Nous avons également voulu évaluer la variation des diamètres moyen de groupes de noeuds tirés aléatoirement dans ces matrices. Dans la première, pour 7 noeuds tirés aléatoirement, le diamètre, soit la latence maximale entre deux noeuds parmi les 7 noeuds tirés, est de $187ms$ en moyenne, tandis que dans la matrice modifiée, ce même diamètre est de $26ms$ en moyenne (soit un facteur 7 entre les deux). Nous avons également calculé que pour 57% des mesures, les distances sont divisées par au moins 6. Toutes ces observations nous ont mené à la conclusion que si on veut travailler sur une matrice sans violation d'inégalité triangulaire, il faut accepter le fait que celle-ci ne reflète pas la réalité des latences dans Internet. Notons qu'on ne sait finalement pas si cette nouvelle matrice est meilleure ou pire que celle d'origine, étant données le nombre de valeurs absentes et absurdes. Ne disposant pas de meilleurs jeux de données nous avons utilisé celui-ci, il est donc nécessaire d'analyser la suite de ces expérimentations en gardant bien cette remarque à l'esprit. Nous ne travaillerons désormais

[5] http ://www.cs.cornell.edu/People/egs/meridian/data.php

qu'avec la matrice de latence modifiée, vérifiant l'inégalité triangulaire.

Pour générer les poids des éléments, nous avons utilisé les données du projet XtremLab [6], qui a pour objectif de recenser les caractéristiques de noeuds participants à la plateforme BOINC (exactement le type de plateformes visées par nos applications). De ces données nous avons extrait les informations concernant les capacités de calcul des noeuds étudiés. Plus précisément on extrait de ces données le nombre d'opérations en virgule flottante par seconde ("FLoating point Operations Per Second") que peut effectuer chaque noeud. Nous nettoyons les valeurs pour ne conserver que celles qui sont cohérentes, allant de 500 000 (500 KFlops) à 10 milliards (10 GFlops). Nous avons tiré aléatoirement 2500 valeurs prises parmi celles-ci (épurées des valeurs absurdes, et normalisées pour obtenir un poids total de 100) pour faire office de poids des éléments des instances de BCCD et de BPCD étudiées. La distribution de ces capacités est illustrée dans la Figure 5.1.

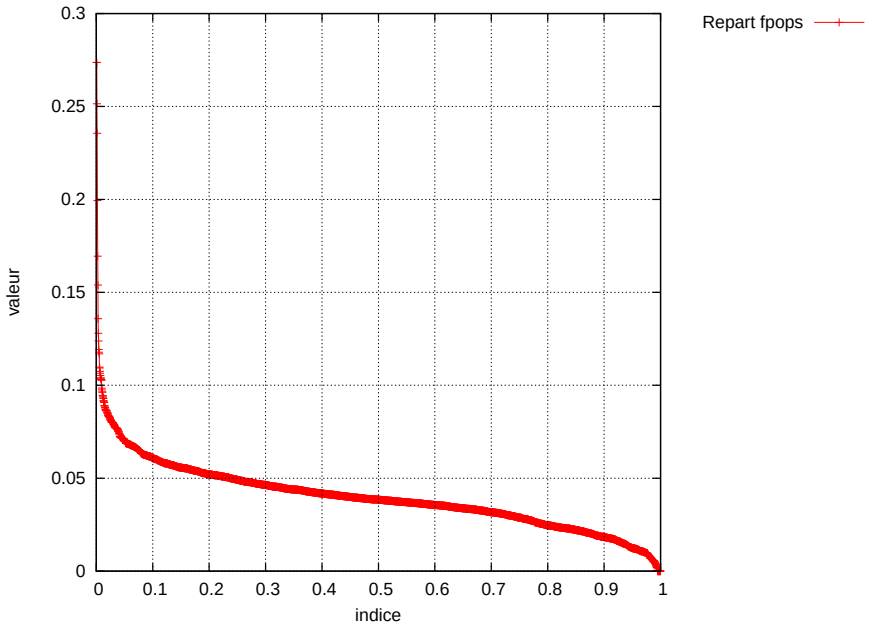


FIG. 5.1 – Distribution des poids utilisés, une fois normalisés.

Pour chaque expérimentation, nous suivons le protocole suivant :

- Dans le cas d'une simulation utilisant Vivaldi ou Sequoia, nous effectuons un plongement de la matrice de latence en utilisant Vivaldi ou Sequoia, selon le cas.
- Les éléments se voient ensuite attribuer un poids parmi les 2500 valeurs tirées des mesures de capacités de calcul du projet XtremLab. Pour chaque expérimentation, un même élément de la matrice de latence se voit attribuer le même poids.
- Dans chacun de ces plongements, les distances minimales et maximales séparant deux noeuds sont identifiées.
- Un des algorithmes pour résoudre BCCD ou BPCD est appliqué successivement au même plongement du même ensemble d'éléments pondérés, en faisant varier la valeur $d_{\max}$ entre la distance minimale et la distance maximale séparant deux points.

Nos expérimentations ont plusieurs objectifs. Il s'agit d'abord d'évaluer l'efficacité des dif-

[6] http ://xw01.lri.fr :4320/

férents algorithmes que nous avons proposés et dédiés à différents espaces métriques, en les comparant les uns aux autres. Il s'agit également de comparer les différents outils de modélisation d'Internet que nous avons selectionnés, en les utilisant pour exécuter des algorithmes complexes. Ainsi nous obtenons un comparatif de ces modèles selon différents critères, qui ne se limitent pas seulement à la mesure de l'erreur des prédictions que ces outils génèrent. Enfin il s'agit de trouver lequel de ces outils offre le meilleur compromis entre la distortion engendrée par le plongement et l'efficacité de l'algorithme existant dans l'espace métrique induit par le plongement.

Nos algorithmes dédiés à des espaces métriques spécifiques offrent en effet de meilleures performances que les algorithmes correspondant dans des espaces métriques quelconques. Il reste cependant à vérifier que le plongement de l'espace des latences dans ces espaces métriques n'engendre pas de distortions trop importantes, qui rendraient obsolètes ces améliorations de complexité algorithmique. Pour cela, nous comparons les performances de chacun des algorithmes proposés appliqués à chacun des plongements correspondant, générés par les outils adéquats.

Nous avons donc effectué des expérimentations dans l'espace généré par Vivaldi, affectant deux coordonnées dans le plan à chaque élément ainsi qu'une hauteur (cf Paragraphe 5.1.1), ainsi que dans les arbres générés par Sequoia.

Nous avons ensuite utilisé Vivaldi pour plonger l'ensemble des noeuds dont nous disposons dans le plan euclidien. Ce plongement modélise de façon moins fidèle les distances entre les noeuds que l'autre plongement généré par Vivaldi que nous utilisons. Nous nous sommes également comparés à un algorithme travaillant directement avec la matrice de latences, sans effectuer aucun plongement, et donc sans aucune perte d'informations concernant les latences séparant les noeuds (mais ce type d'algorithme est irréaliste dans le cadre d'applications réparties à grande échelle, puisqu'il serait trop coûteux de disposer d'une matrice de latence complète).

Pour chaque groupe créé, deux diamètres (mesurant la distance maximale entre deux éléments de ce groupe), mesurés différemment, sont conservés : le premier est mesuré *dans la matrice de latence d'origine* (la matrice modifiée pour vérifier l'inégalité triangulaire), et le deuxième est mesurée *dans le plongement utilisé*.

Pour chaque expérimentation, et chaque valeur de $d_{\max}$, plusieurs valeurs sont récupérées : le nombre de groupes créés, le diamètre maximal parmi les groupes créés, mesuré dans le plongement, deux valeurs du diamètre moyen des groupes créés, l'un dans la matrice de latence d'origine, l'autre dans le plongement, le nombre moyen d'éléments par groupe, et le nombre de groupes valides, comptant le nombre de groupes dont le diamètre dans la matrice d'origine satisfait la contrainte de distance augmentée ($d_{\max}$ multipliée par l'augmentation de ressources de l'algorithme utilisé).

Les expérimentations que nous avons menées ont mis en applications différents algorithmes, appliqués à différents espaces métriques générés par différents outils, qui sont les suivants :

Pour BCCD :
- une adaptation à un espace métrique généré par Vivaldi du $(\frac{1}{11}, 1)$-algorithme d'approximation dans le plan, présenté dans le Paragraphe 4.2.3,
- une adaptation à un espace métrique généré par Vivaldi du $(\frac{2}{5}, 3)$-algorithme d'approximation dans le plan, présenté dans le Paragraphe 4.2.4,
- le $(\frac{1}{11}, 1)$-algorithme d'approximation dans le plan, présenté dans le Paragraphe 4.2.3, sur un plongement dans le plan euclidien des noeuds, généré par Vivaldi,
- le $(\frac{2}{5}, 3)$-algorithme d'approximation dans le plan, présenté dans le Paragraphe 4.2.4, sur un plongement dans le plan euclidien des noeuds, généré par Vivaldi,

- le $(\frac{1}{3}, 2)$-algorithme d'approximation du Paragraphe 4.5 en utilisant Sequoia,
- le $(\frac{2}{5}, 4)$-algorithme d'approximation du Paragraphe 3.3 s'appuyant directement sur la matrice de latence d'origine pour évaluer les distances.

Pour BPCD : – le $(\frac{7}{3}, 2)$-algorithme d'approximation du Paragraphe 3.4 en utilisant Vivaldi,
- le $(\frac{7}{3}, 2)$-algorithme d'approximation du Paragraphe 3.4 en utilisant Sequoia,
- le $(\frac{7}{3}, 2)$-algorithme d'approximation du Paragraphe 3.4 en utilisant un plongement des noeuds dans le plan euclidien,
- le $(\frac{7}{3}, 2)$-algorithme d'approximation du Paragraphe 3.4 en s'appuyant directement sur la matrice de latence d'origine pour évaluer les distances.

Concernant BPCD, on remarque que le même algorithme du Paragraphe 3.4 est utilisé avec toutes les modélisations qui nous ont intéressées. En effet, il n'existe à notre connaissance pas de moyen d'adapter la famille d'algorithmes présentée dans le Paragraphe 4.6 à un espace métrique tel que ceux générés par Vivaldi, en utilisant une augmentation de ressources inférieure à 2 pour résoudre BPCD, et en obtenant un facteur d'approximation meilleur que $\frac{7}{3}$. En outre, cette famille d'algorithmes ne semblait pas réaliste à implémenter de par sa complexité (cette famille d'algorithmes utilise un PTAS pour la résolution du problème de partition en cliques dans un graphe de disque-unité, et ce type d'algorithmes n'est généralement pas implémentable raisonnablement en pratique).

Par ailleurs, le $(\frac{5}{2}, 1)$-algorithme d'approximation présenté dans le Paragraphe 4.7, pour résoudre BPCD dans les arbres (qui aurait donc pu être appliqué à l'utilisation de Sequoia), requiert l'utilisation d'un algorithme de coloration de graphes parfaits, difficilement implémentable également.

Nous avons donc choisi d'utiliser un seul et même algorithme exécuté sur différentes modélisations de l'espace des latences, avec dans l'idée qu'ainsi seront comparées directement les caractéristiques et les réactions des différents plongements face à la résolution de BPCD.

Enfin, nous avons effectué toutes ces simulations pour différentes valeurs de la contrainte de poids W, présente dans BCCD comme dans BPCD. Nous avons expérimenté les cas où $W \in \{0.2, 0.5, 1\}$.

5.2.2 Comparaison Vivaldi/Sequoia pour BCCD

Dans cette partie nous présentons les algorithmes que nous expérimentons pour l'étude pratique de BCCD, ainsi que les résultats obtenus.

Dans un premier temps, nous expérimentons le $(\frac{1}{11}, 1)$-algorithme d'approximation pour BCCD dans le plan, présenté dans le Paragraphe 4.2.3. Pour cela, les noeuds sont placés dans un espace tel que ceux générés par Vivaldi, qu'on nomme "*plan augmenté*" et qu'on a décrit dans le Paragraphe 5.1.1. Rappelons que dans un tel espace chaque noeud se voit attribuer des coordonnées dans le plan euclidien ainsi qu'une hauteur à sommer à la distance euclidienne lors de l'estimation des distances. Nous identifions les courbes correspondant à l'exécution de cet algorithme dans un plan augmenté par "Vivaldi Phase 1" (car il correspond à la première phase du $(\frac{2}{5}, 3)$-algorithme d'approximation dans le plan).

Nous utilisons également ce même algorithme dans un plan euclidien classique. Une autre version de Vivaldi permet en effet de placer les noeuds d'un réseau dans un plan euclidien

classique, sans hauteur additionnelle à chaque noeud. Nous identifions de façon similaire les courbes correspondant à l'exécution de cet algorithme par "Vivaldi 2D Phase 1".

Dans de tels espaces, le facteur d'approximation de $\frac{1}{11}$ assuré par cet algorithme n'est plus vérifié. En effet, ce facteur d'approximation de $\frac{1}{11}$ a été démontré dans le cadre d'un plongement dans le plan muni de la norme L_∞. Comme nous l'avons évoqué dans le chapitre précédent, l'utilisation de L_2 (et, donc, d'un plan euclidien) en lieu et place de la norme L_∞ modifie d'un facteur constant le facteur d'approximation, mais l'algorithme reste valide, sans avoir recours à une augmentation de ressources. Par contre, quand les éléments se voient attribuer des coordonnées euclidiennes ainsi qu'une hauteur, une augmentation de ressources est nécessaire de par la forme des boules dans un tel espace.

Nous expérimentons ensuite l'utilisation du $(\frac{2}{5}, 3)$-algorithme d'approximation du Paragraphe 4.2.4 dans ces deux espaces. Contrairement au cas du plan euclidien, dans un plan muni des hauteurs en chaque noeud l'augmentation de ressources requise est supérieure à 3. Le facteur d'approximation de $\frac{2}{5}$, lui, reste valide, puisqu'il est valide quelle que soit la norme utilisée. Nous identifions les courbes correspondant à l'exécution de cet algorithme dans ces deux espaces par, respectivement, "Vivaldi Phase 2" et "Vivaldi 2D Phase 2".

Les autres algorithmes utilisés sont des applications directes des algorithmes que nous avons présentés dans le Chapitre 4 :

- le $(\frac{1}{3}, 2)$-algorithme d'approximation du Paragraphe 4.5 en utilisant Sequoia, dont nous identifions les courbes par "Sequoia",
- le $(\frac{2}{5}, 4)$-algorithme d'approximation du Paragraphe 3.3 s'appuyant directement sur la matrice de latence d'origine pour évaluer les distances. Nous identifions les courbes appliquant cet algorithme par "AlgoGeneraux".

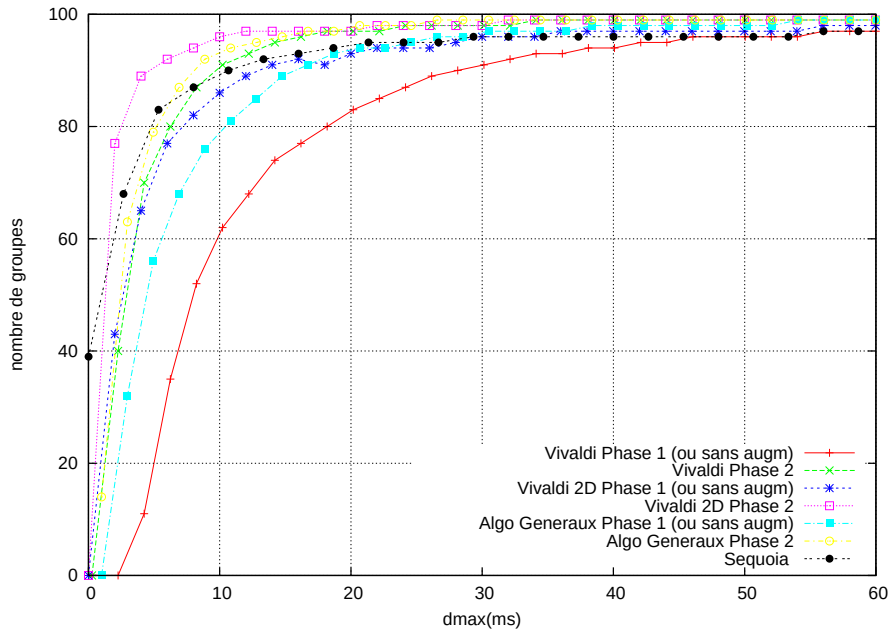


FIG. 5.2 – Variation du nombre de groupes construits en fonction du $d_{\max}$ fixé, en utilisant différents outils de plongement.

Nous ne présentons ici que les courbes concernant le nombre de groupes créés, le diamètre moyen des groupes créés ainsi que le nombre de groupes valides, qui nous ont semblé à la fois

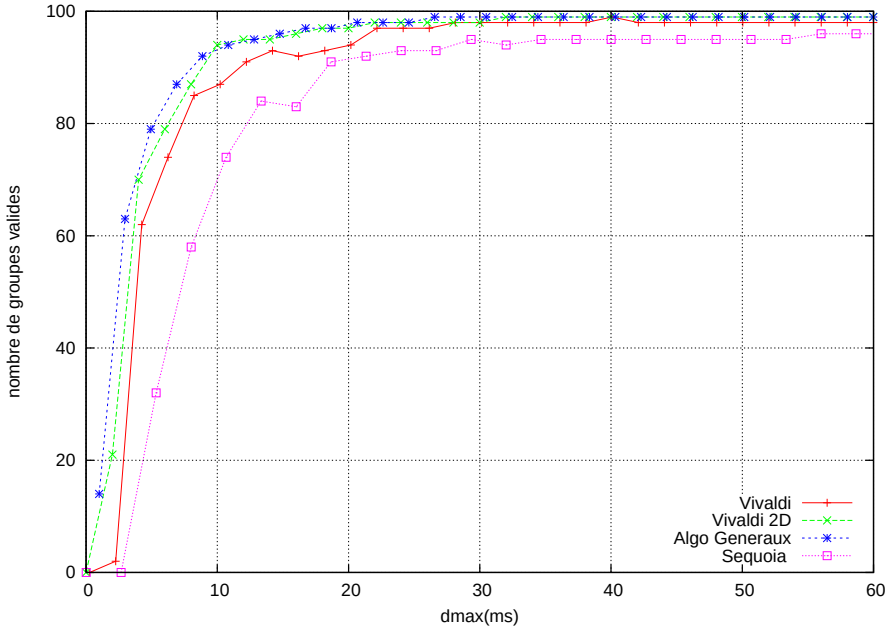


FIG. 5.3 – Variation du nombres de groupes valides construits en fonction du $d_{\max}$ fixé, en utilisant différents outils de plongement.

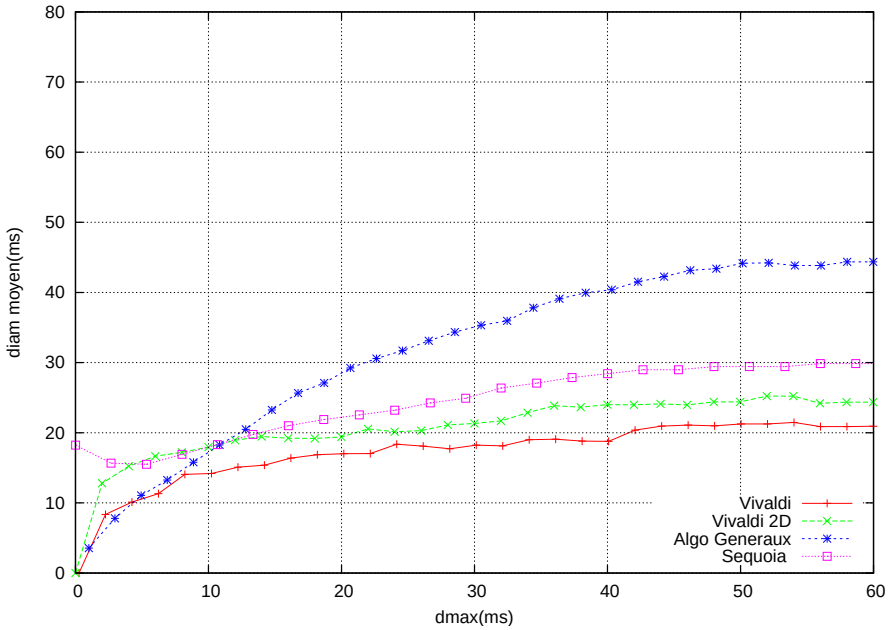


FIG. 5.4 – Variation du diamètre moyen dans la matrice d'origine des groupes construits en fonction du $d_{\max}$ fixé, en utilisant différents outils de plongement.

être les courbes les plus intéressantes à observer, mais également les courbes les plus pertinentes à étudier en vue de tirer une conclusion quant au choix d'un des différents outils de plongement utilisés. Nous présentons également, dans la Figure 5.5, une courbe concernant le nombre de groupes construits pour une contrainte de poids différente, $W = 0.1$.

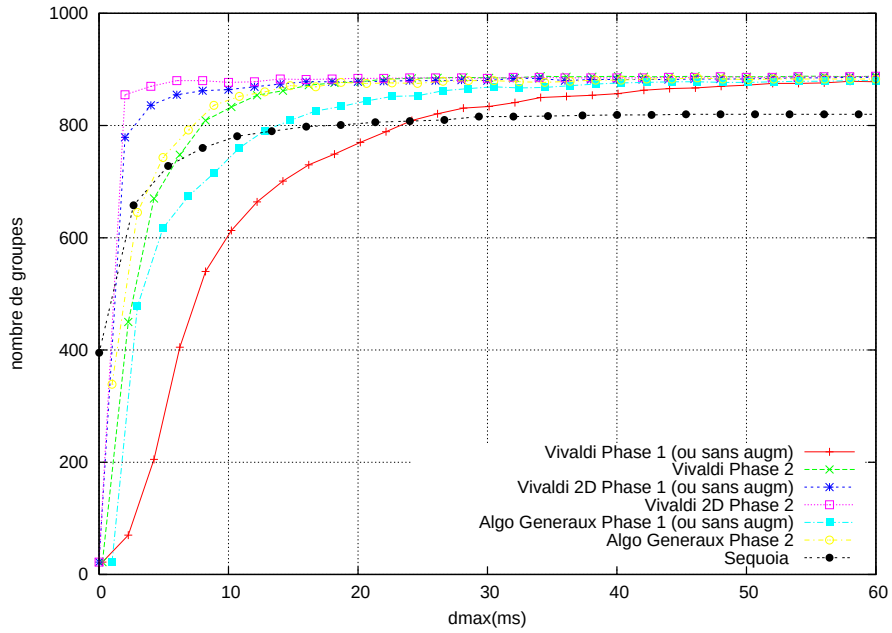


FIG. 5.5 – Variation du nombre de groupes construits en fonction du $d_{\max}$ fixé, en utilisant différents outils de plongement.

De manière générale, on peut observer qu'hormis l'algorithme n'utilisant pas d'augmentation de ressources (et, donc, plus contraint que les algorithmes auxquels on le compare), tous les algorithmes construisent sensiblement le même nombre de groupes. Ils en construisent moins de 100, puisque le poids total des éléments est de 100 (et $W = 1$). Cependant, quand $d_{\max}$ est suffisamment grand, BCCD se ramène au problème de BIN COVERING classique. En remarquant que le poids d'un groupe est contraint à être supérieur à 1, et étant donnée la distribution des poids utilisée (le poids maximal d'un élément est de 0.27, est beaucoup d'éléments sont de poids très faible), il n'est pas étonnant d'observer que le nombre de groupes construits s'approchent beaucoup de 100. Par ailleurs, on peut observer que cela intervient pour des valeurs de $dmax$ faible : au delà de $30ms$ ($10ms$ pour Vivaldi 2D), tous les algorithmes se comportent de la même façon et construisent presque 100 groupes. Ceci semble laisser penser que, une fois un tel $d_{\max}$ atteint, BCCD sur ces instances devient un problème de BIN COVERING classique. La contrainte prédominante est celle du poids et non celle de la distance.

L'observation de la Figure 5.3 nous permet de remarquer qu'hormis pour Sequoia, les groupes construits sont tous valides, *i.e.* tout groupe construit respecte la contrainte de distance fixée par l'algorithme utilisé (*i.e.* en prenant en compte l'utilisation d'une éventuelle augmentation de ressources). Sequoia semble donc moins efficace pour travailler sur BCCD, en raison de la distortion induite par le plongement dans un arbre.

La dernière courbe, celle de la Figure 5.4, nous présente la variation du diamètre moyen des groupes construits en fonction de la contrainte de distance imposée. On peut observer que, quel que soit l'algorithme et le plongement utilisé, bien qu'au delà de $30ms$ le nombre de groupes construits ne varie plus (cf. Figure 5.2), le diamètre moyen des groupes continue de croître (même légèrement à la fin, mais toujours). Plus la contrainte de distance est lâche, plus les groupes exploiteront cette souplesse et créeront des groupes plus "distendus". En vue d'une utilisation pratique de ces algorithmes, ceci montre qu'il est important de bien calibrer la contrainte de

distance souhaitée. Par ailleurs cette courbe montre également que, même si tous les algorithmes permettent d'obtenir des diamètres moyens inférieurs à celui de l'algorithme de base (qui utilise une augmentation de ressources de 4 en s'appuyant directement sur la matrice de latence), Vivaldi semble être le plus efficace.

Notons que les valeurs prises par le diamètre moyen des groupes dans Sequoia est grande au début en raison de la distortion induite par le plongement dans un arbre : dans l'arbre, les éléments d'un même groupe sont proches les uns des autres, mais ne le sont pas dans la matrice de latences d'origine. Enfin, remarquons que les valeurs que prennent les diamètres moyens ne sont pas nécessairement croissantes en fonction du $d_{\max}$, mais que l'augmentation de la contrainte de distance peut permettre la construction de groupes de diamètres moins importants.

Le nombre d'éléments moyen dans un groupe est de 25 (quand $W = 1$), quel que soit $d_{\max}$. Le diamètre d'un ensemble de 25 éléments tirés aléatoirement est de $52ms$. On remarque que le diamètre moyen des groupes tend vers des valeurs bien inférieures à $52ms$ pour toutes les expérimentations effectuées. En effet, les algorithmes que nous utilisons tendent à privilégier la construction de groupes de petit diamètre.

La dernière courbe, présentée dans la Figure 5.5, concerne le nombre de groupes construits par les différents algorithmes pour une contrainte de poids de $W = 0.1$. On peut remarquer que, même si le comportement global des courbes est tout à fait similaire à celui des courbes pour $W = 1$, leur convergence n'est pas identique. En effet, on pourrait s'attendre à ce que les courbes convergent vers un nombre de groupes égal à 1000, correspondant au nombre maximum de groupes constructibles pour une telle contrainte de poids, mais ce n'est pas le cas. Ceci s'explique par le fait que la contrainte de poids est bien plus proche des poids des éléments ($W = 1$ correspond à une contrainte de poids très grande devant les poids des éléments), et, ainsi, il arrive plus souvent que des groupes dépassent sensiblement un poids de 0.1, même quand la contrainte de distance est devenue grande et n'intervient plus. Ainsi on se retrouve confrontés à un problème d'approximation de Bin Covering classique, et on ne construit pas nécessairement le nombre optimal de groupes.

5.2.3 Comparaison Vivaldi/Sequoia pour BPCD

L'utilisation de Sequoia permet de plonger l'ensemble des éléments d'une instance dans un arbre. Dans le Paragraphe 4.7 nous avons prouvé le Théorème 4.46 qui affirme l'existence d'un $(\frac{5}{2}, 1)$-algorithme d'approximation pour BPCD quand les éléments d'une instance sont les sommets d'un arbre. Cependant, étant donnée la complexité de cet algorithme, nous avons choisi d'utiliser l'Algorithme 2 du Paragraphe 3.4.

Nous utilisons cet algorithme aussi bien lorsqu'on utilise Vivaldi pour modéliser l'espace des latences, que lorsque Sequoia est utilisé. Nous identifions chacune des corubes présentées dans chaque figure par le nom de l'outil de plongement utilisé pour la générer, respectivement Vivaldi (pour un plongement dans un plan augmenté), Vivaldi 2D (pour un plongement dans un plan euclidien), Sequoia et "AlgoGeneraux" (sans utiliser aucun plongement, en travaillant directement avec la matrice de latences).

Ainsi, pour le même algorithme, nous pouvons mesurer l'efficacité de l'outil de plongement selon plusieurs critères, respectivement le nombre de groupes créés, le diamètre maximal d'un des groupes créés, et le diamètre moyen des groupes créés, et ce pour différentes valeurs de $d_{\max}$. Les

résultats obtenus ont été tracés sur quatre courbes, présentées dans les Figures 5.6, 5.7, 5.8 et 5.9. Nous présentons également, dans la Figure 5.10, une courbe concernant le nombre de groupes construits pour une contrainte de poids différente, $W = 0.1$.

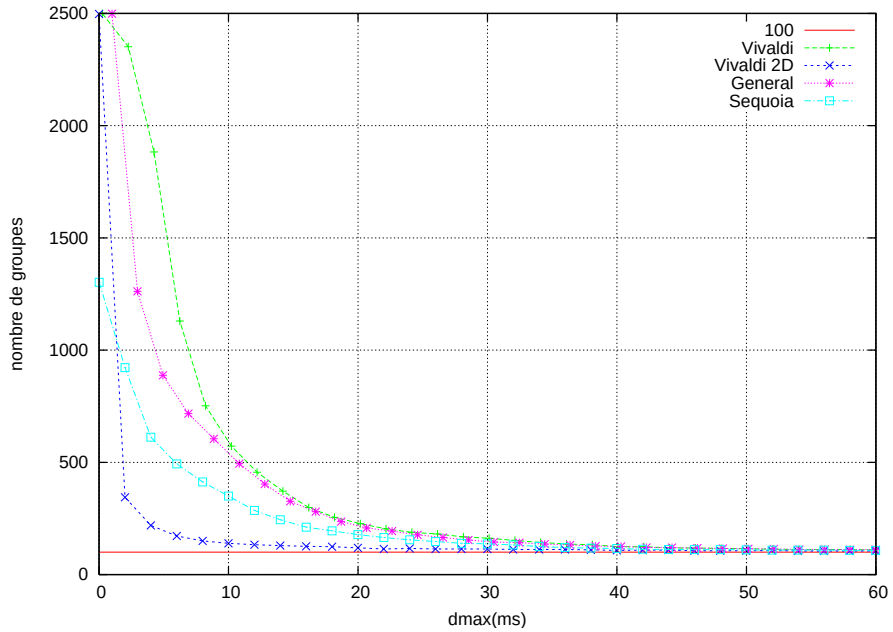


FIG. 5.6 – Variation du nombre de groupes construits en fonction du $d_{\max}$ fixé, pour différents outils de plongement. Les courbes sont tronqués pour des nombres de groupes supérieurs à 700. En effet, quand la contrainte de distance est petite, tous les éléments sont nécessairement placés seuls dans un groupe (et donc on a 2500 groupes au début).

Nous présentons les courbes concernant le nombre de groupes créés, le diamètre moyen des groupes créés ainsi que le nombre moyen d'éléments par groupe créé.

On peut tout d'abord remarquer que, comme pour BCCD, quel que soit le plongement utilisé, l'algorithme construit très vite un nombre de groupes constant. L'algorithme utilisé dans le plan semble converger plus vite, tandis que Vivaldi (plongeant les éléments dans un plan augmenté) semble moins vite converger. Mais au delà de $30ms$, les courbes sont les mêmes, le nombre de groupes créés est constamment très proches de 100. Rappelons que 100 est une borne inférieure sur le nombre de groupes créés, puisque le poids total des éléments est de 100, et que $W = 1$. Il semble donc qu'au delà d'un $d_{\max}$ de $30ms$ on soit ramené à un problème de BIN PACKING classique.

La Figure 5.7 présente le pourcentage de groupes construits respectant la contrainte de distance imposée sur le diamètre dans la matrice d'origine, en prenant en compte l'augmentation de ressources autorisée (*i.e.* on compte le pourcentage de groupes construits de diamètre inférieur à $2d_{\max}$). Pour les petites valeurs de $d_{\max}$, Vivaldi 2D se comporte très mal. Il construit plus vite moins de groupes, mais beaucoup ont un diamètre violant les contraintes de diamètres. Vivaldi quant à lui semble de prime abord moins efficace pour de petites valeurs de $d_{\max}$, comme on peut le voir dans la Figure 5.6, mais tous les groupes construits en utilisant ce plongement sont valides. La distortion induite par Vivaldi 2D est bien sensible ici : l'algorithme, dans le plan, est certes plus efficace que les autres (il est plus facile de construire des groupes dans le plan), mais la distortion que provoque le plongement des noeuds dans le plan est très coûteuse. Sequoia semble être un bon compromis entre efficacité et validité des groupes construits.

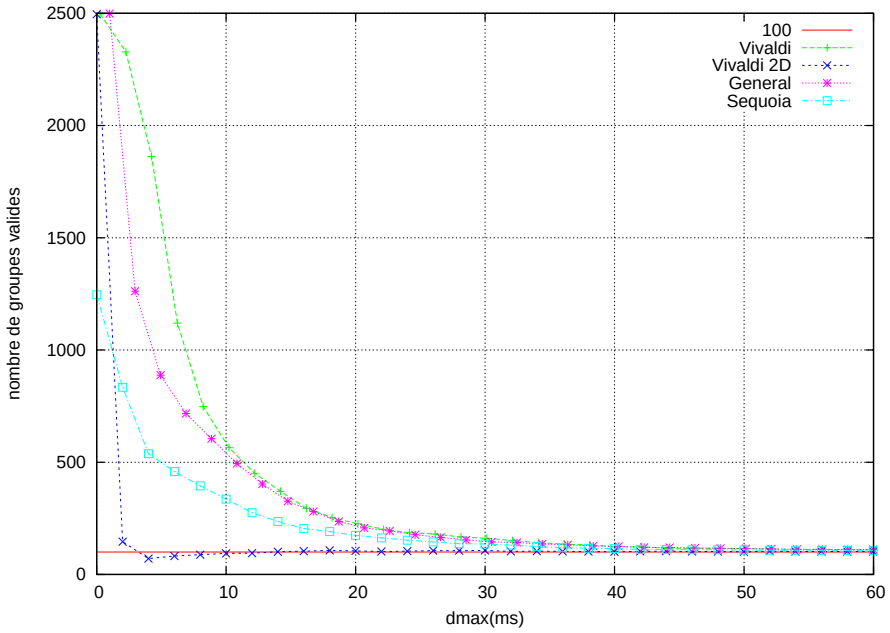


FIG. 5.7 – Variation du pourcentage de groupes valides construits en fonction du $d_{\max}$ fixé, pour différents outils de plongement.

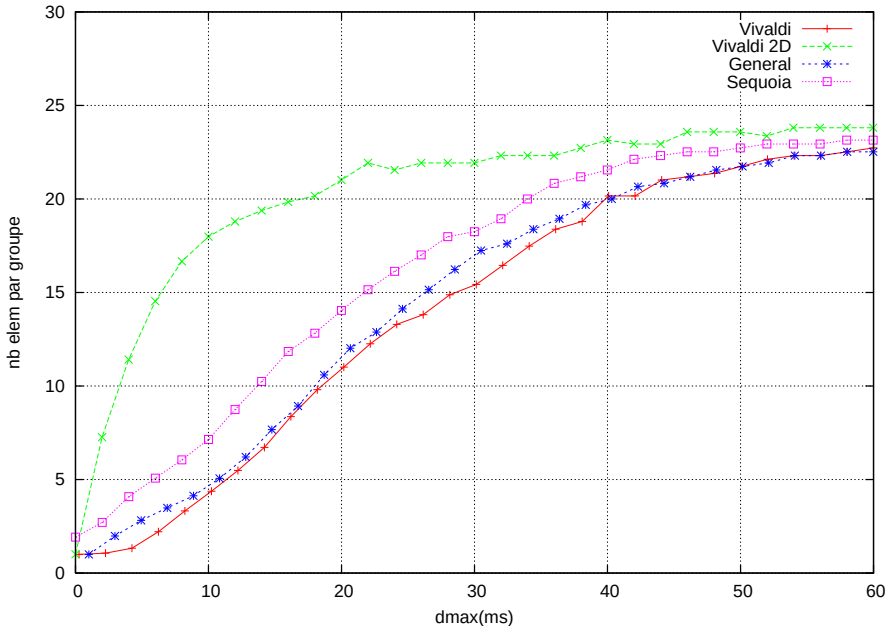


FIG. 5.8 – Variation du nombre moyen d'éléments par groupe construit en fonction du $d_{\max}$ fixé, pour différents outils de plongement.

La Figure 5.8, qui présente la variation du nombre moyen d'éléments par groupe en fonction du $d_{\max}$, nous indique que les groupes comptent moins de 25 éléments, ce qui semble être le nombre idéal pour construire un minimum de groupes, en sachant que le poids total est de 100. La croissance de la courbe nous laisse penser que c'est la contrainte de distance qui empêche

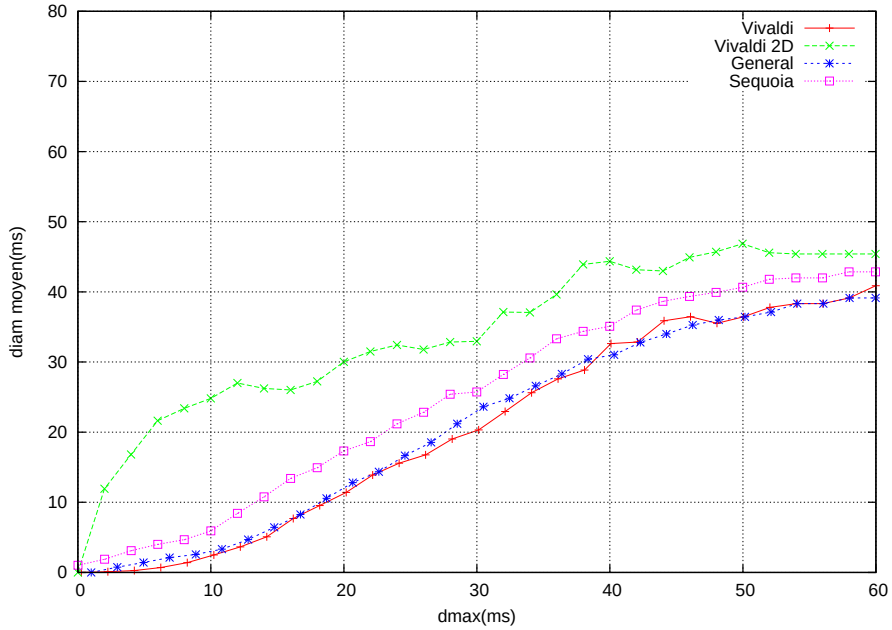


FIG. 5.9 – Variation du diamètre moyen des groupes construits en fonction du $d_{\max}$ fixé, pour différents outils de plongement.

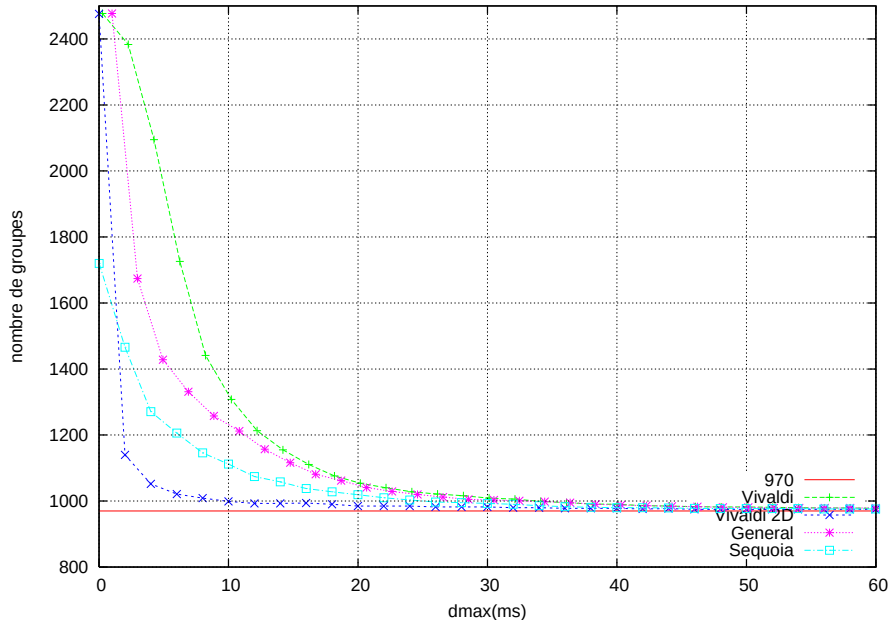


FIG. 5.10 – Variation du nombre de groupes construits en fonction du $d_{\max}$ fixé, pour différents outils de plongement.

l'algorithme de construire des groupes contenant plus d'éléments. Plus cette contrainte se relâche, plus les groupes peuvent contenir d'éléments, de poids de plus en plus proche de $W = 1$. Une fois que le nombre d'éléments par groupe se stabilise, *i.e.* que la contrainte de distance est suffisamment grande pour ne plus se faire ressentir, BPCD devient un problème de c̃lassique.

On peut toutefois remarquer que le nombre moyen d'éléments dans un groupe construit par l'exécution de l'algorithme dans le plan croît plus vite vers un nombre d'éléments par groupe aux alentours de 25, ce qui coïncide avec l'observation de la courbe précédente, qui présentait ce plongement comme le plus efficace relativement au nombre de groupes construits.

Enfin la dernière courbe pour $W = 1$, présentée dans la Figure 5.9, concerne le diamètre moyen des groupes construits. On y observe un net écart entre l'utilisation de Vivaldi pour plonger les éléments dans le plan euclidien et les autres courbes. Ceci, combiné aux observations précédentes, signifie que cette utilisation de Vivaldi permet de construire plus de groupes pour de plus petites valeurs de $d_{\max}$, mais que les diamètres moyens des groupes alors créés sont en réalité en moyenne plus important que les groupes créés en utilisant d'autres plongements. Ce diamètre moyen peut être supérieur à $2d_{\max}$ car il mesure le diamètre des groupes dans la matrice d'origine, et non dans le plongement, et donc mesure également la distortion induite par le plongement utilisé. Considérons alors, par exemple, $d_{\max} = 5ms$: Vivaldi 2D construit des groupes de diamètre $20ms$ en moyenne, tandis que Vivaldi construit des groupes de diamètres très faible, d'environ $2ms$ en moyenne. Le diamètre moyen des groupes construits en utilisant Vivaldi n'atteint une valeur de $20ms$ que pour $d_{\max} = 30ms$. On peut alors comparer le nombre de groupes créés par Vivaldi 2D avec un $d_{\max}$ de $5ms$, qui construit 180 groupes environ, au nombre de groupes créés par Vivaldi avec un $d_{\max}$ de $30ms$, qui est de 160, et observer que finalement les deux sont tout à faits comparables.

La courbe présentant le nombre de groupes construits pour une contrainte de poids $W = 0.1$, présentée dans la Figure 5.10, est très similaire à son équivalent pour $W = 1$. C'est le cas pour toutes les mesures effectuées, pour toutes les valeurs de W choisies. C'est pour cela que nous avons choisi de ne pas présenter les autres courbes équivalentes. Toutes les courbes convergent vers un ensemble de 1000 groupes, correspondant au nombre minimal de groupes qu'il est nécessaire de construire pour une telle contrainte de poids. L'expérience correspondante dans le cadre de BCCD montre que moins de groupes sont construits quand la contrainte de poids se rapproche des poids moyens des éléments. Ici, on observe moins ce phénomène sur cette courbe, mais il est bien présent. Rappelons en effet que lorsqu'on travaille sur BPCD, tous les éléments possédant un poids supérieur à la contrainte de poids fixée sont éliminés de l'instance, et ne sont plus placés dans des groupes (dans BCCD, tout noeud ayant un poids dépassant la contrainte de poids est placé seul dans un groupe, et est donc bien pris en compte). Le poids total des éléments n'est, en l'occurence, pas de 100, mais seulement de 97. La borne inférieure sur le nombre de goupes qu'il est possible de construire est donc de 970, et non de 1000. En prenant en compte cette observation, on constate bien sur la courbe de la Figure 5.10 que le nombre de groupes construits par nos algorithmes n'atteint pas cette borne, même quand la contrainte de distance s'efface pour laisser place à un problème d'approximation de Bin Packing classique.

Chapitre 6

Conclusion

Cette thèse regroupe un ensemble de travaux qui trouvent leur application dans les réseaux à grande échelle, tel Internet. Dans ces réseaux, fortement hétérogènes et de grande taille, nous avons étudié deux types de problèmes :

- Un problème d'agrégation de ressources, dans lequel chaque noeud possède une quantité de ressources, et dans lequel l'objectif est de mettre en place un maximum de groupes possédant une quantité de ressources suffisante pour assurer une certaine tâche. Dans chacun de ces groupes, les noeuds ne doivent pas être trop éloignés les uns des autres en vue d'effectuer un travail en commun. Nous avons modélisé ce problème via Bin Covering *avec Contrainte de Distance* (BCCD).
- Un problème de placement de serveurs, dans lequel chaque client requiert une quantité de ressources spécifique auprès d'un serveur à qui il doit être affecté. Dans ce problème l'objectif est d'identifier un minimum de groupes à associer respectivement à un minimum de serveurs en respectant la capacité préalablement fixée de chacun des serveurs. On souhaite également qu'un serveur n'ait pas à sa charge deux noeuds trop éloignés l'un de l'autre, et que donc la distance entre deux noeuds d'un même groupe soit bornée. Nous avons modélisé ce problème en utilisant Bin Packing *avec Contrainte de Distance* (BPCD).

Dans le premier chapitre nous avons abordé ces deux problèmes dans le cas où la structure du réseau reliant les noeuds est modélisée par un espace métrique quelconque.

Dans le Chapitre 3 nous avons étudié l'inapproximabilité de ces deux problèmes dans ce cadre. Nous avons tout d'abord prouvé que BCCD comme BPCD n'est pas approximable à facteur constant quelle que soit l'augmentation de ressources strictement inférieure à 2 sur le diamètre des groupes construits qu'on s'autorise à utiliser. Nous avons par ailleurs démontré, d'un côté que BCCD n'est pas approximable à un facteur strictement supérieur à $1/2$, et de l'autre que BPCD n'est pas approximable à un facteur strictement inférieur à $3/2$, et ce quelle que soit l'augmentation de ressources qu'on s'autorise à utiliser. Ces deux derniers résultats sont valables quels que soient les espaces métriques considérés.

Nous avons ensuite présenté dans ce même chapitre un $(2/5, 4)$-algorithme d'approximation centralisé pour BCCD, qui fonctionne dans n'importe quel espace métrique. Parallèlement, nous proposons dans le Paragraphe 3.4 un $(7/3, 2)$-algorithme d'approximation pour BPCD qui fonctionne également dans n'importe quel espace métrique. Nous avons étendu ce résultat à la version uniforme de BPCD, pour obtenir un $(2, 2)$-algorithme d'approximation pour cette variante.

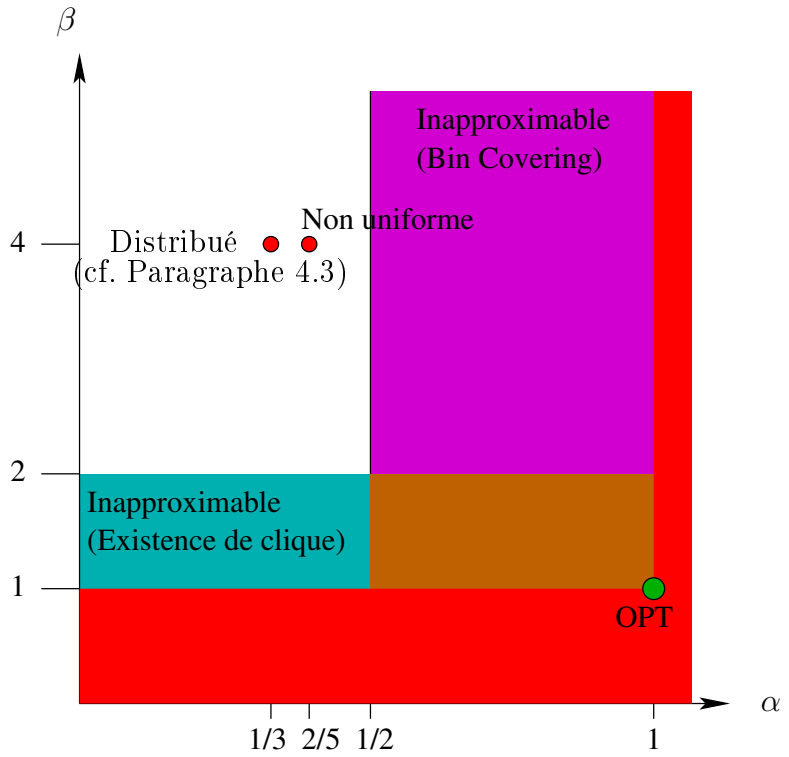


FIG. 6.1 – Résumé des résultats concernant BCCD dans un espace métrique général.

La Figure 6.1 résume les résultats d'inapproximabilité et d'approximation obtenus pour BCCD dans un espace métrique quelconque. Les parties colorées correspondent aux couples (α, β) pour lesquels nous avons prouvé que BCCD n'est pas (α, β)-approximable en temps polynomial.

La Figure 6.2 résume les résultats d'inapproximabilité et d'approximation obtenus pour BPCD dans un espace métrique quelconque. Les parties colorées correspondent aux couples (α, β) pour lesquels nous avons prouvé que BPCD n'est pas (α, β)-approximable en temps polynomial. Est également indiqué sur cette figure le $(2, 2)$-algorithme d'approximation pour la variante uniforme de BPCD, tiré du résultat pour BPCD non uniforme. Rappelons que les résultats d'inapproximabilité de BPCD ne s'appliquent pas tous à la variante uniforme de BPCD. Ainsi dans la Figure 6.2, si l'on considère l'inapproximabilité de la variante uniforme, il ne faut pas considérer comme colorée la partie turquoise.

De manière générale, et comme ces deux figures le montrent bien, il reste encore un écart à combler entre les résultats d'inapproximabilité prouvés pour chacun des deux problèmes étudiés, et les résultats d'approximation obtenus dans un cadre général. Dans le cas de BCCD en particulier, cet écart est important, puisqu'en théorie rien ne prouve que BCCD n'est pas approximable en utilisant une augmentation de ressources de 2, tandis que nous ne parvenons à l'approcher qu'avec une augmentation de ressources de 4. Cependant la preuve de l'inapproximabilité de BCCD peut se faire de deux façons bien différentes, via une réduction au problème de détection de l'existence d'une clique, comme nous l'avons fait dans le Paragraphe 3.2, ainsi que via une réduction au problème de Couplage k-Dimensionnel Maximal, comme présenté dans le Paragraphe 4.1. Ces deux réductions bien différentes mettent en exergue que BCCD est compliqué

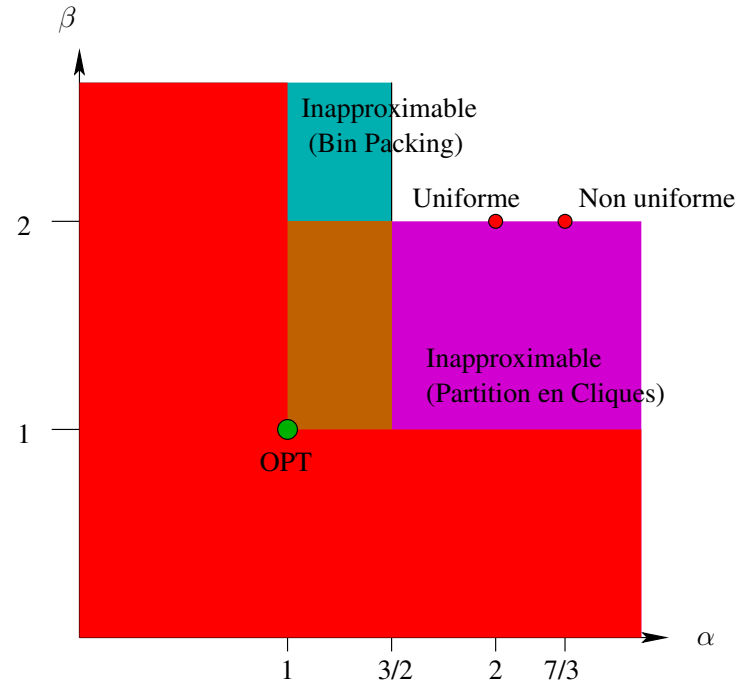


FIG. 6.2 – Résumé des résultats concernant BPCD dans un espace métrique général. On indique le $(2, 2)$-algorithme d'approximation pour la variante uniforme de BPCD bien que les résultats d'inapproximabilité de BPCD ne s'étendent pas tous à cette variante.

pour plusieurs raisons. Ainsi nous conjecturons qu'il est NP-difficile de trouver un algorithme d'approximation à facteur constant pour BCCD en utilisant une augmentation de ressources strictement inférieur à 4. Intuitivement, cela viendrait du fait qu'une augmentation de ressources de 2 au moins est nécessaire pour résoudre l'aspect de BCCD lié à sa réduction au problème d'existence d'une clique, qui consiste en la détection des groupes valides, et qu'une augmentation de ressources d'un facteur 2 supplémentaire serait également nécessaire pour, étant donné l'ensemble des groupes valides, résoudre le problème de choisir lesquels construire pour maximiser le nombre de groupes finaux.

Enfin, toujours dans des espaces métriques quelconques, nous avons étudié la façon dont les résultats que nous avons obtenu pour BPCD s'adaptent à Kcapa, uniforme et non uniforme. Nous obtenons ainsi, en plus de résultats d'inapproximabilité intéressants, le premier algorithme d'approximation connu pour Kcapa non uniforme, assurant une approximation de 4 pour une augmentation de ressources sur le nombre de centres utilisés de 7/3.

Les perspectives concernant ces problèmes sont légèrement différentes. En effet, nous avons montré qu'un algorithme d'approximation pour BPCD peut être adapté en un algorithme d'approximation pour Kcapa. Via cette correspondance, une amélioration de notre algorithme d'approximation pour BPCD (en terme de facteur d'approximation aussi bien qu'en terme d'augmentation de ressources) aurait pour conséquence directe l'amélioration des résultats de Khuller *et al.* dans [KS00] sur Kcapa. Par ailleurs le facteur d'approximation que nous avons obtenu pour la version non uniforme de BPCD (et qui s'étend à Kcapa non uniforme) ne diffère que d'un facteur de 7/6 du facteur d'approximation pour la version uniforme, en utilisant une augmentation de ressources identique et égale à 2. En couplant ces observations au fait qu'avec une augmentation

de ressources inférieure à 2 les deux variantes de BPCD sont impossibles à approcher en temps polynomial, nous pensons qu'une approche différente de la nôtre devra sans doute être envisagée pour améliorer les résultats que nous avons obtenu.

Dans le Chapitre 4 nous avons étudié BCCD et BPCD dans des espaces métriques plus restreints, qui correspondent à plusieurs modélisations classiques du réseau composé des noeuds d'Internet et des latences les reliant.

Pour BCCD, nous avons obtenu les résultats suivants :

- un algorithme d'approximation à facteur constant ne nécessitant aucune augmentation de ressources dans le cas où les éléments sont plongés dans un espace vérifiant certaines propriétés. Cet algorithme s'exécute en temps polynomial dans les espaces métriques dans lesquels la détection et la construction d'un groupe valide de diamètre $d_{\max}$ peut se faire en temps polynomial (en particulier, les espaces métriques dans lesquels le graphe de compatibilité fait partie d'une classe de graphes dans laquelle la recherche d'une clique de poids maximum se fait en temps polynomial),
- en utilisant cet algorithme dans le plan muni de la norme L_∞, on obtient un $(\frac{1}{11}, 1)$-algorithme d'approximation, ne nécessitant donc aucune augmentation de ressources,
- un $(\frac{2}{5}, 3)$-algorithme d'approximation dans le plan muni d'une norme quelconque,
- un $(\frac{1}{3}, 4)$-algorithme d'approximation distribué pour dans $\mathbb{R}^D$ muni de la norme L_∞. Ce résultat a été publié dans [BBDL08],
- un $(\frac{1}{3}, 2)$-algorithme d'approximation lorsque les éléments d'une instance sont plongés dans un arbre (contenant éventuellement des noeuds virtuels, ne correspondant pas à des éléments de l'instance).

Le $(\frac{1}{11}, 1)$-algorithme d'approximation dans le plan muni de la norme L_∞ est notamment intéressant car il prouve que si les éléments d'une instance de BCCD sont placés dans le plan, il n'est pas nécessaire d'avoir recours à une augmentation de ressources pour obtenir une approximation à facteur constant de la solution optimale. Cependant cet algorithme est glouton, un algorithme plus élaboré pourrait peut être aboutir à un meilleur facteur d'approximation.

Le $(\frac{2}{5}, 3)$-algorithme d'approximation dans le plan muni d'une norme quelconque s'étend à tout espace métrique dans lequel il est possible de construire en temps polynomial des groupes de diamètre $d_{\max}$ jusqu'à ce qu'il n'existe plus de groupe valide d'un tel diamètre, puis de faire la même chose avec des groupes de diamètre $3d_{\max}$. Il suffit donc que dans un tel espace métrique un algorithme permettant la détection et la construction de groupes valides s'exécutant en temps polynomial existe. En particulier dans les espaces dans lesquels la résolution du problème de trouver une clique de poids maximum sur le graphe $\mathrm{Comp}(\mathcal{I}, d)$, $d > 0$ se fait en temps polynomial, ce résultat s'applique. Par exemple, si $\mathrm{Comp}(\mathcal{I}, d)$ fait partie d'une classe de graphes où ce problème est solvable (*e.g.* la classe des graphes parfaits).

Il convient cependant de prendre garde au fait que sur de tels espaces métriques, les résultats d'inapproximabilité ne sont plus toujours vrais. Ainsi le résultat précédent nous assure de l'existence d'un $(\frac{2}{5}, 3)$-algorithme d'approximation dans ce type d'espace, mais aucun résultat, *a priori*, ne nous apprend que BCCD n'est pas approximable à facteur constant dans ce cadre.

Dans un environnement distribué, construire des groupes de diamètre $d_{\max}$ dans le plan muni d'une norme choisie apparait comme difficile, de par la synchronie nécessaire, entre autres, à l'exécution d'un algorithme de flot dans un graphe biparti. L'augmentation de ressources de 4 nécessaire à l'exécution du $(\frac{1}{3}, 4)$-algorithme d'approximation distribué dans $\mathbb{R}^D$ muni de L_∞

que nous avons présenté est donc requise même si on exécute cet algorithme dans un plan.

Ce $(\frac{1}{3}, 4)$-algorithme d'approximation distribué pour dans $\mathbb{R}^D$ muni de la norme L_∞ utilise une structure de données appelée "skip-graph" pour organiser les éléments d'une instance. Nous avons prouvé dans [DL10] que la hauteur de cette structure de données, *i.e.* la longueur maximale d'un chemin de recherche d'un élément à un autre dans cette structure, est majorée par $c \log n$, où c est une constante dont la valeur est connue. Pour les skip-lists (une autre structure de données hautement corrélée aux skip-graphs), la borne supérieure est associée à une borne inférieure équivalente [Dev92], de telle sorte que la hauteur d'une skip-list, normalisée par $\log_2 n$, tend en probabilité vers une autre constante connue. Nos travaux ne nous ont pas permis de trouver une borne inférieure correspondante pour les skip-graphs (autre que celle existante pour les skip-lists), et nos résultats expérimentaux semblent indiquer que notre majoration n'est pas fine. La question de la convergence en probabilité de la hauteur normalisée d'un skip-graph vers une certaine constante reste donc ouverte. Les techniques que nous avons utilisées n'ont pas grandes chances de donner de meilleurs résultats, et une analyse plus fine des relations de dépendance entre les différentes skip-lists d'un skip-graph semble nécessaire.

Enfin, l'étude de BCCD dans les arbres utilisés par Sequoia était motivée par l'idée qu'en restreignant l'espace métrique des instances de BCCD à une classe de graphes simple comme celle des arbres nous allions être en mesure d'obtenir de meilleures approximations. Nous avons présenté un $(\frac{1}{3}, 2)$-algorithme d'approximation pour BCCD dans les arbres. Cependant, nous pensons que ce même algorithme est peut être un $(\frac{1}{2}, 2)$-algorithme d'approximation. Par ailleurs, il est important de remarquer que ce résultat utilise une augmentation de ressources d'un facteur 2 alors qu'aucun résultat d'inapproximation ne nous permet de nous assurer qu'il est impossible d'approcher une solution optimale au BCCD dans les arbres sans avoir recours à une augmentation de ressources quelconque. Une piste de recherche qui nous intéresse est de prouver que, dans les arbres, BCCD n'est pas approximable à facteur constant sans avoir recours à une augmentation de ressources d'un facteur supérieur ou égal à 2.

Nous avons mené le même type d'étude concernant BPCD, et obtenu les résultats suivants :

- une famille d'algorithmes assurant une $(\frac{5}{2} + \varepsilon, 1)$-approximation dans le plan euclidien, ne nécessitant donc aucune augmentation de ressources,
- un $(\frac{5}{2}, 1)$-algorithme d'approximation lorsque les éléments d'une instance sont plongés dans un arbre (du même type qu'auparavant, *i.e.* pouvant contenir des noeuds virtuels).

On peut remarquer, de la même façon que pour BCCD, que le résultat dans le plan est intéressant puisqu'il prouve que dans certains espaces métriques, l'utilisation d'une augmentation de ressources n'est pas nécessaire.

On peut s'inspirer de la famille d'algorithmes présentée dans le cadre du plan euclidien pour obtenir des algorithmes pouvant être exécutés dans tout graphe dans lequel une approximation à facteur constant du problème de partition en cliques de cardinalité minimale peut être construite en temps polynomial. Le facteur d'approximation obtenu découle alors directement du facteur d'approximation de l'algorithme de partition en cliques.

Enfin il semble clair que l'amélioration de ces deux résultats obtenus pour BPCD requiert soit une amélioration des techniques utilisées par Epstein et Levin dans leurs travaux sur BPAC [EL08], soit l'utilisation d'une approche différente de résolution de ce problème dans ces espaces métriques particuliers. Soulignons cependant que, comme ces résultats n'utilisent pas d'augmentation de ressource, une amélioration des facteurs d'approximation obtenus pourrait

avoir des conséquences intéressantes sur les résultats correspondant concernant BPAC.

En guise de perspectives plus générale, d'autres approches de nos problèmes peuvent être envisagées : nous avons tout d'abord relâché nos contraintes de distance en travaillant avec de l'augmentation de ressources pour obtenir des résultats dans des espaces métriques généraux. Nous avons ensuite utilisé des travaux traitant des possibles plongements des réseaux à grande échelle qui nous intéressent (en particulier, Internet), dans des espaces métriques plus simples à étudier. Nous avons observé que, bien que ces plongements semblent efficaces, ils ne couvrent pas tous les aspects nécessaires à une modélisation correcte des réseaux de grande échelle. Les violations d'inégalités triangulaires, par exemple, sont souvent ignorées ou très peu considérées, tandis que nos résultats y sont très sensibles. Une autre approche pourrait être d'envisager ces réseaux comme munis de variantes des inégalités triangulaires classiques. Considérer Internet comme une infra-métrique est une première piste, examinée par Lebhar *et al.* dans [LFV08]. Dans ces travaux, l'inégalité triangulaire est relachée, et présentée de la façon suivante : $d(u,v) \leq \rho.\max(d(u,w),d(v,w))$, pour tout triplet de noeuds (u,v,w). Leurs résultats semblent montrer qu'on pourrait modéliser Internet par un espace vérifiant ce type d'inégalité relachée, pour $\rho = 2$. L'adaptation directe de nos algorithmes à ce cadre d'études ne change pas les augmentations de ressources théoriques nécessaires à nos algorithmes. Cependant des expérimentations pratiques en adaptant nos algorithmes à des espaces infra-métriques pourrait produire des résultats intéressants si on ne se contente plus d'ignorer les violations d'inégalité triangulaire, comme nous l'avons fait, mais qu'elles sont incluses dans ce concept d'infra-métrique.

Il peut être intéressant de se pencher sur une généralisation de BPCD, qui fournit une formulation plus globale aux problèmes de placements de serveurs qui nous intéressent : dans le même cadre de réseaux à grande échelle, séparons l'ensemble des noeuds et "serveurs potentiels" et "clients" (certains peuvent être membres des deux ensembles). Chaque serveur possède une capacité (qu'on pourra considérer uniforme) et chaque client une demande quantitativement valuée. Etant donnée une instance composée de ces deux ensembles de noeuds, et d'une contrainte de distance $d_{\max}$ fixée, l'objectif est de sélectionner un minimum de serveurs en vue d'affecter la demande de chaque client à un serveur, de telle sorte que chaque client soit à moins de $d_{\max}$ du serveur auquel il est affecté. Une brève étude de ce problème permet d'observer qu'il est impossible à approcher à facteur constant en temps polynomial sans utiliser une augmentation de ressources supérieure ou égale à 3 sur la distance entre un serveur et un client lui étant affecté (à moins que $P = NP$, et ce via une réduction au problème de *Set Cover*). Bien qu'en pratique, il semble que des algorithmes gloutons assez simples produisent des résultats très bons, nous ne sommes pas parvenus à prouver théoriquement qu'ils étaient effectivement efficaces. Ce problème offrant une généralisation intéressante et une approche différente des problèmes de placements de serveurs classiques nous semble une piste de recherche fructueuse.

Enfin, n'oublions pas qu'un des paramètres principaux de tous les réseaux auxquels nous nous intéressons est leur dynamisme. Dans ce manuscrit, nous nous sommes plutôt concentrés sur leur hétérogénéité, en mettant de côté leur dynamisme, mais il est nécessaire, en vue d'une application pratique, d'examiner le réalisme de nos algorithmes dans un cadre qui, en plus d'être distribué, est hautement dynamique. L'utilisation d'un skip-graph conjointement à un Z-ordre, par exemple, offre une structure de données distribuée et maintenable dans un environnement dynamique. En pratique cependant, tout dépend de la fréquence des arrivées et départs de noeuds, et ce type de structure peut déjà paraître trop complexe. Rappelons cependant que, comparées aux plateformes pair-à-pair, les plateformes de grande échelle comme BOINC ont généralement un dynamisme modéré, les mêmes noeuds revenant souvent et pendant des périodes assez longues.

Pour revenir sur l'approche que nous avons menée, il semble clair, *a posteriori*, que travailler sur des espaces métriques généraux, sur ce type de problèmes, n'a pas vraiment de sens. En effet, il s'agit de problèmes théoriquement complexes, comme en témoignent les résultats d'inapproximabilité et le besoin d'avoir recours à de l'augmentation de ressources. Travailler dans un espace métrique quelconque complexifie encore ces problèmes. D'autre part les réseaux auxquels ces problèmes s'appliquent ne correspondent pas à des espaces métriques quelconques mais possèdent tous des particularités exploitables pour simplifier le cadre dans lequel on se place.

De nombreuses études évoquent en effet les propriétés des réseaux "humains" (sociaux, Internet, etc) qui satisfont tous de nombreuses propriétés communes. Il semble alors plus intéressant de chercher à développer des algorithmes destinés directement à être appliqués à des espaces métriques correspondant aux réseaux visés.

La seconde approche que nous avons envisagée nous a permis de prendre conscience d'un certain manque de modélisations efficaces d'Internet, en particulier pour y exécuter des algorithmes complexes. En effet, Vivaldi comme Sequoia semblent fournir des plongements dans des espaces métriques relativement simples, dans lesquels il est possible d'obtenir des estimations de latence relativement précises. Cependant l'imprécision de ces prédictions, même faible, suffit à rendre des algorithmes complexes peu efficaces, notamment en raison de la présence de violations d'inégalités triangulaires, mal modélisée par ces outils. En effet, deux éléments tous deux proches d'un même troisième élément peuvent, en cas de violation de l'inégalité triangulaire, se trouver loin l'un de l'autre.

Ainsi avoir de bonnes prédictions de la latence séparant chaque paire de noeuds ne suffit pas à avoir de bonnes prédictions des diamètres d'ensemble d'éléments, puisqu'il suffit d'une valeur absurde dans un groupe pour que le diamètre de celui-ci explose. Le nombre de mesures de distances entre éléments d'un même groupe croissant exponentiellement avec la taille des ensembles d'éléments considérés, plus les groupes sont nombreux et de cardinalité importante, plus ils sont sensibles aux violations d'inégalités triangulaires.

Par ailleurs, les espaces métriques obtenus par les plongements utilisant les différents outils existants ne nous ont pas permis d'effectuer un bond en terme de complexité algorithmique. L'un des intérêts souhaitable de ces plongements est en effet de pouvoir travailler sur des espaces métriques dans lesquels les opérations à effectuer sont grandement facilitées, de par la nature de l'espace considéré. On pourrait donc espérer obtenir d'encore meilleurs résultats dans ces espaces.

Il semble qu'il y ait un travail important à faire dans la modélisation d'Internet en particulier, qui prenne en compte les violations d'inégalité triangulaires comme faisant partie intégrante de l'espace des latences, et non comme de légères imperfections des mesures. L'objectif est de construire un outil qui parvienne à la fois à plonger les noeuds dans un espace métrique simple, dans lequel des algorithmes complexes peuvent être grandement simplifiés (en terme de complexité), et à fournir des prédictions en terme de latence qui soient les plus précises possibles.

Les travaux de modélisation effectués actuellement plongent les noeuds dans des espaces dans lesquels la distance entre deux points est la plus courte distance entre ces deux points dans l'espace de plongement. Ce type de modélisation a été bien étudié et semble finalement relativement limitée par la structure même d'Internet, qui n'est fondamentalement pas un espace métrique, ce qui fait que toute tentative de le plonger dans un espace métrique aboutira toujours à d'irréductibles distortions, non maitrisables.

En effet dans Internet les paquets ne circulent pas nécessairement en suivant les plus courts chemins entre les noeuds, mais suivent des trajets complexes, dépendant de politiques de routage souvent locales et non nécessairement reliées à des objectifs de performance et d'efficacité (mais liées à des politiques d'échanges entre différents fournisseurs d'accès à Internet, par exemple). Une piste est peut être d'envisager un plongement de l'espace des latences dans un espace non métrique, par exemple une classe de graphes particulière munie d'un algorithme de routage, qui, bien qu'idéalement simple, permettrait de mieux modéliser le fait que dans Internet ce ne sont pas nécessairement les plus courts chemins qui sont employés. Voir Internet comme l'agrégation de communautés à différents niveaux de hiérarchie pourrait également être une piste intéressante.

Enfin, la question se pose de la pertinence des problèmes auxquels nous nous sommes intéressés. En effet nous considérons la latence comme nécessaire à minimiser lorsque plusieurs noeuds sont destinés à s'échanger fréquemment des messages sans évoquer, par exemple, les problèmes de bande passante. Dans un premier temps rappelons que les problèmes que nous étudions sont plutôt destinés à modéliser des échanges de données de petites tailles mais très fréquents.

Par ailleurs, la latence est déjà un point critique dans de nombreux domaines, comme les jeux vidéos [Arm03]. En observant l'évolution des technologies actuelles, il semble que les progrès en terme de latence tendent à perdre de la vitesse. En effet, les grandes distances sont couvertes par des liaisons s'approchant de plus en plus de la vitesse de la lumière, à un facteur environ 30 au plus (un ping entre une université bordelaise et une université parisienne de $15ms$ correspond à une vitesse de $40000km.s^{-1}$, ce qui est moins de 10 fois moins que la vitesse de la lumière), qui est une borne physique maximale qui semble ne pas pouvoir être dépassée. En parallèle, les puissances de calcul (au moins d'ordinateurs placés en parallèle ou de multi-coeurs), les capacités de stockage et les bandes passantes ne semblent pas être sur le point d'atteindre leurs limites physiques. Ceci semble nous indiquer que dans les années à venir, les problèmes de latence se feront de plus en plus présents, et que les types d'agrégations de ressources que nous avons étudiés, sous forme de groupes de faible latence, pourraient devenir primordiaux.

Bibliographie

[ABK+07] Ittai Abraham, Mahesh Balakrishnan, Fabian Kuhn, Dahlia Malkhi, Venugopalan Ramasubramanian, and Kunal Talwar. Reconstructing approximate tree metrics. In *PODC '07 : Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 43–52, New York, NY, USA, 2007. ACM.

[ACK+02] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@home : an experiment in public-resource computing. *Commun. ACM*, 45(11) :56–61, 2002.

[AEG05] L. Arge, D. Eppstein, and M.T. Goodrich. Skip-webs : efficient distributed data structures for multi-dimensional data sets. *Proceedings of the twenty-fourth annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 69–76, 2005.

[AIKS91] Alok Aggarwal, Hiroshi Imai, Naoki Katoh, and Subhash Suri. Finding k points with minimum diameter and related problems. *J. Algorithms*, 12(1) :38–56, 1991.

[AJKL84] S.F. Assmann, D.S. Johnson, D.J. Kleitman, and J.Y.T. Leung. On a dual version of the one-dimensional bin packing problem. *Journal of algorithms*, 5(4) :502–525, 1984.

[And04] David P. Anderson. Boinc : A system for public-resource computing and storage. In *GRID '04 : Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[Arm03] G. Armitage. An experimental estimation of latency sensitivity in multiplayer Quake 3. In *Proceedings of the 11th IEEE International Conference on Networks (ICON 2003)*, pages 137–141, 2003.

[AS03] J. Aspnes and G. Shah. Skip graphs. *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 384–393, 2003.

[AS07] James Aspnes and Gauri Shah. Skip graphs. *ACM Transactions on Algorithms*, 3(4) :37–56, November 2007.

[AW05] James Aspnes and Udi Wieder. The expansion and mixing time of skip graphs with applications. In *SPAA '05 : Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 126–134, New York, NY, USA, 2005. ACM.

[B+92] M. Biró et al. Precoloring extension. I. Interval graphs. *Discrete Mathematics*, 100(1-3) :267–279, 1992.

[BBDL08] O. Beaumont, N. Bonichon, Ph. Duchon, and H. Larchevêque. Distributed approximation algorithm for resource clustering. In *OPODIS 2008*, volume 5401 of *Lecture Notes in Computer Science*, pages 564–567. Springer, 2008.

[Ber60] C. Berge. Les problemes de coloration en théorie des graphes. *Publ. Inst. Stat. Univ. Paris*, 9 :123–160, 1960.

[BGS95] M. Bellare, O. Goldreich, and M. Sudan. Free bits, pcps and non-approximability-towards tight results. *Foundations of Computer Science, Annual IEEE Symposium on*, 0 :422, 1995.

[BIKP93] J. Bar-Ilan, G. Kortzars, and D. Peleg. How to allocate network centers. *J. Algorithms*, pages 15 :385–415, 1993.

[CCDC90] B.N. Clark, S. Colbourn David, and J. Charles. Unit disk graphs. *Discrete mathematics*, 86(1-3) :165–177, 1990.

[CDK+04] R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris. Practical, distributed network coordinates. *ACM SIGCOMM Computer Communication Review*, 34(1) :113–118, 2004.

[Chu06] Julia Chuzhoy. Covering problems with hard capacities. *SIAM J. Comput.*, 36(2) :498–515, 2006.

[CJK01] J. Csirik, D.S. Johnson, and C. Kenyon. Better approximation algorithms for bin covering. *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 557–566, 2001.

[COK02] Y. Chen, C. Overton, and RH Katz. Internet Iso-bar : A scalable overlay distance monitoring system. *Journal of Computer Resource Management, Computer Measurement Group, Spring Edition*, 2002.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC '71 : Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.

[CW99] Fabián A. Chudak and David P. Williamson. Improved approximation algorithms for capacitated facility location problems. In *IPCO*, pages 99–113, 1999.

[CW02] János Csirik and Gerhard J. Woeginger. Resource augmentation for online bounded space bin packing. *J. Algorithms*, 44(2) :308–320, 2002.

[DCKM04] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi : a decentralized network coordinate system. *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 15–26, 2004.

[Dev92] Luc Devroye. A limit theory for random skip lists. *The Annals of Applied Probability, Vol. 2, No. 3*, pages 597–609, Aug. 1992.

[DL10] Philippe Duchon and Hubert Larchevêque. On the search path length of random binary skip graphs. In *6th ACM Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, Austin, 2010.

[dlVL81] W. Fernandez de la Vega and G.S. Lueker. Bin packing can be solved within $1+\varepsilon$ in linear time. *Combinatorica*, pages 1 :349–355, 1981.

[DP09] A. Dumitrescu and J. Pach. Minimum clique partition in unit disk graphs. *CoRR abs/0909.1552*, 2009.

[Edm65] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17(3) :449–467, 1965.

[EGCGJ97] Jr. E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing : a survey. pages 46–93, 1997.

[EK75] S. Even and O. Kariv. An O (n2. 5) algorithm for maximum matching in general graphs. In *Foundations of Computer Science, 1975., 16th Annual Symposium on*, pages 100–112, 1975.

[EL08] L. Epstein and A. Levin. On Bin Packing with Conflicts. *SIAM Journal on Optimization*, 19(3) :1270–1298, 2008.

[EvS07] Leah Epstein and Rob van Stee. Online bin packing with resource augmentation. *Discrete Optimization*, 4(3-4) :322–333, 2007.

[EZ04] TS Eugene and N.H. Zhang. A Network Positioning System for the Internet. In *Proceedings of the 4th Symposium on Internet Technologies and Systems ; USENIX*. Citeseer, 2004.

[FJJ+01] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. IDMaps : A global Internet host distance estimation service. *IEEE/ACM Transactions on Networking (TON)*, 9(5) :525–540, 2001.

[GJ+79] M.R. Garey, D.S. Johnson, et al. *Computers and Intractability : A Guide to the Theory of NP-completeness*. WH freeman San Francisco, 1979.

[GNS06] Michael T. Goodrich, Michael J. Nelson, and Jonathan Z. Sun. The rainbow skip graph : a fault-tolerant constant-degree distributed data structure. In *SODA '06 : Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 384–393, New York, NY, USA, 2006. ACM.

[Gon85] T.F. Gonzalez. Clustering to minimize the intercluster distance. *Theoretical Computer Science*, pages 38 :293–306, 1985.

[GSG02] P. K. Gummadi, S. Saroiu, and S. D. Gribble. King : estimating latency between arbitrary internet end hosts. *Computer Communication Review*, 32(3) :11, 2002.

[Hås99] J. Håstad. Clique is hard to approximate withinn 1- ε. *Acta Mathematica*, 182(1) :105–142, 1999.

[HJS+03] N.J.A. Harvey, M.B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet : A scalable overlay network with practical locality properties. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems-Volume 4*, page 9. USENIX Association, 2003.

[HS85] Dorit S. Hochbaum and David B. Shmoys. A Best Possible Heuristic for the k-Center Problem. *Mathematics of Operations Research*, 10(2) :180–184, 1985.

[HSS06] E. Hazan, O. Schwartz, and S. Safra. On the Hardness of Approximating k-Set-Packing. *Computational Complexity*, 15(1) :20–39, 2006. Preliminary version in APPROX'03.

[HT93] M. Hujter and Z. Tuza. Precoloring extension. II. Graph classes related to bipartite graphs. *Acta Math. Univ. Comenianae*, 62(1) :1–11, 1993.

[HT08] M. Hujter and Z. Tuza. Precoloring extension iii : Classes of perfect graphs. *Combinatorics, Probability and Computing*, 5(01) :35–56, 2008.

[Joh74] D.S. Johnson. Approximation algorithms for combinatorial problems*. *Journal of Computer and System Sciences*, 9(3) :256–278, 1974.

[KLL09] M.J. Kao, C.S. Liao, and D.T. Lee. Capacitated domination problem. *Algorithmica*, pages 1–27, 2009.

[KM07] Fabian Kuhn and Thomas Moscibroda. Distributed approximation of capacitated dominating sets. In *SPAA '07 : Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pages 161–170, New York, NY, USA, 2007. ACM.

[KPR00] M.R. Korupolu, C.G. Plaxton, and R. Rajaraman. Analysis of a Local Search Heuristic for Facility Location Problems. *Journal of Algorithms*, 37 :146188, 2000.

[KS00] S. Khuller and Y.J. Sussmann. The Capacitated K-Center Problem. *SIAM Journal on Discrete Mathematics*, 13 :403, 2000.

[KST93] J. Köbler, U. Schöning, and J. Torán. *The graph isomorphism problem : its structural complexity.* Springer, 1993.

[Lav95] C. Lavault. Evaluation des algorithmes distribués. *Hermès, Paris*, 1995.

[LBSB09] C. Lumezanu, R. Baden, N. Spring, and B. Bhattacharjee. Triangle inequality and routing policy violations in the Internet. *Passive and Active Network Measurement*, pages 45–54, 2009.

[LFV08] Emmanuelle Lebhar, Pierre Fraigniaud, and Laurent Viennot. The inframetric model for the internet. In *Proceedings of the 27th IEEE International Conference on Computer Communications (INFOCOM)*, pages 1085–1093, Phoenix, 2008.

[LHC03] Hyuk Lim, Jennifer C. Hou, and Chong-Ho Choi. Constructing internet coordinate system based on delay measurement. In *IMC '03 : Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 129–142, New York, NY, USA, 2003. ACM.

[LMY98] R. Lupton, F.M. Maley, and N. Young. Data Collection for the Sloan Digital Sky Survey–A Network-Flow Heuristic. *Journal of Algorithms*, 27(2) :339–356, 1998.

[LP86] L. Lovász and M.D. Plummer. *Matching theory.* Elsevier Science Ltd, 1986.

[LS06] C. Lumezanu and N. Spring. Playing vivaldi in hyperbolic space. *Proc. of SIGCOMM-IMC*, 2006.

[Lyn96] N.A. Lynch. *Distributed algorithms.* Morgan Kaufmann, 1996.

[MBI+95] MV Marathe, H. Breu, H.B.H. Iii, SS Ravi, and DJ Rosenkrantz. Simple Heuristics for Unit Disk Graphs. *Networks*, 25 :59–68, 1995.

[MPS92] J.I. Munro, T. Papadakis, and R. Sedgewick. Deterministic skip lists. *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pages 367–375, 1992.

[NZ02] T.S.E. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *IEEE INFOCOM*, volume 1, pages 170–179. Citeseer, 2002.

[Ore90] Jack Orenstein. A comparison of spatial query processing techniques for native and parameter spaces. *SIGMOD Rec.*, 19(2) :343–352, 1990.

[P+81] S. Paz et al. Non deterministic polynomial optimization problems and their approximations. *Theoretical Computer Science*, 15(3) :251–277, 1981.

[PCW+] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Lighthouses for scalable distributed location. *Peer-to-Peer Systems II*, pages 278–291.

[Pel00] D. Peleg. *Distributed computing : a locality-sensitive approach.* Society for Industrial Mathematics, 2000.

[PTW01] M. Pál, É. Tardos, and T. Wexler. Facility location with nonuniform hard capacities. In *FOCS '01 : Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 329, Washington, DC, USA, 2001. IEEE Computer Society.

[Pug90] W. Pugh. Skip Lists : A Probabilistic Alternative to Balanced Trees. *Communications of the ACM*, 33(6), 1990.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. *A scalable content-addressable network*. ACM Press New York, NY, USA, 2001.

[RMK+09] Venugopalan Ramasubramanian, Dahlia Malkhi, Fabian Kuhn, Mahesh Balakrishnan, Archit Gupta, and Aditya Akella. On the treeness of internet latency and bandwidth. In *SIGMETRICS '09 : Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, pages 61–72, New York, NY, USA, 2009. ACM.

[RS97] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 475–484. ACM, 1997.

[SMK+01] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord : A scalable peer-to-peer lookup service for internet applications. *Proceedings of the 2001 SIGCOMM conference*, 31(4) :149–160, 2001.

[STA97] D.B. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *In Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, 1997.

[TC03] L. Tang and M. Crovella. Virtual landmarks for the internet. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, page 152. ACM, 2003.

[TSRB71] C. Toregas, R. Swain, C. ReVelle, and L. Bergman. The location of emergency service facilities. *Operations Research*, 19(6) :1363–1373, 1971.

[W+01] D.B. West et al. *Introduction to graph theory*. Prentice Hall Upper Saddle River, NJ, 2001.

[XSLB04] Z. Xu, P. Sharma, S.J. Lee, and S. Banerjee. Netvigator : Scalable network proximity estimation. *HP Laboratories Technical Report, HPL-2004-28*, 2004.

[Yue91] M. Yue. A simple proof of the inequality $FFD(L) \leq 11/9OPT(L) + 1, \forall L$ for the FFD bin-packing algorithm. *Acta Mathematicae Applicatae Sinica (English Series)*, 7(4) :321–331, 1991.

Index