Titre: Title:	Résolution exacte du problème de partitionnement de données avec minimisation de variance sous contraintes de cardinalité par programmation par contraintes
Auteur: Author:	Mohammed Najib Haouas
Date:	2020
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Haouas, M. N. (2020). Résolution exacte du problème de partitionnement de données avec minimisation de variance sous contraintes de cardinalité par programmation par contraintes [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/4207/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Résolution exacte du problème de partitionnement de données avec minimisation de variance sous contraintes de cardinalité par programmation par contraintes

MOHAMMED NAJIB HAOUAS
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Janvier 2020

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Résolution exacte du problème de partitionnement de données avec minimisation de variance sous contraintes de cardinalité par programmation par contraintes

présenté par **Mohammed Najib HAOUAS**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Amal ZOUAQ, présidente
Gilles PESANT, membre et directeur de recherche
Daniel ALOISE, membre et codirecteur de recherche
Claudio CONTARDO, membre

DÉDICACE

à mes parents et à ma brillante soeur,
pour tous leurs sacrifices et leur soutien...

REMERCIEMENTS

J'aimerais remercier mes directeurs de recherche, Monsieur Pesant et Monsieur Aloise, pour leur brillant et infaillible encadrement tout au long de mon projet. Leurs idées, leur patience et leur expertise m'ayant aidées à surmonter tous les obstacles.

J'aimerais également remercier le Conseil de recherches en sciences naturelles et en génie du Canada ainsi que le ministère de l'éducation et de l'enseignement supérieur du Québec pour leur précieuse aide financière.

RÉSUMÉ

Le partitionnement de données représente une procédure destinée à regrouper un ensemble d'observations dans plusieurs sous ensembles homogènes et/ou bien séparés. L'idée derrière une telle activité est de simplifier l'extraction d'information utile en étudiant les groupes résultants plutôt que les observations elles-mêmes.

Cela dit, plusieurs situations appellent à ce que la solution générée respecte un ensemble de contraintes données. En particulier, on exige parfois que les groupes résultants comportent un nombre prédéfini d'éléments. On parle de partitionnement avec contraintes de cardinalité.

On présente alors, dans ce travail, une approche de résolution exacte pour le partitionnement de données avec minimisation de la variance sous contraintes de cardinalité. En utilisant le paradigme de la Programmation par Contraintes, on propose d'abord un modèle adéquat du problème selon celui-ci. Ensuite, on suggère à la fois une stratégie de recherche rehaussée ainsi que deux algorithmes de filtrage. Ces outils ainsi développés tirent avantage de la structure particulière du problème afin de naviguer l'espace de recherche de façon efficace, à la recherche d'une solution globalement optimale.

Des expérimentations pratiques montrent que notre approche procure un avantage important par rapport aux autres méthodes exactes existantes lors de la résolution de plusieurs exemplaires du problème.

ABSTRACT

Data clustering is a procedure designed to group a set of observations into subsets that are homogeneous and/or well separated. The idea behind such an endeavor is to simplify extraction of useful information by studying the resulting groups instead of directly dealing with the observations themselves.

However, many situations mandate that the answer conform to a set of constraints. Particularly one that involves the target number of elements each group must possess. This is known as cardinality constrained clustering.

In this work we present an exact approach to solve the cardinality constrained Euclidian minimum sum-of-squares clustering. Based on the Constraint Programming paradigm, we first present an adequate model for this problem in the aforementioned framework. We then suggest both an upgraded search heuristic as well as two filtering algorithms. We take advantage of the structure of the problem in designing these tools to efficiently navigate the search space, looking for a globally optimal solution.

Computational experiments show that our approach provides a substantial boost to the resolution of several instances of the problem in comparison to existing exact methods.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

ABS	Activity-based Search
ccMSSC	Cardinality Constrained MSSC
CL	Cannot-Link, Constraint
COP	Constraint Optimization Problem
CP	Constraint Programming
CPC	CP Clustering
CPO	CP Optimizer, IBM
CP RBBA	CP Repetitive Branch and Bound
CSP	Constraint Satisfaction Problem
DFS	Depth-First Search
DP	Dynamic Programming
FDS	Failure-directed Search
GCC	Global Cardinality Constraint
IBS	Impact-based Search
KKT	Karush-Kuhn-Tucker, Conditions
LB	Lower Bound
LIMA	Less is more Approach
LNS	Large Neighborhood Search
LP	Linear Program
MCF	Minimum-cost Flow
ML	Must-Link, Constraint
MSSC	Minimum Sum of Squares Clustering
RBBA	Repetitive Branch and Bound Algorithm
TU	Totally Unimodular, Matrix
UB	Upper Bound
VNS	Variable Neighborhood Search
WCSS	Within Cluster Sum of Squares

CHAPITRE 1 INTRODUCTION

Le traitement de données fait parfois intervenir un type de problèmes dont le but est de trouver une solution qui satisfait un ensemble de contraintes prédéfinies. Selon le contexte, la solution recherchée a non seulement besoin de satisfaire ces contraintes, mais a aussi besoin de respecter un certain critère de « qualité. » De façon usuelle, on parle d'un coût de solution. Dans le cas de la fouille de données, ces problèmes d'optimisation peuvent se présenter, entre autres, comme des problèmes de partitionnement selon un certain critère. La résolution de ces problèmes s'avère être importante pour extraire de l'information utile à partir d'un ensemble d'observations. Afin de comprendre les éléments relatifs à ce travail de recherche et dans une perspective d'informer le lecteur sur la portée de celui-ci, on s'intéresse d'abord au partitionnement de données ainsi qu'à la motivation derrière l'usage de contraintes dans de celui-ci. Ensuite, on parle de la définition des problèmes de satisfaction des contraintes ainsi que de celle des problèmes d'optimisation sous contraintes. Enfin, on définit ce qu'est la programmation par contraintes, l'outil proposé pour résoudre ces problèmes d'optimisation combinatoire.

1.1 Le partitionnement et son rôle dans la fouille de données

La fouille des données est devenue un enjeu majeur. Dans une ère marquée par la collecte des données, il est devenu primordial de trouver des façons efficaces et rapides afin d'en extraire de l'information utile [1].

Une façon d'accomplir cela est à travers le regroupement des observations en entités homogènes. En effet, il est plus facile et plus utile d'étudier des macro-groupes représentatifs des observations que les observations mêmes. On parle alors de partitionnement ou de *clustering* de données. Afin de réaliser un tel partitionnement, on se base sur une mesure qui quantifie la différence entre chaque observation (i.e. une mesure de dissemblance ou *dissimilarity*). Cependant, il y a plusieurs définitions « d'homogénéité » et, dépendamment du critère choisi, plusieurs partitions sont alors possibles [1].

La grande majorité des problèmes de partitionnement se présente comme des problèmes d'optimisation en lien avec les mesures de dissemblances entre les observations. Le critère d'homogénéité se manifeste alors comme la fonction-objectif à optimiser [1].

Formellement, un problème de clustering se présente comme suit [2] :

Définition 1.1.1. Soient $O = \{o_1, o_2, ..., o_n\}$ un ensemble d'observations et $d : O^2 \mapsto \mathbb{R}^+$ une mesure de dissemblance (pas nécessairement une distance). Une k-partition $\Delta \in \mathcal{A}$ de O (avec $\mathcal{A}$ l'ensemble des partitions possibles) est telle que :

$$\forall c \in \{1, ..., k\}, C_c \neq \varnothing, \cup_c C_c = O \text{ et } \forall c \neq c', C_c \cap C_{c'} = \varnothing$$

Soit le critère $\gamma_d : \mathcal{A} \mapsto \mathbb{R}^+$ un critère de partitionnement qui utilise la mesure d. Δ^* est une partition optimale si $\Delta^* = \operatorname{argmin} \gamma_d$ (resp. $\Delta^* = \operatorname{argmax} \gamma_d$). ◊

Exemple 1.1.1. Soit les 4 points suivants :

$$o_1 : (1, 0) \quad o_2 : (4, 4) \quad o_3 : (0, 1) \quad o_4 : (5, 5)$$

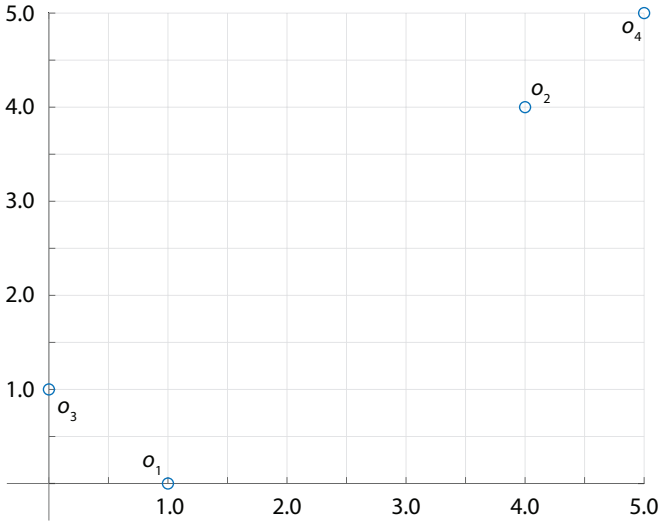

Figure 1.1 Observations à partitionner sur $\mathbb{R}^2$

On désire trouver une partition de deux groupes ($k = 2$) qui minimise le critère suivant :

$$\gamma_d = \sum_{c \in \{1,...,k\}} \sum_{o \in C_c} d(o, \bar{o}_c) \qquad \text{tel que} \qquad \bar{o}_c \text{ centroïde de } C_c$$

Si d est la distance euclidienne au carré, la solution est telle que :

$$\Delta = \{\{o_1, o_3\}, \{o_2, o_4\}\}$$

avec :

— $C_1 = \{o_1, o_3\}$ et $C_2 = \{o_2, o_4\}$;

— $\bar{o}_1(0.5, 0.5)$ et $\bar{o}_2(4.5, 4.5)$;

— $\gamma_d{}^* = (0.5+0.5)+(0.5+0.5) = 2$, valeur de la fonction-objectif évaluée à cette solution optimale.

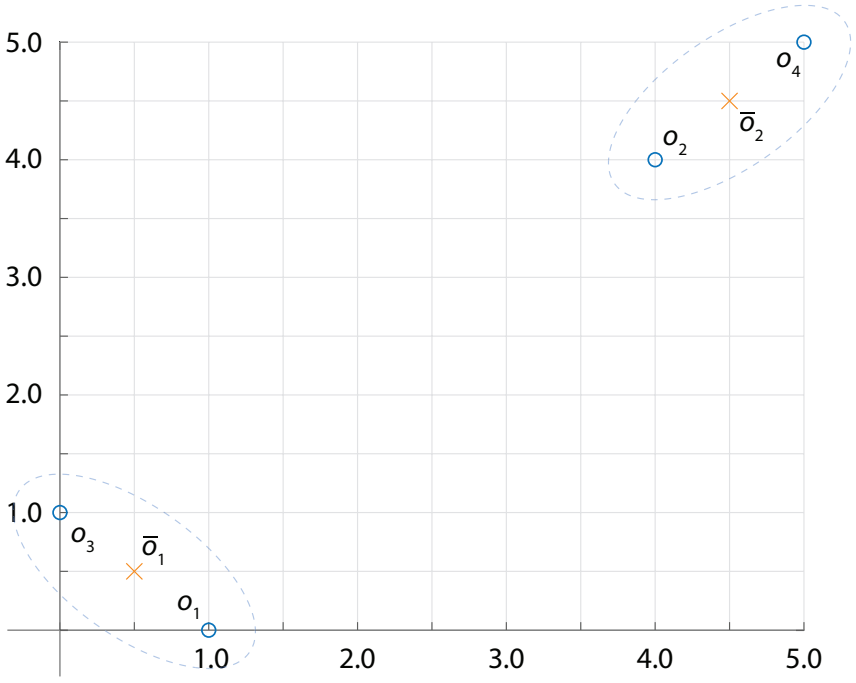

Figure 1.2 Observations partitionnées

△

1.2 Le partitionnement sous contraintes

Souvent, la résolution des problèmes de partitionnement n'engage que l'optimisation du critère d'homogénéité. On parle d'un processus d'apprentissage non-supervisé. En effet, la réponse, sous forme de groupes anonymes, est déduite d'un ensemble de données non-libellées et ce par un procédé automatique et systématique (l'optimisation) [3].

Ceci est par contraste aux processus d'apprentissage supervisés où les données d'apprentissage sont libellées par un avis d'expert ou des constatations a priori. La réponse, à son tour, prend la forme de classes complètement caractérisées. Il est à noter que libeller les observations induit un coût implicite dans la mesure où il s'agit d'un effort non négligeable, voire impossible [3].

Entre ces deux extrêmes se trouve l'apprentissage semi-supervisé. On retrouve dans celui-ci une caractérisation partielle des données d'apprentissage. Dans le cas du clustering, ceci se traduit par des contraintes sur l'appartenance des points à un groupe où encore par des contraintes sur les caractéristiques du groupe lui-même (taille, forme...) [3].

L'apprentissage semi-supervisé a tendance à améliorer la qualité des résultats de façon considérable [3].

1.3 Les CSP

Les problèmes de satisfaction de contraintes ou *Constraint Satisfaction Problems* (CSP) sont des problèmes combinatoires où l'on cherche une instanciation complète d'un ensemble de variables munies de domaines respectifs. Ladite instanciation se doit de respecter un ensemble de contraintes qui agissent chacune sur un sous-ensemble des variables du problème.

De façon formelle [4] :

Définition 1.3.1. Un CSP est un triplet CSP $= \langle X, D, C \rangle$ où :

- X est l'ensemble des variables : $X = \{x_1, x_2..., x_n\}$;
- D est le domaine des variables tel que D est l'ensemble des valeurs qui peuvent être prises par les variables dans X. Le domaine d'une variable $x \in X$, noté $D(x)$, est un sous-ensemble de D ;
- C est l'ensemble des contraintes : $C = \{c_1, c_2..., c_m\}$. Pour chaque contrainte $c \in C$, on définit le sous-ensemble $X(c) \subseteq X$ qui représente les variables sur lesquelles c agit. Ce sous-ensemble s'appelle la portée de la contrainte c.

$\Diamond$

Définition 1.3.2. Soit $c \in C$ et $X(c) = \{x_1, x_2..., x_k\} \subseteq X$ alors $c \subseteq D(x_1) \times D(x_2) \times ... \times D(x_k)$. On dit que le k-tuple $(v_1, v_2, ..., v_k)$ est une solution de c si, et seulement si, $(v_1, v_2, ..., v_k) \in c$. $\Diamond$

Résoudre un CSP revient à trouver une affectation qui instancie chaque variable dans X, telles que les valeurs prises respectent toutes les contraintes dans C. Ou alors, en d'autres termes :

Définition 1.3.3. Une solution à CSP $= \langle X, D, C \rangle$ correspond à une fonction d'affectation τ définie comme :

$$\tau : x \in X \mapsto v \in D(x) \qquad \text{telle que} \qquad \forall c \in C, \tau(X(c)) \in c$$

$\Diamond$

Exemple 1.3.1. Soit le problème CSP $= \langle X, D, C \rangle$ tel que :

- $X = \{x_1, x_2, x_3\}$
- $D = \{0, 1, 2\}$
- $C = \{x_1 \neq 0, x_2 = x_3, x_3 = 2\}$
- $D(x) = D \quad \forall x \in X$

alors :

- $X(x_2 = x_3) = \{x_2, x_3\}$
- $\{x_2 = x_3\} = \{(0, 0), (1, 1), (2, 2)\}$
- $\tau(\{x1, x2, x3\}) = \{1, 2, 2\}$ est une solution
- $\tau(\{x1, x2, x3\}) = \{2, 2, 2\}$ est une autre solution

$\triangle$

1.4 Les COP

Les problèmes d'optimisation sous contraintes ou *Constraint Optimization Problems* (COP) sont une extension des CSP. Alors qu'on est intéressé, dans le cas des CSP, par n'importe quelle solution parmi potentiellement plusieurs, on cherche, en revanche, dans le contexte d'un COP, une seule solution en particulier. Une solution dont la qualité est la meilleure.

Afin d'évaluer la qualité d'une solution en particulier, on introduit au problème une fonction de coût f (aussi appelée fonction-objectif). Celle-ci donne un score réel à toute affectation valide [4].

Définition 1.4.1. Un COP est un quadruplet COP $= \langle X, D, C, f \rangle$ où :

- $\langle X, D, C \rangle$ définit un CSP ;
- f est une fonction telle que : $f : S \mapsto \mathbb{R} \mid \tau \in S =$ espace des solutions.

$\Diamond$

Le but est de trouver l'affectation τ qui minimise (resp. maximise) f :

Définition 1.4.2. La solution de COP $= \langle X, D, C, f \rangle$ est une affectation $\tau^* \in S$ telle que :
- τ^* est une solution de $\langle X, D, C \rangle$;
- argmin $f = \tau^*$ (resp. argmax $f = \tau^*$).

$\Diamond$

Exemple 1.4.1. Soit le problème vu à l'exemple 1.3.1 auquel on ajoute l'objectif suivant :

$$\min x_1 + x_2 + x_3$$

Dans ce cas, la seule solution possible est $\tau(\{x1, x2, x3\}) = \{1, 2, 2\}$. $\triangle$

1.5 Le partitionnement sous contraintes est un cas particulier de COP

Il devient évident à présent que le partitionnement sous contraintes n'est autre qu'un COP [2]. Si on définit les variables comme étant l'appartenance de chacune des observations à l'un des k groupes de Δ, alors le COP $= \langle X, D, C, f \rangle$ correspondant est tel que :
- $X = \{x_1, ..., x_n\}$ avec x_i groupe d'appartenance de l'observation o_i ;
- $D = \{1, ..., k\}$ et $D(x_i) = D \; \forall i \in [1, n]$;
- C, les contraintes agissant sur les observations à travers leurs représentants dans X ;
- $f = \gamma$, critère de partitionnement à optimiser.

Plusieurs techniques et cadres théoriques, tantôt exacts, tantôt approximatifs, existent pour résoudre des COP [5]. Dépendamment des demandes de l'utilisateur et de la taille des problèmes, l'une ou l'autre de ces techniques se trouve être favorisée, au détriment d'autres peut-être plus lentes ou trop imprécises. Il est question, dans ce travail, d'une méthode de résolution exacte adaptée au partitionnement de données et basée sur la programmation par contraintes.

1.6 La programmation par contraintes

La programmation par contraintes ou Constraint Programming (CP) est un paradigme de programmation utilisé pour la résolution de problèmes combinatoires, notamment les CSP et les COP. Il est déclaratif dans la mesure où l'utilisateur y traduit son problème comme une suite de contraintes individuelles qui mettent en relation les variables de celui-ci, sans se soucier de leur ordre d'exécution ni des détails autour du moteur de résolution en arrière-plan [6, 7].

La CP résout les problèmes à travers un arbre de recherche. À chaque nœud, une décision, qui correspond à l'instanciation ou à la réduction du domaine d'une variable en particulier, est prise par le moteur de résolution. Ces décisions sont telles que l'on espère que l'arbre de recherche soit le plus petit possible, en éliminant, par exemple, les choix infructueux au plus vite. On parle d'une heuristique de branchement. Cependant, à elle seule, celle-ci n'est pas suffisante pour venir à bout des problèmes plus compliqués [6, 8].

C'est là que la deuxième moitié de la CP intervient : la propagation de contraintes. En effet, à chaque décision de branchement, i.e. à chaque changement de domaine d'une variable, certaines contraintes sont ainsi dire « réveillées. » Elles agissent sur le domaine des variables dans leur portée en y enlevant les valeurs immédiatement devenues incohérentes à la suite des choix faits par l'heuristique de branchement. Une valse est alors initiée où l'action de chaque contrainte vient réveiller les autres. Ceci se fait jusqu'à l'atteinte d'un « point fixe » dans lequel plus aucun domaine ne peut être réduit davantage [6, 9].

Par ailleurs, en CP, on parle surtout de contraintes globales. Il s'agit de contraintes qui agissent sur toutes les variables de leur portée en même temps. Celles-ci sont munies de processus de filtrage des domaines plus rigoureux qui exploitent la structure du problème même pour se débarrasser du plus grand nombre de valeurs incohérentes [6, 9]. Différentes contraintes atteignent différents niveaux de *cohérence locale*. On cite, par exemple [6] :

- la *cohérence de domaines*, un état dans lequel chaque valeur de chaque variable dans la portée d'une contrainte admet un support ;
- la *cohérence de bornes*, un état dans lequel seules les valeurs aux extrémités des domaines de chaque variable dans la portée d'une contrainte admet un support.

La CP tire ainsi son efficacité du mariage intelligent entre propagation de contraintes globales et stratégies de recherche.

Exemple 1.6.1. Soit le CSP suivant :

- CSP $= \langle X, D, C \rangle$
- $X = \{x_i\}_{i \in [1,4]}$
- $D(x_1) = \{0, 1, 2, 3, 4\} \quad D(x_2) = D(x_3) = D(x_4) = \{1, 2, 3\}$
- $C = \{c_i\}_{i \in [1,4]} \mid c_1 = (x_2 > x_1),\ c_2 = (x_3 > x_1),\ c_3 = (x_4 > x_1),$
  $c_4 = \texttt{alldifferent}(x_2, x_3, x_4)$

Dont le réseau de contraintes est comme suit :

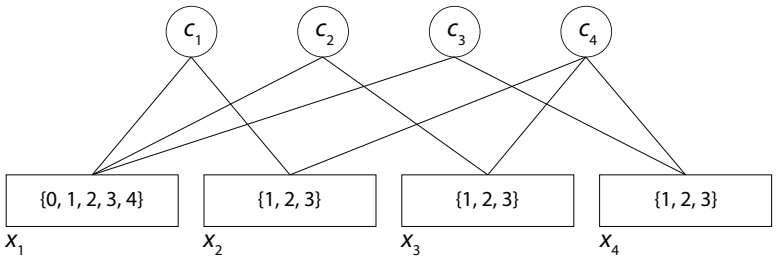

Figure 1.3 Réseau des contraintes initial

Une première propagation des contraintes résulte en ces nouveaux domaines :

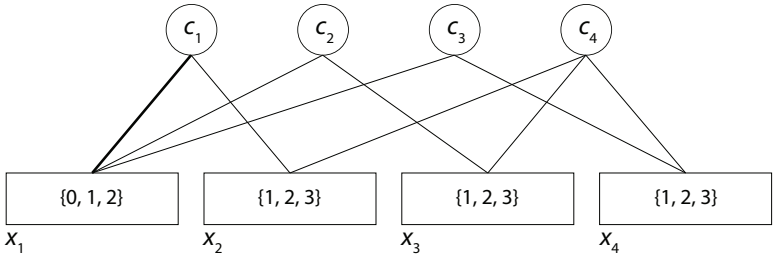

Figure 1.4 Domaines après propagation initiale

La contrainte c_1 enlève les valeurs 3 et 4 du domaine de x_1 car celles-ci n'ont pas de support dans le domaine de x_2. On se retrouve alors dans une situation de point fixe.

On est maintenant prêt à faire un premier choix dans la recherche de solution. Si $x_1 = 2$ est la décision prise, alors, après propagation de c_1, c_2 et c_3, on obtient les domaines suivants :

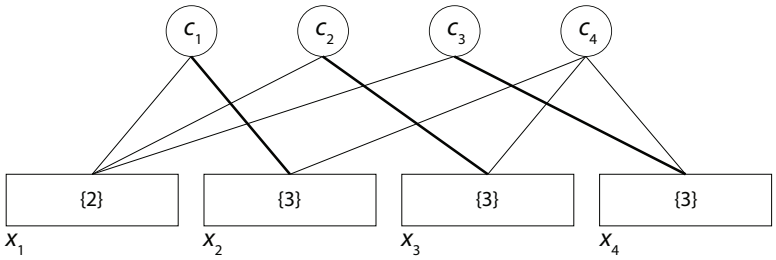

Figure 1.5 Domaines après propagation suivant le choix $x_1 = 2$

Arrivé à c_4, sa propagation va provoquer un échec. En effet, elle stipule que x_2, x_3 et x_4 doivent être différentes. Or, avec les décisions prises, elles ne peuvent prendre chacune que

la valeur 3. Cette branche de l'arbre est donc infructueuse et le moteur de résolution fait un retour en arrière pour essayer d'autres configurations à la recherche d'une solution valide :

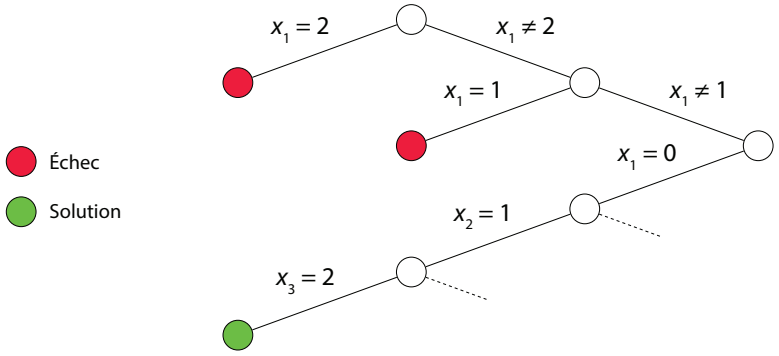

Figure 1.6 Arbre de recherche

Les choix $(x_1 = 0, x_2 = 1, x_3 = 2)$ mènent à une instanciation valide. La valeur de x_4 est assignée à la propagation de c_4 au dernier noeud. En effet, seule l'affectation $x_4 = 3$ est possible.

D'autres affectations valides existent dans d'autres branches de l'arbre. Dans le cas d'un COP, le moteur de résolution aurait continué avec *un branch and bound* au lieu de s'arrêter. △

Le but de ce travail de recherche est d'utiliser la CP pour résoudre de façon exacte des problèmes de partitionnement particuliers.

1.7 Problématique

La CP est une méthode nouvelle dans le domaine du partitionnement sous contraintes et il y a encore lieu à l'améliorer. Un cadre de résolution des problèmes de partitionnement par CP existe déjà mais celui-ci peine à résoudre les plus gros exemplaires [2, 5, 10]. De plus, celui-ci est encore à ses débuts et d'autres contraintes doivent être développées pour permettre la résolution de plus de problèmes encore. Ainsi, ce travail de recherche se présente comme un effort qui vient capitaliser sur ceux déjà entrepris par la communauté scientifique.

En l'occurrence, on s'intéresse, dans ce cas, aux deux composantes de la CP pour le partitionnement : stratégie de recherche et propagation de contraintes. D'abord, on explore des améliorations aux performances de résolution à travers un rehaussement des heuristiques de branchement déjà établies. Ensuite, on s'intéresse au cas du partitionnement sous contraintes de cardinalité. On tâche alors de proposer une nouvelle contrainte globale en vue de rendre

plus efficace la résolution de ce genre de problèmes. Il s'agit ici de problèmes nouveaux en CP auxquels on ne s'est jamais intéressé auparavant.

1.8 Objectifs de recherche

Minimiser la variance des partitions est un des critères les plus utilisés en traitement de données. C'est aussi souvent un dont les résultats sont les plus satisfaisants. En effet, les groupes obtenus contiennent des observations entre lesquels la variabilité est minimale (i.e. les groupes obtenus sont homogènes) tout en étant les plus séparés possibles [11]. Minimiser la variance revient à minimiser les distances (ou leurs carrés) entre les observations et les centroïdes des groupes obtenus. On appelle ces problèmes *Minimum Sum of Squares Clustering* ou MSSC.

C'est ce critère, en utilisant la distance euclidienne, qui fera l'objet de ce travail de recherche. Il s'agit d'un problème NP-Difficile en dimension générale [12]. Comme rapporté plus haut, ce n'est pas la première fois que la CP est appliquée pour résoudre ce problème. Un premier objectif de recherche est donc d'améliorer le travail déjà fait à travers le perfectionnement de l'heuristique de branchement déjà en place [5]. On propose alors deux améliorations clés aux stratégies existantes.

Un deuxième objectif de recherche est d'étendre ce cadre de résolution des problèmes de partitionnement par CP au cas où on voudrait que les groupes résultants obéissent à des contraintes de cardinalité connues. On retrouve ce cas de figure dans plusieurs champs d'application concrets qui exigent des groupes avec un nombre prédéterminé d'éléments chacun. On cite comme exemples le partitionnement distribué [13], la gestion de catégories en affaires [14], la composition de groupes de travail [15], le partitionnement de documents [14] et la segmentation d'images [16]. Ces contraintes de cardinalité peuvent aussi être utilisées pour renforcer le processus de partitionnement face à la présence de points de données aberrantes [17]. On parle alors de *Cardinality Constrained MSSC*, ou ccMSSC. L'idée est de rendre la résolution de ce cas particulier plus efficace en exploitant les caractéristiques du problème. Cela se matérialise sous la forme d'une nouvelle contrainte globale munie d'algorithmes de filtrage. On en propose deux versions : un algorithme simple adapté d'un autre déjà en place et un algorithme plus avancé faisant appel à des opérations plus poussées.

1.9 Plan

Ce document est divisé en six parties principales.

Tout d'abord, on commence par une mise en contexte pour familiariser le lecteur avec tous les aspects essentiels du problème afin de l'initier à la compréhension du travail de recherche ainsi qu'aux contributions souhaitées. Dans cette partie, il est question d'expliciter le problème du MSSC ainsi que de bâtir la fondation sur laquelle la résolution de ce problème aura lieu.

Deuxièmement, on enchaîne avec une revue de littérature dans laquelle est donné un résumé exhaustif d'outils pertinents. Cela est fait dans la perspective de rendre possibles de nouvelles contributions.

Par la suite, il est question de ramener ensemble les acquis de la revue de littérature pour bâtir la première contribution de ce travail de recherche. On parle ici des améliorations proposées quant à l'heuristique de branchement.

La partie adjacente est quant à elle dédiée à la deuxième contribution. Celle-ci est axée sur le partitionnement avec contraintes de cardinalité ainsi que sur les algorithmes de filtrage qui lui sont propres.

Ensuite, on consacre une partie au résumé, à l'illustration et à l'interprétation des principaux résultats issus de tests sur les algorithmes et sur les stratégies développées. On y fait usage, entre autres, d'exemples issus de la littérature afin d'avoir un point de référence commun.

En dernier lieu, on fait la synthèse des principaux résultats et contributions tout en laissant la porte ouverte à d'autres avenues de recherche en explicitant les limites ainsi que les opportunités potentielles.

CHAPITRE 2 CADRE PRÉLIMINAIRE ET MISE EN CONTEXTE

On essaye de familiariser le lecteur dans cette section avec les principaux éléments entourant le présent travail de recherche. En l'occurrence, on commence d'abord par présenter le problème du MSSC ainsi que ses caractéristiques notables. On enchaîne ensuite avec le modèle CP qui représente la fondation sur laquelle le présent effort est bâti. Enfin, on explique une classe de problèmes connue et utile dans le domaine de l'optimisation combinatoire.

2.1 Le partitionnement avec minimisation de variance

On a établi plus haut l'importance conséquente du partitionnement de données dans le traitement de celles-ci. Une façon de mener à bien cette tâche, avec des résultats satisfaisants, est de résoudre le problème du MSSC. Celui-ci revient à minimiser la variance des groupes d'observations après partitionnement.

Définition 2.1.1. Soit l'ensemble d'observations $O = \{o_i\}_{i\in[1,n]} \mid o_i \in \mathbb{R}^s$ que l'on veut partitionner en k groupes distincts ($k < n$).

L'objectif du MSSC est de trouver les centres $c_j, j = 1, ..., k$ tels qu'ils représentent une solution du problème d'optimisation sous contraintes suivant [18] :

$$\begin{aligned} \text{minimiser} \quad & \sum_{i=1}^{n}\sum_{j=1}^{k} \frac{1}{2} w_{ij}\|o_i - c_j\|^2 && \|\cdot\| = \text{norme euclidienne} \\ \text{tel que} \quad & \sum_{j=1}^{k} w_{ij} = 1 && \forall i \in [1,n] \\ & w_{ij} \in \{0,1\} && \forall i \in [1,n], \forall j \in [1,k] \end{aligned}$$

$\Diamond$

Dans la définition 2.1.1, la contrainte relie, à travers les coefficients w_{ij}, les points o_i à un centre c_i unique. En effet, une observation ne peut appartenir qu'à un seul groupe à la fois, ce dernier représenté par son centre.

La fonction Lagrangienne correspondante à ce problème est :

$$L(\boldsymbol{c}, \boldsymbol{\lambda}) = \sum_{i=1}^{n}\sum_{j=1}^{k} \frac{1}{2} w_{ij}\|o_i - c_j\|^2 + \lambda_1 \left(1 - \sum_{j=1}^{k} w_{1j}\right) + ... + \lambda_n \left(1 - \sum_{j=1}^{k} w_{nj}\right) \qquad \lambda_i \in \mathbb{R} \quad (2.1)$$

On peut ainsi calculer les dérivées suivantes :

$$\frac{\partial L}{\partial c_j} = -\sum_{i=1}^{n} w_{ij} \left[c_j \bullet (o_i - c_j)\right] \qquad \forall j \in [1, k] \tag{2.2}$$

$$\frac{\partial L}{\partial \lambda_i} = 1 - \sum_{j=1}^{k} w_{ij} \qquad \forall i \in [1, n] \tag{2.3}$$

Les conditions nécessaires de premier ordre de Karush-Kuhn-Tucker (KKT) exigent, entre autres, que :

$$\begin{aligned} \nabla L = \vec{0} \Rightarrow \frac{\partial L}{\partial c_j} = 0 \Leftrightarrow -\sum_{i=1}^{n} w_{ij} \left[c_j \bullet (o_i - c_j)\right] = 0 \qquad & \forall j \in [1, k] \\ \Leftrightarrow \sum_{i=1}^{n} (o_i - c_j) = 0 \qquad & \forall j \in [1, k] \end{aligned} \tag{2.4}$$

Ceci revient à [18] :

$$-nc_j + \sum_{i=1}^{n} o_i = 0 \Leftrightarrow c_j = \frac{\sum_{i=1}^{n} o_i}{n} \qquad \forall j \in [1, k] \tag{2.5}$$

Il devient clair, selon l'équation 2.5, que les centres c_j des groupes ne sont autres que leurs centroïdes. L'exemple 1.1.1 est donc une illustration du MSSC pour un cas $n = 4, s = 2, k = 2$.

Le lecteur remarquera que le problème est trivial pour le cas $k = 1$ où il suffit d'appliquer l'équation 2.5 à l'ensemble des observations pour trouver l'unique centre c_1 relatif à tous les points.

2.2 Problèmes de réseaux de flot : flot de coût minimal

Certains problèmes d'optimisation et/ou d'affectation ont une structure qui s'apparente à un réseau à travers lequel on essaye de faire passer un « fluide » avec un coût minimal (*Minimum-cost Flow*, MCF). Il serait alors judicieux de définir le cadre autour de cela, compte tenu du sujet du présent effort de recherche. En effet, traiter certains problèmes comme des problèmes de réseaux de flot présente plusieurs avantages vu que ceux-là sont bien étudiés et jouissent d'algorithmes extrêmement performants.

2.2.1 Définition

Un réseau de flot [19] se présente comme un graphe orienté $G = (V, A)$ dont :

- les $|V|$ sommets ne sont pas tous nécessairement directement connectés deux à deux ;
- les arcs dans A sont des paires (i, j) ordonnées avec $i, j \in V$.

Chaque arc $e \in A$ est muni des caractéristiques suivantes :

- x_e la quantité de fluide qui transite par e ;
- l_e (resp. u_e) la quantité minimale (resp. maximale) de fluide qui doit (resp. peut) transiter par e ;
- a_e le coût induit par le passage d'une unité de fluide à travers e ;

En contrepartie, chaque sommet $v \in V$ est muni d'une quantité s_v équivalente au volume de fluide qui en émane. Si cette quantité est positive, on parle d'une source. Si cette quantité est négative, on parle d'un puits. Enfin, si cette quantité est nulle, on parle d'un noeud-transit, où le fluide ne fait que passer d'un groupe d'arcs entrants à un autre d'arcs sortants de v [19].

On parle d'un flot *réalisable* si celui-ci respecte, à la fois, la loi de conservation de masse — au sens de la dynamique des fluides en régime permanent. i.e., la quantité de fluide perdue dans les puits doit nécessairement être égale à la quantité de fluide gagnée dans les sources — mais aussi les capacités maximales et minimales des arcs [19].

On parle d'un flot *optimal* dans le cas d'un MCF si celui-ci est non seulement réalisable mais emprunte, en outre, les arcs qui résultent en le coût le plus bas [19].

Exemple 2.2.1. Soit le réseau suivant :

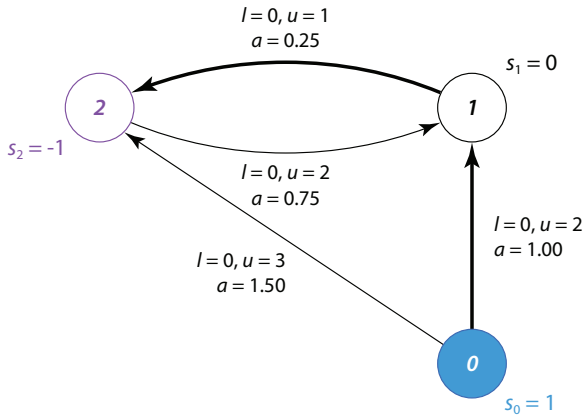

Figure 2.1 Exemple de réseau de flot

où :

— $V = \{0, 1, 2\}$;
— $A = \{(0, 1), (0, 2), (1, 2), (2, 1)\}$;
— $x_{(0,1)} = 0$, $x_{(0,2)} = 1$, $x_{(1,2)} = 0$ et $x_{(2,1)} = 0$ est un flot réalisable et son coût est 1.5 ;
— $x_{(0,1)} = 1$, $x_{(0,2)} = 0$, $x_{(1,2)} = 1$ et $x_{(2,1)} = 0$ est un flot optimal et son coût est 1.25.

△

2.2.2 Résolution

De façon plus formelle, il est possible de reformuler le problème de MCF à travers un réseau de cette façon :

$$\begin{aligned} \text{minimiser} \quad & \sum_{e \in A} a_e x_e & & \\ \text{tel que} \quad & \sum_{e \in A | e=(v,i)} x_e - \sum_{e \in A | e=(j,v)} x_e = s_v & & \forall v \in V \\ & l_e \leq x_e \leq u_e & & \forall e \in A \end{aligned}$$

où l'objectif représente le total des coûts induits et où les contraintes représentent la dynamique des fluides.

Il devient clair à présent qu'un problème de MCF n'est autre qu'un problème d'optimisation linéaire sous contraintes (*Linear Program*, LP). Celui-ci peut dont être résolu par n'importe quel algorithme de résolution de LP, en l'occurrence l'algorithme du Simplex et ses variantes.

Or, il existe une variante de l'algorithme du Simplex adaptée aux problèmes dont la structure s'apparente à celle d'un problème de MCF : Network Simplex. Ce dernier présente des performances entre 10 et 100 fois meilleures que celles des méthodes conventionnelles [19]. Network Simplex a une complexité temporelle de $\mathcal{O}\left(|V||A| \log |V| \cdot \log(|V| \cdot \max_{e \in A} a_e)\right)$ [20].

2.2.3 Solution entière aux LP

Si un LP sous contraintes $\mathbf{A}x = b$ admet une solution, elle est :

— soit unique et se trouve sur un des sommets du polyèdre formé par les contraintes ;
— soit une parmi une infinité dont les coûts sont les mêmes et qui se trouvent toutes sur une arête du polyèdre des contraintes (si celle-ci se trouve être orthogonale au gradient de l'objectif à la solution optimale).

Si, en plus :

- la matrice $\mathbf{A}$ est totalement unimodulaire (*Totally Unimodular matrix*, TU),
- le vecteur b a des composantes entières et
- le vecteur x ne contient que des variables non bornées ou bornées par des valeurs entières,

alors tous les sommets du polyèdre formé par les contraintes ont des coordonnées entières [21]. Cela veut dire aussi que si une solution x^* existe à ce LP, elle est forcément entière (ou peut l'être en se ramenant au sommet le plus proche).

Cette observation est extrêmement utile car elle permet de résoudre certains problèmes d'optimisation à variables entières (souvent des problèmes très durs) en passant par leur relaxation linéaire. On peut alors faire usage d'un des nombreux algorithmes pour LP, de complexité polynomiale, pour trouver des solutions entières à la relaxation. Celles-ci sont naturellement valides et globalement optimales pour le problème de départ non relaxé.

2.3 Description du modèle CP pour la résolution des problèmes de partitionnement avec minimisation de variance

Afin de définir le modèle CP pour la résolution du MSSC, il est nécessaire de définir les variables ainsi que les contraintes qui les régissent.

Variables représentatives Afin de représenter les observations O dans le modèle CP, on fait usage d'un tableau de variables `x`. Celui-ci est de taille $n = |O|$. Chaque variable de ce tableau lie une observation au groupe auquel elle doit appartenir. Par exemple, `x[i] = c` traduit le fait que l'observation o_i doive appartenir au groupe C_c. De ce fait, chaque variable dans le tableau `x` a un domaine $D(\texttt{x[i]}) = D = \{0, 1, ...k-1\}$.[1] Cette représentation suit la convention établie à la section 1.5. Ce sont également sur ces variables que le branchement se fera.

Variables connexes On appelle variables connexes des variables secondaires qui aident à la résolution du problème. Souvent, ce sont des variables qui servent de lien entre différents aspects du modèle. On définit dans cette catégorie le tableau de variables `card` qui représente les cardinalités des différents groupes C_c. Les domaines des variables de `card` sont alors tous $\{0, 1, ..., n\}$.

1. Pour les fins de ce modèle, tous indices confondus commencent désormais à 0, sauf mention contraire.

Objectif La fonction-objectif à la définition 2.1.1 est minimale avec un choix de centres optimal. Cependant, ces mêmes centres peuvent être déduits à partir de leur groupes respectifs. Ainsi, il est possible de résoudre le problème sans rendre ces centres explicites.

En utilisant la norme euclidienne, on peut réécrire, de façon équivalente [5], le problème à la définition 2.1.1 comme ceci :

$$\text{minimiser} \quad \sum_{c=0}^{k-1} \frac{1}{2} \frac{1}{|C_c|} \sum_{o,o' \in C_c} \|o - o'\|^2 \tag{2.6}$$

L'objectif consiste désormais en la recherche des $|C_c|$ éléments des k groupes C_c plutôt que leurs centres. Cette formulation facilite la mise en place d'une expression avec facteurs logiques dont on peut se servir dans le modèle CP :

$$\text{minimiser} \quad \sum_{\mathtt{c}=0}^{k-1} \frac{1}{\mathtt{card[c]}} \sum_{\mathtt{i}=0}^{n-1} \sum_{\mathtt{j}=\mathtt{i}}^{n-1} (\mathtt{x[i]} = \mathtt{c} \wedge \mathtt{x[j]} = \mathtt{c}) \cdot \|o_{\mathtt{i}} - o_{\mathtt{j}}\|^2 \tag{2.7}$$

On parle ici de contraintes réifiées (e.g. $\mathtt{x[i]} = \mathtt{c} \wedge \mathtt{x[j]} = \mathtt{c}$) dans la mesure où celles-ci sont transformées en objets logiques qui rendent compte de leur état (respectées ou non). Cela permet de les faire intervenir dans des expressions plus complexes (comme à l'expression 2.7) afin de faciliter l'expression et/ou la résolution du problème donné.

En fait, l'objectif 2.7 doit être représenté dans le modèle par une variable réelle à optimiser, `Z`. On opte alors à contraindre cette dernière de la manière suivante :

$$\mathtt{Z} = \sum_{\mathtt{c}=0}^{k-1} \frac{1}{\mathtt{card[c]}} \sum_{\mathtt{i}=0}^{n-1} \sum_{\mathtt{j}=\mathtt{i}}^{n-1} (\mathtt{x[i]} = \mathtt{c} \wedge \mathtt{x[j]} = \mathtt{c}) \cdot \|o_{\mathtt{i}} - o_{\mathtt{j}}\|^2 \tag{2.8}$$

Cette quantité s'appelle Within-Cluster Sum of Squares (WCSS). De cette façon, l'objectif devient alors :

$$\text{minimiser} \quad \mathtt{Z} \tag{2.9}$$

Contraintes : Cardinalité des groupes Afin que les variables dans le tableau `card` soient fonctionnelles, il est nécessaire de les contraindre à prendre les valeurs relatives à la cardinalité de chaque groupe.

Ceci se fait à l'aide des variables `x` en conjonction avec une Global Cardinality Constraint (GCC) [22] :

$$\mathtt{GCC(card,\ [0,\ 1,\ ...,\ } k-1\mathtt{],\ x)} \tag{2.10}$$

Définition 2.3.1. Global Cardinality Constraint ou GCC est une contrainte qui force le nombre d'occurrences des valeurs que prennent les variables de sa portée à appartenir à certains intervalles d'entiers (représentés par des domaines de variables, dans notre cas `card`). Celle-ci maintient un graphe biparti d'affectation avec, d'un côté, les variables et, de l'autre, les valeurs possibles. Chaque variable est connectée à une valeur si elle appartient à son domaine. GCC atteint une cohérence de domaine, à travers un problème de flot dérivé du graphe biparti, en complexité temporelle dans $\mathcal{O}(m\sqrt{n})$, avec m le nombre d'arcs dans le graphe d'affectation et n le nombre de sommets [23]. ◊

Contraintes : bris de symétrie À ce stade, le modèle est complet et sa résolution mènera à une solution du MSSC. Cependant, les lecteurs avertis remarqueront qu'il peut y avoir plusieurs solutions équivalentes à ce même problème. En effet, les indices que se voient attribuer les groupes obtenus peuvent être interchangés entre eux. Or, il importe peu quel indice a été attribué à quel groupe. Ce qui importe, au contraire, est l'égalité des indices de groupes entre plusieurs points, dénotant l'appartenance à un même groupe. On parle, dans ce cas, de symétrie de valeur.

En pratique, cela est un aspect indésirable en CP et en résolution de problèmes combinatoires en général. En effet, la présence de symétries rend la résolution moins efficace à travers un arbre de recherche plus grand. On perd du temps à redécouvrir des solutions analogues et, pire encore, on perd des ressources à explorer, à répétition, des sous-arbres similaires infructueux [24, 25].

Une solution immédiate, mais naïve, est de fixer k points à différents groupes. Mais cette solution requiert une connaissance a priori de la solution optimale. Par ailleurs, cette solution est aussi inefficace car elle n'exploite pas de façon approfondie la structure du problème afin de filtrer le plus grand nombre de valeurs possibles.

Théorème 2.3.1. Étant donné un ensemble de symétries de valeurs et une instanciation partielle des variables, élaguer toutes les valeurs symétriques de l'arbre de recherche est NP-difficile. [24] ◄

À première vue, le théorème 2.3.1 semble nous poser un problème. Cependant, il existe un cas particulier où se débarrasser de toutes les symétries de valeur peut être réalisé en temps polynomial. Ce cas particulier concerne les exemplaires où les valeurs peuvent être partitionnées en un nombre fini de groupes et où celles-ci sont interchangeables entre ces derniers [25]. Justement, le MSSC se trouve être une illustration de ce cas particulier.

Ainsi, on peut imposer une contrainte globale de priorité de valeur sur les variables dans le tableau `x` [25]. Plus formellement, cette contrainte fera en sorte que `x[0]` $= 0$ et que si `x[i]` $= c$, `i` $\in]0, n[, c \in]0, k[$, alors $\exists$`j` $<$ `i` $\mid$ `x[j]` $= c - 1$. [26]

La contrainte utilisée s'écrit alors de cette façon :

$$\texttt{intValPrecedence(x, } c-1\texttt{, } c\texttt{)} \qquad \forall c \in]0, k[\tag{2.11}$$

Cette contrainte maintient, pour une seule paire de valeurs, une cohérence de domaine en temps linéaire sur le nombres de variables concernées. Ainsi, la complexité totale est $\mathcal{O}(kn)$ pour l'ensemble des valeurs c. Il est théoriquement possible d'atteindre une meilleure propagation en considérant toutes les paires de valeurs possibles [26], ramenant la complexité à $\mathcal{O}(k^2 n)$. En pratique cependant, le gain lié à la meilleure propagation est presque inexistant [26].

De façon tout à fait analogue [25], si une telle contrainte de priorité de valeur est indisponible, le même résultat peut être atteint en imposant une contrainte qui force l'appartenance de la séquence des valeurs prises par `x` au langage rationnel définit par l'automate $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ ci-dessous [27] :

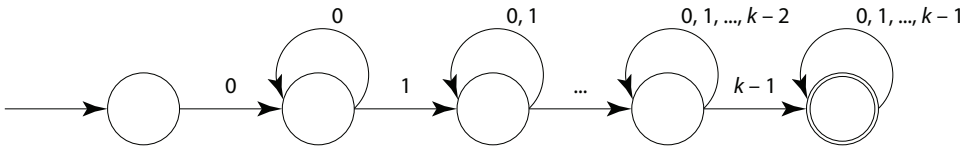

Figure 2.2 Automate fini déterministe pour la contrainte de bris de symétrie. (Le double cercle dénote l'état final.)

La contrainte s'écrit dans un modèle comme suit :

$$\texttt{regular(x, } M\texttt{)} \tag{2.12}$$

Cette contrainte maintient aussi une cohérence de domaine en temps $\mathcal{O}(n|\Sigma||Q|) = \mathcal{O}(nk^2)$ au pire cas, avec n le nombre de variables, Σ l'alphabet et Q l'ensemble d'états [27].

Maintenant que le contexte entourant ce travail de recherche a été établi, on s'intéresse, dans le chapitre suivant, aux travaux connexes qui pourraient venir l'appuyer.

CHAPITRE 3 REVUE DE LITTÉRATURE

Cette section représente un résumé de la littérature entourant le présent projet de recherche. Elle permet de définir l'environnement académique et technique dans lequel il s'inscrit. Celui-ci est très dense, ramenant ensemble le partitionnement de données et la CP, deux domaines en plein essor. Les éléments discutés ici peuvent servir de point de départ aux méthodes qu'on désire développer afin d'apporter des contributions fiables et utiles.

3.1 Quelques méthodes de résolution du MSSC sans contraintes

3.1.1 K-means

K-means est un des plus importants algorithmes en traitement de données [28]. C'est une heuristique simple mais efficace qui permet de résoudre le MSSC avec des résultats souvent de haute qualité.

Une exécution de K-means, dans sa version la plus simple [29], commence par une partition aléatoire non-optimale, via un placement arbitraire de k centres dans l'espace (les centres décident des groupes en assignant les éléments au centre le plus proche, voir section 2.1).

Chaque itération de K-means se divise en deux phases principales [29]. La première, à la figure 3.1, consiste à relocaliser les centres de la précédente partition non-optimale pour occuper les nouveaux centroïdes.

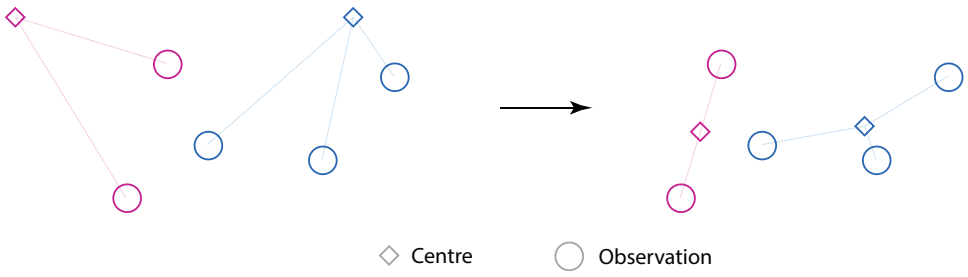

Figure 3.1 K-means : mise à jour des centres

La deuxième phase, à la figure 3.2, quant-à-elle, consiste à redéfinir la partition en liant les observations aux nouveaux centres les plus proches.

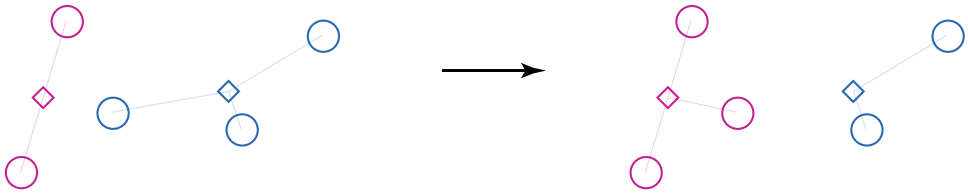

Figure 3.2 K-means : mise à jour des groupes

Cette activité de relocalisation et de redéfinition de la partition continue jusqu'à ce qu'un critère de convergence soit atteint. Sa complexité spatiale est $\mathcal{O}(n(s+k))$ alors que la complexité temporelle d'une itération est $\mathcal{O}(nk)$ [29]. Cependant, sa complexité temporelle totale au pire cas est super-polynomiale avec $2^{\Omega(\sqrt{n})}$ [30].

3.1.2 Repetitive Branch and Bound

Repetitive Branch and Bound Algorithm (RBBA) est une technique de résolution exacte du MSSC dont l'efficacité est compétitive avec d'autre méthodes [31].

Comme son nom l'indique, elle consiste en plusieurs recherches Branch and Bound en série. Plus spécifiquement, le problème est divisé en $n-k$ sous-problèmes de MSSC. Chacun de ces sous-problèmes $\pi_i, i \in [1, n-k]$ contient les $k+i$ dernières observations de O. On résout chacun de ceux-là, dans l'ordre, à l'aide d'une recherche Branch and Bound. Résoudre les problèmes qui ne contiennent que k points ou moins est inutile car leur solution est non seulement triviale mais est aussi sans importance. [31]

La séparation du problème principal en plusieurs sous-problèmes sert un aspect clé et est par conséquent la source de la force de RBBA. En effet, les auteurs ont trouvé un moyen de faire valoir la solution optimale aux problèmes $\pi_1, ..., \pi_{i-1}$ pour formuler des bornes plus serrées lors de la résolution du problème π_i. [31]

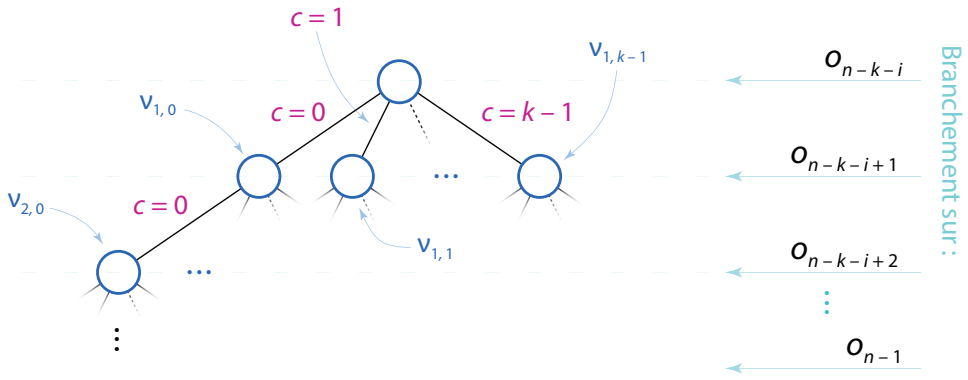

Figure 3.3 RBBA : résolution du sous problème π_i

En résumé, ces bornes sont telles que :

$$\mathrm{UB}(\pi_i) \coloneqq f(\Delta^*_{\pi_{i-1}}) \tag{3.1}$$

$$\mathrm{LB}(\pi_i, \nu_{m,c}) \coloneqq \mathrm{WCSS}(\Delta_{\pi_i|\nu_{m,c}}) + \mathrm{WCSS}(\Delta^*_{\pi_{i-m}}) \tag{3.2}$$

avec :

- f est une fonction connue qui, étant donné la solution au problème π_{i-1}, évalue la borne supérieure du WCSS au problème π_i ;
- Δ^*_π : partition optimale du problème π ;
- $\Delta_{\pi|\nu_{m,c}}$: partition courante au noeud $\nu_{m,c}$ du problème π, relative à l'assignation de m observations.

Lors de chaque recherche branch and bound, le branchement se fait sur les observations dans un ordre lexicographique, à chaque fois en plaçant une de celles-ci dans un groupe c différent. Si $\mathrm{LB}(\pi_i, \nu_{m,c}) > \mathrm{UB}(\pi_i)$, cette branche de l'arbre est interrompue à $\nu_{m,c}$ et un retour en arrière a lieu. [31]

Ainsi, avec ces bornes de meilleure qualité, chaque recherche subséquente est beaucoup plus efficace car les branches infructueuses sont identifiées et élaguées plus rapidement.

Les sous-problèmes sont alors résolus l'un à la suite de l'autre jusqu'au dernier. Celui-ci contient, par définition, les n observations et sa solution est donc celle du problème maître.

3.2 Quelques méthodes de résolution du MSSC sous contraintes

3.2.1 COP-Kmeans

Lors de la résolution du MSSC, une première manifestation des contraintes peut se faire sous la forme de contraintes d'appartenance. Par exemple, on peut imposer qu'une paire de points doive, ou non, appartenir au même groupe. Ces contraintes s'appellent Must-Link (ML) et Cannot-Link (CL), respectivement. Il s'agit ici d'un problème difficile où le simple fait de prouver sa faisabilité est en général un problème NP-Complet quand on a affaire à des contraintes CL [32].

Si on revient à K-means, tel que présenté à la section 3.1.1, on remarque que les partitions sont redéfinies en assignant chaque point au centre recalculé le plus proche (voir figure 3.2). Ces assignations se font de façon systématique et ne dépendent que de la distance entre chaque observation et les centres recalculés.

Cependant, supposons que nous ayons une liste de contraintes d'appartenance à faire respecter. Au lieu d'assigner les observations de façon systématique au centre le plus proche, on considère aussi le fait que cette assignation ne puisse avoir lieu que si les points du même groupe respectent les contraintes. Si une des contraintes est violée, le point se voit assigner au centre le plus proche suivant. Cette modification de K-means est au coeur de COP-Kmeans [33].

COP-Kmeans est alors une extension de K-means dans laquelle l'algorithme commence par créer une fermeture transitive sur toutes les contraintes présentées, à partir de laquelle l'ensemble complet des contraintes est dérivé et présenté à l'algorithme de K-means (voir section 3.1.1). L'assignation des points à sa deuxième phase (figure 3.2) se fait de façon gloutonne, en respectant ces contraintes. [33]

Cet aspect vorace d'affectation signifie aussi que COP-Kmeans ne garantit par toujours une solution. En effet, il se pourrait qu'une série de choix gloutons mène à une impasse [33]. e.g., soit $k = 2$ et O tel que nous venons d'effectuer les choix suivants : $o_0 = 0$, $o_1 = 0$, $o_2 = 1$ et $o_3 = 1$. Si le prochain point $o^\star$ est tel que $o^\star \neq o_1$ et $o^\star \neq o_2$ alors aucune affectation de $o^\star$ valide n'est possible. Le lecteur remarquera que le problème est tout de même consistant car la partition $o_0 = 0$, $o_1 = 1$, $o_2 = 1$, $o_3 = 0$ et $o^\star = 0$ est valide.

3.2.2 LIMA-VNS pour MSSC balancé

Outre les contraintes ML et CL, une seconde manifestation des contraintes peut se faire sous la forme de contraintes de forme. Par exemple, on peut imposer des contraintes de

cardinalité des groupes résultants. Une de ces contraintes les plus rencontrées est l'égalité des cardinalités, relative au problème du MSSC balancé (Balanced MSSC, bMSSC). Cela représente un problème NP-Difficile [34].

Ce problème se présente comme un candidat idéal pour une résolution à l'aide d'une recherche locale : étant donné une partition de groupes balancés, on pourrait explorer d'autres partitions voisines en mettant en oeuvre des échanges de paires de points entre groupes. On pourrait faire cela à la recherche de partitions de moindres coûts, jusqu'à ce qu'un critère d'arrêt soit atteint. A chaque itération, on aura $\mathcal{O}(n^2)$ échanges à considérer, comme montré à la figure 3.4. [35]

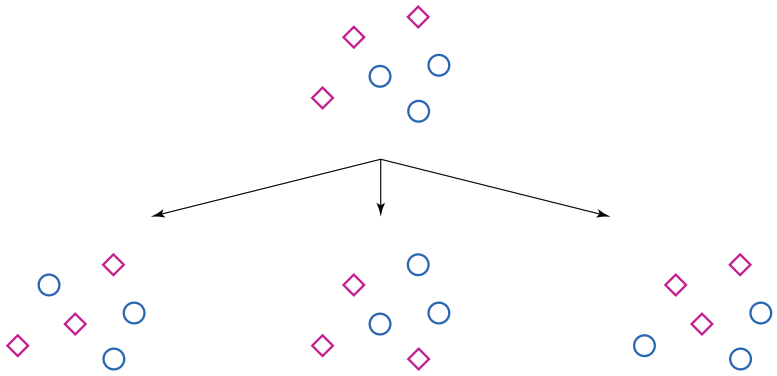

Figure 3.4 Illustration du voisinage de recherche locale pour MSSC balancé

On peut pousser cette idée plus loin et considérer d'autres voisinages, l'un à la suite de l'autre, si on ne trouve pas de meilleure solution dans le voisinage courant. Ce type de recherche locale s'appelle Variable Neighborhood Search (VNS).

L'idée fondamentale d'une VNS est d'effectuer une recherche locale en utilisant, de façon incrémentielle, différents voisinages de solutions. Soit q la meilleure solution trouvée jusqu'à maintenant, une itération typique de VNS commence par chercher une solution q' voisine à q dans le voisinage $\mathcal{N}_i$. Ensuite, elle met en place une descente locale à partir de q' pour l'améliorer en q''. Si q'' a un meilleur coût que q, une nouvelle itération est entamée avec $q \leftarrow q''$ dans le voisinage de départ $\mathcal{N}_0$. Sinon, une nouvelle itération est initiée avec la même solution q, sauf que maintenant la recherche locale se poursuit dans le voisinage $\mathcal{N}_{i+1}$. [35]

Appliquée au problème de MSSC balancé, une VNS simple mais efficace (adhérant à la Less is More Approach, LIMA) consiste en la sélection des voisinages $\mathcal{N}_i$ comme étant l'échange aléatoire de i paires d'observations, simultanément. La descente, quant-à-elle, représente l'échange systématique d'une paire de points à la fois, à la recherche d'une meilleure solution. [35]

Cette méta-heuristique est davantage accélérée en identifiant des techniques de recalcul de coût à temps constant, éliminant ainsi la nécessité de refaire des calculs fastidieux à répétition. Au final, cet algorithme est un des plus performants et des plus simples pour la résolution heuristique du problème du bMSSC. [35]

3.2.3 Optimisation conique pour MSSC avec contraintes de cardinalité

Il est question ici du problème principal discuté dans ce travail de recherche où on désire résoudre un MSSC dans lequel les cardinalités sont décidées à l'avance (ccMSSC, voir exemples à la section 1.8). Le bMSSC discuté ci-dessus est un cas particulier du ccMSSC.

Les auteurs de [17] proposent de faire valoir le paradigme d'optimisation convexe pour résoudre des relaxations linéaires du ccMSSC. Ces méthodes permettent de retrouver des solutions globalement optimales pour les versions relaxées du problème en temps polynomial.

Par la suite, les auteurs proposent plusieurs heuristiques qui permettent d'arrondir la solution au problème relaxé pour retrouver des solutions valides au problème maître. Dans leur méthode, les algorithmes permettent aussi d'émettre des garanties quant à la solution obtenue à travers le calcul de bornes inférieures et supérieures [17].

Pour quelques-uns des exemplaires utilisés, ces bornes s'avèrent être égales [17]. Pour ces cas, il est alors possible de prouver a posteriori l'optimalité globale des solutions à ces exemplaires.

Ainsi, bien que cette méthode ne soit pas, à strictement parler, exacte, elle semble être la meilleure pour résoudre certains ccMSSC jusqu'à optimalité globale (dans le cas où les bornes obtenues de part et d'autre sont égales). Aucune autre méthode de résolution exacte n'existe [17] autre que d'utiliser des cadres de résolution déclaratifs généraux comme la CP ou autres [36].

3.3 Résolution du MSSC à travers la CP

Il existe deux méthodes de résolution du MSSC en CP, issues des mêmes auteurs [5, 10].

3.3.1 CP Clustering

On peut utiliser la CP pour résoudre le modèle défini à la section 2.3, ce qui reviendrait, en effet, à résoudre le MSSC. Cependant, en pratique, ce modèle n'est utilisable que pour de très petites instances du MSSC. En effet, on n'y fait pas valoir les outils propres à la CP. En particulier, il ne comporte aucune contrainte globale relative à la minimisation du WCSS et

ne définit aucune heuristique de branchement afin de limiter la taille de l'arbre de recherche et le nombre de retours en arrière.

Dans une perspective de remédier à cela, une contrainte globale a été proposée pour filtrer à la fois les variables dans x ainsi que la variable Z [5]. Étant donné une instanciation partielle de x et $\mathsf{Z} \in [\mathrm{LB}(\mathsf{Z}), \mathrm{UB}(\mathsf{Z})]$, cette contrainte commence par calculer une borne inférieure du WCSS, laquelle vient resserrer le domaine de Z. Ensuite, dans la même opération de propagation, elle calcule, pour chaque instanciation possible des variables dans x, une borne inférieure du WCSS. Si celle-ci dépasse $\mathrm{UB}(\mathsf{Z})$, la valeur est éliminée du domaine de la variable concernée. Cet algorithme s'appelle CPC pour *CP Clustering* [5].

Bornes inférieures locales L'algorithme calcule, pour chaque C_c, une borne inférieure $\underline{Z}(C_c, m)$ de sa contribution au WCSS du problème, si on venait à lui assigner m des q points libres (i.e., $m \in [0, q]$). Cette quantité représente alors le meilleur coût possible induit par C_c pour chaque valeur de m. Cette borne est calculée en considérant les meilleures distances entre les points libres et les points fixes ainsi que les points libres entre eux. [5]

Borne inférieure globale Cette étape calcule une borne inférieure du WCSS du problème, laquelle est utilisée pour resserrer le domaine de Z. Il est facile de remarquer que cette borne inférieure représente la meilleure répartition des q points libres sur les k groupes C_c. Elle peut alors être égale à la somme de leurs plus basses contributions, calculée ci-dessous :

$$LB(\mathsf{Z}) = \underline{Z}(\{C_0, ..., C_{k-1}\}, q) = \min_{m_0+...+m_{k-1}=q} [\underline{Z}(C_0, m_0) + ... + \underline{Z}(C_{k-1}, m_{k-1})] \tag{3.3}$$

Connaître les différents m_c explicitement est inutile. Il suffit, à chaque fois, de diviser les points libres entre un groupe en particulier, d'une part, et le reste des groupes, de l'autre. On peut alors définir la relation de récurrence suivante :

$$\underline{Z}(\{C_0, ..., C_c\}, m) = \min_{i \in [0,m]} [\underline{Z}(\{C_0, ..., C_{c-1}\}, i) + \underline{Z}(C_c, m - i)] \tag{3.4}$$

Ainsi, il est possible de retrouver la réponse à 3.3 par programmation dynamique (*Dynamic Programming*, DP) en utilisant la relation 3.4. [5]

Filtrage L'opération de filtrage vient évaluer une borne inférieure du WCSS en supposant une affectation de points libres à un groupe donné.

On considère $\mathcal{C}$ l'ensemble des groupes partiellement remplis dont $C_c^\star \in \mathcal{C}$ est muni du point libre $o^\star$. Ainsi, analogue aux relations 3.3 et 3.4, on calcule la borne inférieure relative à cette nouvelle configuration comme ceci [5] :

$$\underline{Z}(\mathcal{C}, q-1) = \min_{i \in [0,q-1]} [\underline{Z}(\mathcal{C} \setminus C_c^\star, i) + \underline{Z}(C_c^\star, q-1-i)] \tag{3.5}$$

avec :

$$\underline{Z}(\mathcal{C} \setminus C_c^\star, i) = \max_{j \in [0,q-i]} [\underline{Z}(\{C_0, ..., C_{k-1}\}, i+j) - \underline{Z}(C_c, j)] \tag{3.6}$$

Si la quantité 3.5 dépasse la meilleure valeur de $\mathtt{Z}$ trouvée (i.e., UB($\mathtt{Z}$)), alors la valeur c est filtrée du domaine de la variable dans $\mathtt{x}$ relative à $o^\star$. On ne raisonne alors plus sur la faisabilité, comme expliqué dans le chapitre 1, mais sur l'optimalité (on ne garde que les valeurs qui peuvent mener à de meilleures solutions). On parle d'un filtrage basé sur le coût (Cost-based Filtering) [37]. Les termes intervenant dans 3.5 sont recalculés à partir du tableau de DP utilisé pour la relation 3.4 [5].

Une propagation de cette contrainte se fait avec une complexité spatiale de $\mathcal{O}(n^2)$ et une complexité temporelle de $\mathcal{O}(kq^2 \log q + qn)$. [5]

Heuristique de recherche Une contrainte globale, on le rappelle, n'est qu'une partie de la résolution du problème. L'autre étant la stratégie de recherche. Ainsi, les auteurs de la contrainte globale discutée plus haut proposent de l'accompagner d'une heuristique de branchement simple pour faciliter la résolution du problème. Celle-ci se caractérise par deux phases (plus de détails à la section 4.2) :

1. une première phase dans laquelle on génère la solution initiale de façon gloutonne : les variables se voient assigner des valeurs l'une à la suite de l'autre en privilégiant l'affectation qui augmente l'objectif le moins à chaque fois ;
2. une deuxième phase dans laquelle on commence par calculer, pour chaque variable, la plus petite augmentation de l'objectif en lui attribuant une valeur de son domaine. Ensuite, on sélectionne, parmi ces couples variable-valeur, le couple qui maximise cette augmentation de l'objectif.

La solution initiale générée par l'étape 1 ci-dessus définit la borne UB($\mathtt{Z}$) initiale, contre laquelle les premiers filtrages vont être faits. La recherche se poursuit après comme un branch and bound binaire qui suit la stratégie à l'étape 2 [5].

3.3.2 CP Repetitive Branch and Bound

Cette deuxième méthode de résolution du MSSC en CP (CP RBBA) incorpore les caractéristiques de RBBA afin d'atteindre des performances largement meilleures à CPC [10].

Le principe de fonctionnement est le même que celui de RBBA vu à la section 3.1.2 avec une différence majeure : les $n - k$ sous-problèmes ne sont plus résolus comme un simple Branch and Bound. Ils sont, au contraire, traités comme des problèmes CP à part entière [10].

Le problème est divisé en $n - k$ modèles COP, chacun relatif aux dernières $k + i$ observations, i étant le numéro du sous-problème (voir section 3.1.2). Étant des COP, ces modèles incorporent les contraintes du problème maître. La stratégie de branchements, utilisée dans la résolution de ces sous-problèmes, suit les recommandations de RBBA [10]. Ces sous-problèmes sont alors résolus l'un à la suite de l'autre, avec un moteur CP, jusqu'au dernier.

Au final, il est possible d'atteindre un avantage conséquent grâce à l'usage de bornes de meilleure qualité par rapport à CPC [10].

3.4 Impact de l'ordre et des initialisations sur les performances de certaines méthodes de résolution

L'ordre de considération des points dans O a souvent un impact conséquent sur les performances des méthodes de résolution, tant au niveau de la qualité qu'au niveau de la rapidité. Il est alors utile d'étudier leur impact sur certaines méthodes usuelles.

3.4.1 RBBA

En pensant à RBBA à la section 3.1.2, il est facile de voir pourquoi l'ordre des points aurait un impact sur les performances. En effet, chaque sous-problème π_i ne considère que les $k + i$ dernier points de O. Ceux-ci sont naturellement décidés par l'ordre des observations dans O.

Or, il a été observé que les performances variaient de façon significative selon la séquence des points ajoutés aux sous-problèmes au fur et à mesure de la résolution. Ceci est à cause du fait que certains arrangements font en sorte que les points qui restent à la fin de la recherche sont trop perturbateurs et compliquent la résolution. On les qualifie de perturbateurs à cause du fait que la solution aux nouveaux sous-problèmes est significativement différente des sous-problèmes antérieurs (on rappelle que ces derniers sont utilisés pour aider la résolution des sous-problèmes qui leur succèdent). Ceci est davantage exacerbé par le fait que les derniers sous-problèmes sont relativement très larges (proches de la taille initiale du problème). [31]

Un ordonnancement efficace des observations identifié par les auteurs de RBBA est de placer la suite des paires de points les plus proches de part et d'autre de la liste. i.e., on place la paire de points les plus proches aux positions 0 et $n-1$ de la liste, la seconde paire de points les plus proches aux positions 1 et $n-2$ et ainsi de suite. Cela a pour effet d'assurer que les derniers points à ajouter aux sous problèmes (à partir de $\pi_i \mid k+i > n/2$) viennent rejoindre le point avec lequel ils étaient jumelés avant la séparation (un point qui leur était le plus proche). Cet ordre des observations s'appelle *Nearest Neighbor* [31].

Par ailleurs, les auteurs de CP RBBA suggèrent un deuxième ordonnancement pour RBBA, appelé *Farthest First*, qui peut produire parfois de meilleurs résultats. L'idée derrière *Farthest First* est d'ordonner les observations toujours en sélectionnant le i-ème point comme le plus éloigné des premiers $i-1$ points sélectionnés.

Alternativement, si une bonne solution est connue a priori, dans le contexte de RBBA appliqué à un problème de *Clusterwise Regression*, il a été identifié que : [38]

- Un ordre séquentiel des observations, selon leur appartenance aux groupes, nuit aux performances ;
- Un ordre qui alterne les observations selon leur groupe d'appartenance donne lieu à de bonnes performances ;
- Une meilleure stratégie que les deux dernières consiste à alterner les points tout en les ordonnant par ordre décroissant d'erreurs aux hyperplans générés de façon heuristique.

3.4.2 K-means++

Au même titre que les performances de RBBA sont sensibles à l'ordre des observations, K-means est sensible à l'initialisation des centres à partir desquels la recherche commence. En effet, cela a un impact à la fois sur la vitesse de convergence mais aussi sur la qualité de la solution. Ainsi, certaines initialisations peuvent mener à des minima locaux de meilleure qualité que d'autres. En outre, on rappelle que le pire cas de K-means est super-polynomial (voir section 3.1.1).

Intuitivement, on penserait que des centres initiaux les plus éparpillés parmi les observations devraient produire de meilleurs résultats. Ceci se trouve être exactement l'idée derrière K-means++ qui choisit des centres initiaux pour K-means qui sont de préférence les plus espacés entre-eux. Plus formellement, cela est atteint en suivant les étapes suivantes : [39]

1. Sélectionner une observation au hasard comme premier centre c_0 ;

2. Choisir une observation o_i comme nouveau centre c_i avec une probabilité égale à :

$$P(o_i) = \frac{d^2_{\min}(o_i)}{\sum_{o_i \in O} d^2_{\min}(o_i)} \tag{3.7}$$

   $d^2_{\min}(o_i)$ étant la distance euclidienne carrée minimale entre o_i et les centres $c_0, ..., c_{i-1}$

3. Répéter 2 jusqu'à l'obtention de k centres.

D'après les étapes décrites ci-dessus, il est clair que les centres sont choisis l'un à la suite de l'autre en préférant toujours un nouveau centre qui se trouve plus loin (équation 3.7). Les résultats empiriques d'une telle initialisation viennent confirmer l'importance de choisir des centres éparpillés pour à la fois accélérer K-means mais aussi pour obtenir de meilleures solutions. [39]

3.5 GCC avec coûts

On rappelle au lecteur la GCC, contrainte globale discutée à la section 2.3. Celle-ci permet de maintenir le nombre d'occurrences de valeurs dans les domaines des variables de sa portée. Or il existe [40] une version de cette contrainte qui prend en compte des coûts d'affectation dans le maintien des cardinalités : à chaque couple variable-valeur est associé un coût et le but est de minimiser la somme de ceux-ci (typiquement, minimiser les coûts relatifs aux affectations choisies).

Cette contrainte se présente comme un excellent outil à introduire à un modèle de CP dans lequel on essaye de résoudre un problème qui fait intervenir à la fois des cardinalités à respecter ainsi que des coûts à minimiser. On cite le problème de distribution d'horaires de travail comme exemple : chaque horaire doit être comblé par un nombre de personnes donné mais chaque travailleur exprime une certaine préférence pour chacun des horaires. Dans un effort de satisfaire un nombre maximal de travailleurs, on essaye de minimiser les coûts en même temps qu'on fasse respecter les cardinalités [40].

3.5.1 Principe de fonctionnement

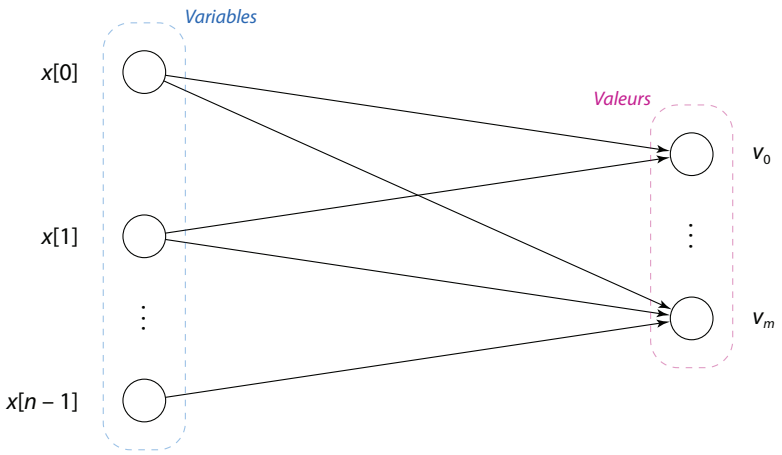

Figure 3.5 Graphe d'affectation d'un problème donné

La GCC avec coûts démarre d'un graphe d'affectation $G = (V, A)$ comme celui à la figure 3.5. Chaque arc $e \in A$ dans ce graphe est muni d'un coût a_e égal à l'affectation variable-valeur qu'il connecte. Le but est de trouver le sous-ensemble d'arcs qui connectent chaque variable avec une valeur de telle sorte que :

1. chaque variable soit affectée à une valeur unique ;
2. le nombre de variables égales à une certaine valeur appartienne à un intervalle bien défini (i.e., cardinalités maximale et minimale) ;
3. la somme des poids de ces arcs soit minimale (ou maximale selon le problème).

En utilisant les outils relatifs au MCF discutés à la section 2.2, retrouver ce sous-ensemble d'arcs est simple. Il suffit de dériver un réseau de flot à partir du graphe d'affectation :

— en alimentant, à partir d'une source commune, tous les sommets relatifs aux variables avec une seule unité de fluide chacun et
— en récoltant le flot dans un puits connecté aux sommets relatifs aux valeurs.

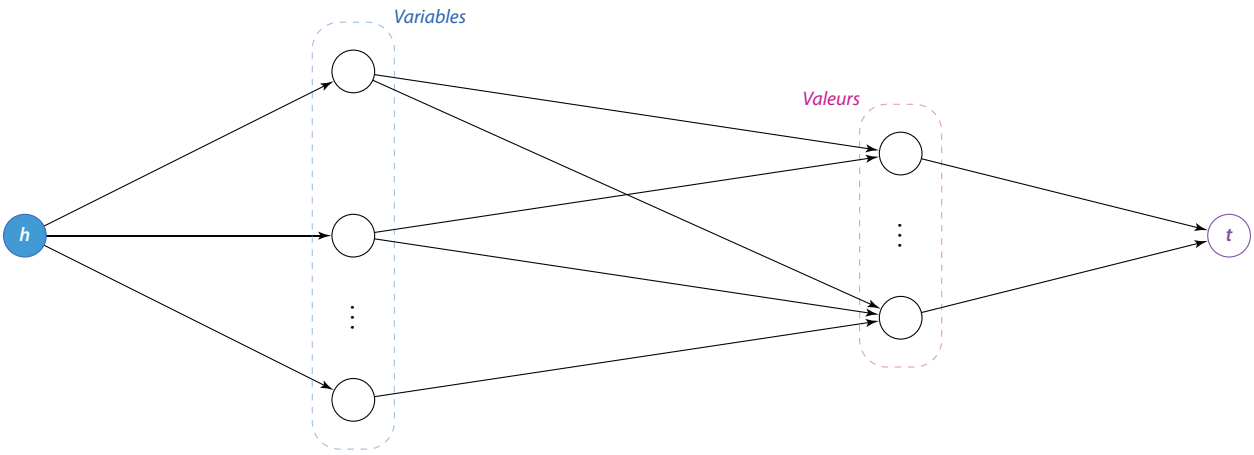

Figure 3.6 Réseau dérivé du graphe d'affectation d'une GCC avec coûts

Tous les arcs supplémentaires ont un coût nul (seuls les arcs existants relatifs aux affectations sont munis de coûts). Tous les arcs ont aussi une exigence l_e et une capacité u_e égales à 0 et 1 respectivement sauf les arcs qui relient au puits les sommets relatifs aux valeurs : ceux là on une exigence et capacité égales aux bornes respectives des cardinalités correspondantes voulues.

Les arcs choisis par le MCF sur ce réseau dérivé correspondent à l'affectation de poids optimal qui respecte les cardinalités voulues.

3.5.2 Résolution et filtrage

Contrairement à la section 2.2, GCC avec coûts ne résout pas le MCF comme un LP. A la place, la contrainte emploie un algorithme de chemins de poids minimaux successifs pour trouver un circuit de flot optimal dans le réseau. La résolution du problème suit une méthode analogue à celle décrite à la section 3.3.1. On commence par filtrer le domaine de l'objectif avec la valeur du MCF trouvé. Ensuite, on procède à un filtrage basé sur le coût des variables entières du problème.

Ce filtrage basé sur le coût suppose des affectations valides en particulier et réévalue le MCF du problème augmenté à chaque fois. Si cette valeur dépasse la borne supérieure de l'objectif, elle est filtrée du domaine de la variable concernée.

3.5.3 Complexité et niveau de cohérence

L'algorithme des chemins de poids minimaux successifs a une complexité de $\mathcal{O}(n(m+n\log n))$ avec : n, nombre de sommets, et m, nombre d'arcs.

Cependant, à l'aide d'un graphe résiduel de flot, il est possible de rendre tous les calculs d'une GCC avec coûts incrémentiels (et éviter des calculs répétitifs inutiles). On observe les changements à chaque propagation et on met à jour le MCF au fur et à mesure à l'aide du même algorithme des chemins de poids minimaux successifs. La complexité temporelle est ramenée à $\mathcal{O}(k(m + n\log n))$ avec k, nombre de variables actives (variables qui ont subi un changement de domaine). Le filtrage basé sur le coût est lui aussi dérivé d'une mise à jour du MCF avec une complexité de $\mathcal{O}(\min\{n, \#\text{valeurs}\} \cdot (m + n\log n))$ en utilisant le même algorithme des chemins de poids minimaux.

CHAPITRE 4 AMÉLIORATION DE L'HEURISTIQUE DE RECHERCHE

On présente dans ce chapitre la première contribution souhaitée. Celle-ci est en rapport avec le premier des deux axes de la CP : la stratégie de recherche. On démarre à partir d'une stratégie pour la résolution du MSSC par CP connue dans la littérature. Par la suite, on y identifie des faiblesses, lesquelles on essaye de corriger à l'aide de nouvelles procédures.

4.1 Initialisation de la recherche

4.1.1 Retour sur l'initialisation gloutonne de CP Clustering

Les filtrages en relation avec la résolution du MSSC par CPC reposent sur le calcul de bornes et sur des comparaisons entre celles-ci (voir section 3.3.1 et chapitre 5). Comme vu au paragraphe *Heuristique de recherche* à la section 3.3.1, il est nécessaire d'avoir une solution initiale contre laquelle les premiers filtrages auront lieu.

Afin d'avoir les meilleures performances possibles, cette solution initiale doit :

1. avoir le coût le plus bas possible ;
2. être facile à générer.

Comme proposé dans [5], une façon de générer une telle solution consisterait à brancher immédiatement sur la première paire variable-valeur qui fait augmenter `Z` le moins.

4.1.2 Limitations de l'initialisation gloutonne

Si on se fie à la section précédente, une conclusion, de prime abord raisonnable, serait que l'ordre des points n'importe aucunement à la résolution du problème. Après tout, la stratégie gloutonne avancée plus haut est déterministe et branchera sur un choix bien défini à chaque fois. Cependant, on attire l'attention du lecteur à l'implication suivante :

$$|C_c| = 1 \Rightarrow \text{WCSS}(C_c) = 0 \tag{4.1}$$

Cette observation, visiblement triviale, implique que l'assignation d'une observation à un groupe vide ne fait pas augmenter `Z`. Cela veut dire que, tant qu'il y a des groupes vides, l'heuristique à la section 4.1.1 branche sur la première observation non assignée en la mettant dans le premier groupe vide. En d'autres termes, les k premiers branchements assignent toujours, dans l'ordre, $o_0, ..., o_{k-1}$ à $C_0, ..., C_{k-1}$. Ceci a alors une implication directe sur le

reste des branchements à faire et donc sur la solution initiale générée. Plus particulièrement encore, cela a un impact sur le coût de cette dernière (coût qui est utilisé pour les filtrages initiaux). On voit alors, à l'instar de la section 3.4, que l'ordre et l'initialisation pourraient avoir un impact sur les performances.

De plus, pour une même solution initiale, sa présentation dans l'arbre de recherche a un impact significatif sur la taille subséquente de celui-ci. Ceci est dû au fait que cela influence l'agencement des choix à faire et que certains de ces choix permettent d'éliminer des sous-arbres infructueux beaucoup plus vite (i.e., plus proche de la racine). En effet, pour une même solution initiale, il existe $n!$ différents branchements possibles. Un branchement vorace ne nous donne aucun contrôle sur la façon d'introduire cette solution initiale à la recherche. La section 4.1.4 donne plus de détails sur cet aspect en particulier.

Enfin, le branchement vorace proposé à la section 4.1.1 est particulièrement inadéquat à la résolution d'un ccMSSC et donne lieu à des solutions initiales de mauvaise qualité. Ceci est normal, il n'y a aucune prise en compte de l'aspect des cardinalités restreintes dans le processus de génération de solution.

4.1.3 Génération d'une solution initiale

Supposons que l'on connaisse a priori une bonne solution au problème, τ_0 (qui peut être ou non égale à la solution optimale τ^*). Une façon d'assurer de bonnes performances est de s'arranger pour faire les branchements initiaux non pas selon la section 4.1 mais de telle sorte qu'ils suivent τ_0.

Dans l'esprit d'une initialisation heuristique rapide et de bonne qualité, au lieu d'un branchement vorace, on propose de faire valoir une des multiples heuristiques existantes (voir quelques-unes discutées au chapitre 3) pour la résolution du MSSC sous contraintes. Ces heuristiques doivent générer une solution réalisable et qui respecte toutes les contraintes introduites au problème.

Un démarrage à partir d'une solution inconsistante (qui ne respecte pas les contraintes) pourrait marcher aussi. Cependant, elle aura un impact préjudiciable sur les performances dans la mesure où le moteur CP devra dévier des choix dictés par sa stratégie de recherche et en faire d'autres de moins bonne qualité.

4.1.4 Introduction de la solution initiale à la recherche

Comme vu à la section 4.1.2, obtenir une solution initiale n'est qu'une partie du problème. L'autre étant son introduction à la recherche.

Exemple 4.1.1. Pour cette même solution initiale ($n = 4, k = 2$) :

$$\texttt{x[0]} = 0, \texttt{x[1]} = 1, \texttt{x[2]} = 1, \texttt{x[3]} = 1$$

Il existe plusieurs façons ($4! = 24$) de la présenter dans l'arbre de recherche, dont :

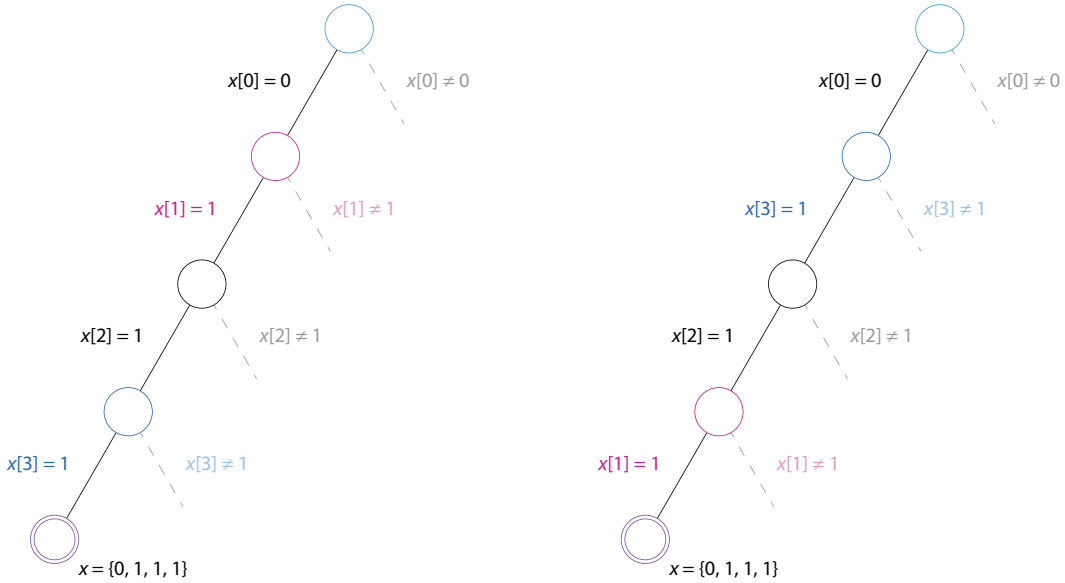

Figure 4.1 Branches d'initialisation de recherche

△

A l'exemple 4.1.1, on constate que pendant que les deux arbres mènent à la même solution initiale $\{0, 1, 1, 1\}$, ceux-là sont différents. Par exemple, la branche $\texttt{x[1]} \neq 1$ n'est pas au même endroit. Si celle-ci est infructueuse, elle éliminerait un plus gros sous-arbre dans le cas de gauche que dans le cas de droite dans lequel cette branche infructueuse sera revisitée plusieurs fois. On élimine un plus gros sous-arbre dans le cas de gauche car on se trouve près du sommet de l'arbre, où la majorité des variables n'est pas encore instanciée.

4.1.5 Nouvelles méthodes retenues pour l'introduction de la solution initiale

Les méthodes proposées plus bas s'inspirent de celles présentées dans [38] pour le cas du problème de *Clusterwise Regression* (voir section 3.4.1). On les adapte ici au cas du MSSC. L'idée générale derrière celles-ci est que l'agencement des branchements relatifs à la solution initiale feront en sorte que les branches qui ont une plus grande probabilité d'être mauvaises soient ramenées vers la racine de l'arbre (voir section 4.1.4). On parle d'un principe *fail-first*,

très utilisé en CP [6]. On espère aussi, à travers ces méthodes, ramener les affectations plus problématiques à la racine de l'arbre, où on a le plus de flexibilité.

Distances décroissantes au centre Cette méthode assigne les points à leur groupes respectifs (tels que dicté par la solution heuristique initiale) par ordre décroissant de distances par rapport aux centres de ces derniers. La première observation à être affectée sera alors celle qui est la plus éloignée du centre de son groupe alors que la dernière est la plus rapprochée du centre du sien.

Distances minimales décroissantes aux autres centres Cette méthode est analogue à la dernière si ce n'est qu'on n'utilise plus la distance avec le centre du groupe de l'observation mais plutôt la distance minimale entre l'observation et les centres des groupes qui ne sont pas munis de ladite observation.

4.2 Heuristique de recherche subséquente

Une fois la solution initiale introduite et la première borne supérieure calculée, une autre stratégie de branchement prend la relève afin de continuer le branch and bound. Les auteurs dans [5] (voir section 3.3.1, paragraphe *Heuristique de recherche*) proposent de sélectionner la variable `x[i]` avec la valeur $c \in D(\texttt{x[i]})$ tels que :

$$\texttt{i} = \underset{\texttt{i} \in [0,n[\ |\ |D(\texttt{x[i]})|>1}{\operatorname{argmax}} \Delta\texttt{Z}|_{\texttt{x[i]}=c} \quad \text{avec} \quad c = \underset{c \in D(\texttt{x[i]})}{\operatorname{argmin}} \Delta\texttt{Z}|_{\texttt{x[i]}=c} \, \forall \texttt{i} \in [0, n[\tag{4.2}$$

ceci est en sachant que :

$$\Delta\texttt{Z}|_{\texttt{x[i]}=c} = \frac{\mathrm{WCSS}(C_c) \cdot |C_c| + \mathrm{SS}(o_\texttt{i}, C_c)}{|C_c| + 1} - \mathrm{WCSS}(C_c) \quad \text{tel que } \texttt{x[i]} \text{ représente } o_\texttt{i} \tag{4.3}$$

avec :

$$\mathrm{SS}(o, C) = \sum_{o_i \in C} \|o_i - o\|^2 \qquad \text{tel que} \quad C \subseteq O \tag{4.4}$$

On propose de garder cette heuristique car elle semble de bonne qualité. En effet, celle-ci suit un principe souvent vu en CP et qui a fait ses preuves. Par exemple, dans [41, 42], on définit une mesure d'impact (Impact Based Search, IBS) ou d'activité (Activity Based Search, ABS) pour chaque couple variable-valeur et on fait un choix selon un principe max-min semblable à celui discuté plus haut.

4.3 Bris d'égalité dynamique

4.3.1 Motivation

A première vue, on avancerait l'argument que briser des égalités entre choix de branchements est inutile en résolution de MSSC car, pour un ensemble de données arbitraire, les chances que deux branchements distincts engendrent le même score selon l'équation 4.3 sont presque nulles. Cependant, les lecteurs attentifs se rappelleront la constatation 4.1 et se rendront compte que les cas d'égalité sont loin d'être si rares.

On attire l'attention sur l'arrivée à un noeud de l'arbre de recherche où au moins un groupe est complètement vidé de ses éléments. Ceci arrive typiquement après plusieurs retours en arrière à la recherche de meilleures solutions. Dans ce cas, l'heuristique discutée à la section 4.2 assigne le premier point libre au premier groupe vide dont l'index appartient au domaine de sa variable représentative.

Il devient clair qu'un bris d'égalité s'avère indispensable pour avoir un meilleur contrôle sur la taille de l'arbre de recherche et ainsi, améliorer les performances. La section suivante en présente un qui peut aider à la résolution.

4.3.2 Bris d'égalité dynamique sur la somme des carrés des points libres

L'idée générale derrière la méthode recherchée est qu'on essaye, à travers elle, de maximiser les chances de réduire l'arbre de recherche en forçant le moteur CP à identifier les mauvaises branches plus tôt. En effet, l'endroit de départ des groupes joue un rôle significatif lors du calcul des quantités qui influencent les branchements (en l'occurrence les bornes inférieures).

On propose alors de brancher sur la variable qui représente l'observation libre la plus recluse. Une façon de faire cela est choisir la variable dont la somme des carrés est maximale par rapport aux points libres.

On choisit alors de brancher sur le point $o^\star$ tel quel :

$$o^\star = \underset{o \in U}{\operatorname{argmax}} \, \mathrm{SS}(o, U) \tag{4.5}$$

avec U ensemble des points libres.

4.4 Résumé

On propose dans ce chapitre une amélioration de la stratégie de recherche proposée avec CPC (voir section 3.3.1).

Premièrement, on remplace son initialisation vorace par l'usage d'heuristiques avancées et établies. Cela nous permet de démarrer à partir de meilleures solutions et de contrôler l'ordre des premiers branchements menant à une solution initiale. Pour améliorer les performances, on retient deux ordres prometteurs.

Deuxièmement, on renforce la recherche principale avec un mécanisme qui permet d'éviter un problème identifié avec la présence de groupes vides dans les solutions partielles. Ce mécanisme permet un meilleur contrôle des choix effectués dans une perspective d'amélioration des performances.

CHAPITRE 5 PARTITIONNEMENT AVEC CONTRAINTES DE CARDINALITÉ

Ce chapitre s'intéresse quant à lui à la seconde contribution souhaitée. On y discute deux versions d'une nouvelle contrainte globale axée sur l'accélération de la résolution du ccMSSC. On commence alors par un bref retour sur le calcul des contributions individuelles des observations. Par la suite, on présente un premier algorithme de filtrage simple et rapide. Enfin, on identifie les limites de cette dernière approche pour en proposer une autre, certes gourmande en calculs, mais produisant des valeurs plus fidèles.

5.1 Retour sur le calcul des plus petites contributions des groupes dans CPC

L'algorithme de filtrage est d'abord présenté, entre autres, avec une instanciation partielle des variables x. Afin de procéder à l'élimination des valeurs qui ne peuvent produire de meilleures solutions lors de la recherche, il est nécessaire de calculer des bornes (voir section 3.3.1).

Pour comprendre comment les bornes inférieures sont calculées, il est utile de considérer l'illustration suivante :

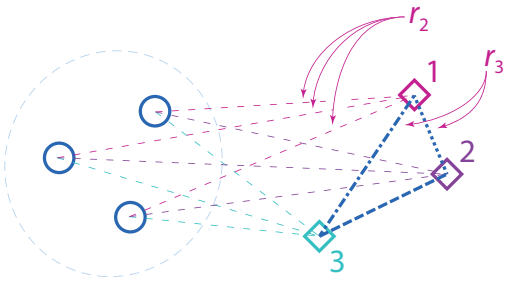

Figure 5.1 Instanciation partielle et distances utiles au calcul des bornes

La figure 5.1 montre le cas où on voudrait ajouter les points 1, 2 et 3 au groupe de gauche. Les segments marqués r_2 représentent les distances considérées entre chaque point libre et les points du groupe candidat. Les segments r_3 représentent quant à eux les distances considérées entre chaque point libre et ses voisins. Si on devait ajouter un ensemble de points libres (comme celui de la figure 5.1) à un groupe donné, le nouveau WCSS de ce dernier sera (selon l'équation 2.6) :

$$\frac{\text{WCSS}(C_c) \cdot |C_c| + \text{distances } r_2 + \text{distances } r_3}{|C_c| + \#\text{ de nouveaux points}} \tag{5.1}$$

Si on décompose le cas à la figure 5.1 de telle sorte que l'on sépare les contributions individuelles des points, on obtient les trois ensembles de distances suivantes :

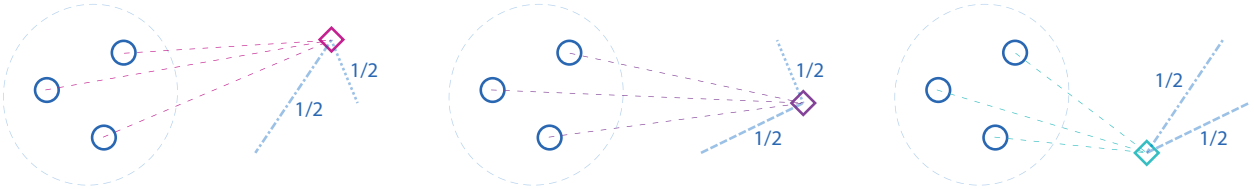

Figure 5.2 Décomposition des distances utiles aux calculs des bornes (les distances entres paires de points libres sont divisées à parts égales)

Supposons maintenant qu'on ne veuille inclure au groupe C_c que deux points quelconques et qu'on ne soit intéressé que par une borne inférieure sur le nouveau coût du groupe. On ne considère alors, pour chacun des points dans la figure 5.2, que la demi-distance r_3 la plus courte. L'autre demi-distance est écartée. Ensuite, on ne sélectionne que les deux points qui ont les sommes des distances incidentes les plus petites (total des distances r_2 plus la demi distance r_3 maintenue, voir figure 5.3). [5]

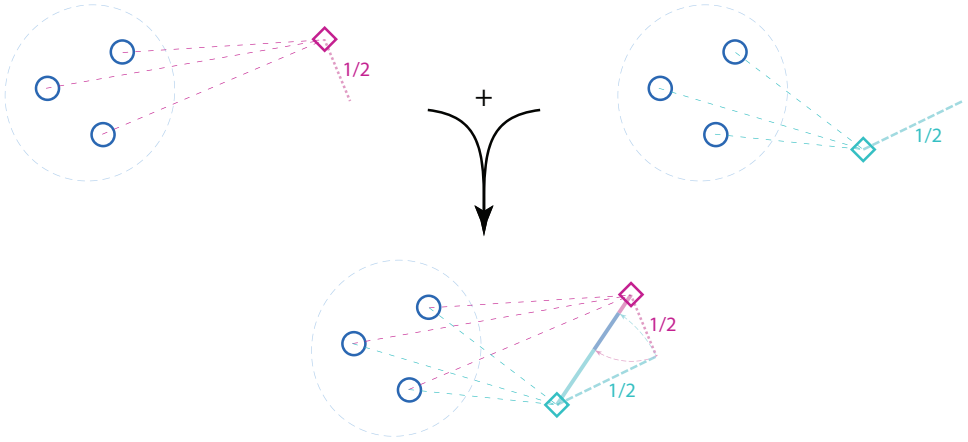

Figure 5.3 Construction de la borne inférieure du WCSS après assignation de deux points

Figure 5.4 Détail de la construction de la borne inférieure du WCSS au niveau des distances r_3

Ces deux points ainsi réunis forment une borne inférieure car, d'une part, on a sélectionné les deux points qui ont la somme des distances incidentes les plus basses et, de l'autre, dans cette somme, on n'a gardé que la demi-distance r_3 la plus courte. La figure 5.4 montre qu'en faisant ce dernier choix, on aura assurément une somme des demi-distances r_3 plus basse ou égale à la distance réelle.

Cette construction est généralisable à plus de 2 points et à tous les groupes C_c où il suffira, à chaque fois, de considérer les m points dont la somme des distances incidentes est la plus basse avec, à chaque point, les $m-1$ plus petites demi-distances r_3. En procédant de cette manière, on assure d'isoler la contribution de chaque point par rapport au reste des points quand il s'agit de sélectionner m points quelconques.

En guise d'exemple, si on se réfère à la situation de la figure 5.1, sélectionner $m = 3$ points libres revient à considérer les 3 distances r_2 ainsi que les 2 demi-distances r_3 de chacun des 3 points. Ceci revient alors à comptabiliser toutes les distances illustrées à la figure 5.2.

5.2 Premier algorithme de filtrage simple

5.2.1 Contributions minimales des groupes

Étant donné une instanciation partielle des variables du problème, il est possible, à l'aide de la construction discutée à la section 5.1, de calculer la plus petite contribution locale de chacun des groupes. A leur tour, ces contributions peuvent être utilisées pour mettre en oeuvre les filtrages nécessaires pour aider à la résolution du problème.

De façon formelle et générale [5] (voir section 3.3.1, paragraphe *Bornes inférieures locales*), la plus petite contribution d'un groupe auquel on assigne m des q points libres est telle que :

$$\underline{Z}(C_c, m) = \frac{Z(C_c) \cdot |C_c| + \sum_{i=1}^{m} R_i}{|C_c| + m} \tag{5.2}$$

avec la suite :

$$R_i = r_{2,c}(i) + \sum_{j=1}^{m} r_{3,j}(i) \tag{5.3}$$

en sachant que :

- $Z(C_c) = \mathrm{WCSS}(C_c)$;
- $m \leq q$ avec $q = |U|$, le nombre de points libres ;
- $(R_i)_{i=1}^{m}$ est une suite non décroissante : $u > v \Leftrightarrow R_u \geq R_v$;
- R_i est la contribution minimale du i-ème point libre ;
- $r_{2,c}(i)$ est la somme des distances au carré entre le point libre i et les points de C_c ;

— $(r_{3,j}(i))_{j=1}^{m}$ est une suite non décroissante : $u > v \Leftrightarrow r_{3,u}(i) \geq r_{3,v}(i)$;
— $r_{3,j}(i)$ est la demi-distance au carré entre le point libre i et un j-ème point libre selon l'ordre de la suite $(r_{3,j}(i))_{j=1}^{m}$ (incluant i, i.e. $r_{3,1}(i) = 0\ \forall i$).

Les formules plus haut sont destinées à un cas général. Or, la contribution du présent travail est axée sur le ccMSSC. C'est-à-dire :

$$|C_c|_\Delta = n_c \quad \forall c \in [0, ..., k[\qquad | \qquad \sum_{c \in [0,...,k[} n_c = n \tag{5.4}$$

avec n_c des données du problèmes relatives aux cardinalités voulues pour chacun des C_c dans le partitionnement final Δ.

Ainsi, en tout temps, on sait quels m_c interviennent à la formule 3.3 pour le calcul des différents $\underline{Z}(C_c, m_c)$, avec :

$$m_c = n_c - |C_c| \tag{5.5}$$

Dans ce cas, on réécrit :

$$\underline{Z}(C_c, m_c) = \underline{Z}_0(C_c) \tag{5.6}$$

Armé de $\underline{Z}_0(C_c)$, il est possible maintenant de mettre en oeuvre les filtrages pertinents au ccMSSC.

5.2.2 Borne inférieure globale dans le cas d'un ccMSSC

D'après 3.3, en conjonction avec 5.5 et avec 5.6, la borne inférieure globale du problème, quand les variables x sont partiellement instanciées, est telle que :

$$\underline{Z}(\{C_0, ..., C_{k-1}\}) = \sum_{c=0}^{k-1} \underline{Z}_0(C_c) \tag{5.7}$$

La constatation 5.5 élimine alors le besoin de résoudre la formule 3.3 à travers la formule de récurrence 3.4. Ceci contribue à rendre la résolution du ccMSSC plus efficace en agissant sur deux aspects distincts :

1. remplace un calcul fastidieux de complexités temporelle $\Theta(kq^2)$ et spatiale $\Theta(kq)$ (équation 3.4) par un autre, plus simple, de complexités respectives $\Theta(k)$ et $\Theta(1)$ (équation 5.7) ;

2. produit des bornes inférieures plus serrées car celles-ci ne considèrent qu'un cas particulier de répartition des points libres entre les groupes (jusqu'à leurs complétions respectives), plutôt que le cas général.

D'après les acquis des chapitres 1 et 3, on s'attend à ce que ces deux aspects aient un effet conséquent sur les performances de résolution du ccMSSC par la CP. Ceci est par contraste à l'approche classique, qui fait valoir l'aspect déclaratif de la CP et qui consiste à contraindre les cardinalités des groupes avec une contrainte séparée dans le même modèle. Cette dernière approche représente une perte d'information sur la structure du problème qu'on ne pourrait plus faire valoir pour accélérer la résolution.

Finalement, la formule 5.7 vient resserrer le domaine de Z selon le mécanisme suivant :

$$\mathrm{LB}(\mathsf{Z}) \leftarrow \underline{Z}(\{C_0, ..., C_{k-1}\}) \qquad \text{tel que} \quad \mathsf{Z} \in [\mathrm{LB}(\mathsf{Z}), \mathrm{UB}(\mathsf{Z})] \tag{5.8}$$

Si à n'importe quel moment la borne calculée à 5.7 dépasse $\mathrm{UB}(\mathsf{Z})$ (i.e., $\mathsf{Z} = \varnothing$), la branche est interrompue et un retour en arrière a lieu.

5.2.3 Bornes relatives aux valeurs et filtrage

Afin de filtrer une valeur du domaine d'une variable dans x, on évalue ici une borne inférieure globale du problème, qui suppose l'affectation de ladite valeur à cette variable.

Une façon naïve mais qui gaspille une énorme quantité des ressources est de reprendre les sections précédentes pour toutes les affectations possibles et venir comparer le résultat de la formule 5.7 à $\mathrm{UB}(\mathsf{Z})$. Si celle-ci dépasse cette borne, la valeur relative à l'affectation supposée doit être éliminée.

On peut formuler cependant une procédure de recyclage des calculs faits afin de réévaluer l'équation 5.7 pour chaque couple variable-valeur relatif à un des q points libres donnés.

Soit $\mathcal{C}$ l'ensemble des groupes partiellement remplis dont $C_c^\star \in \mathcal{C}$ est muni du ℓ-ème point libre $o^\star$, alors :

$$\begin{aligned}
\underline{Z}(\mathcal{C}) &= \underline{Z}_0(C_0) + ... + \underline{Z}_0(C_c^\star) + ... + \underline{Z}_0(C_{k-1}) \\
&= \underline{Z}(\mathcal{C} \setminus C_c^\star) + \underline{Z}_0(C_c^\star) \\
\underline{Z}(\mathcal{C}) &= \underline{Z}(\{C_0, ..., C_{k-1}\}) - \underline{Z}_0(C_c) + \underline{Z}_0(C_c^\star)
\end{aligned} \tag{5.9}$$

On a accès à tous les termes dans la formule 5.9 sauf au dernier. On essaye alors de calculer une borne inférieure de la contribution du groupe C_c, si celui-ci est supposé contenir le point libre $o^\star$, de cette façon :

$$\begin{aligned}
\underline{Z}_0(C_c^\star) &= \frac{\underline{Z}(C_c, m_c - 1) \cdot (|C_c| + m_c - 1) + \text{contribution du } \ell\text{-ème pt libre}}{|C_c| + m_c} \qquad (5.10)\\
&\geq \frac{Z(C_c) \cdot |C_c| + \sum_{i=1}^{m_c-1} R_i + \sum_{i=1}^{m_c-1} r_{3,m_c}(i) + r_{2,c}(\ell) + \sum_{j=1}^{m_c} r_{3,j}(\ell)}{|C_c| + m_c}\\
&= \frac{Z(C_c) \cdot |C_c| + \sum_{i=1}^{m_c-1} \left[r_{2,c}(i) + \sum_{j=1}^{m_c-1} r_{3,j}(i)\right] + \sum_{i=1}^{m_c-1} r_{3,m_c}(i) + r_{2,c}(\ell) + \sum_{j=1}^{m_c} r_{3,j}(\ell)}{|C_c| + m_c}\\
&= \frac{Z(C_c) \cdot |C_c| + \sum_{i=1}^{m_c-1} \left[r_{2,c}(i) + \sum_{j=1}^{m_c-1} r_{3,j}(i) + r_{3,m_c}(i)\right] + r_{2,c}(\ell) + \sum_{j=1}^{m_c} r_{3,j}(\ell)}{|C_c| + m_c}\\
&= \frac{Z(C_c) \cdot |C_c| + \sum_{i=1}^{m_c-1} \left[r_{2,c}(i) + \sum_{j=1}^{m_c} r_{3,j}(i)\right] + r_{2,c}(\ell) + \sum_{j=1}^{m_c} r_{3,j}(\ell)}{|C_c| + m_c}\\
\underline{Z}_0(C_c^\star) &\geq \frac{(|C_c| + m_c - 1) \cdot \underline{Z}_1(C_c) + r_{2,c}(\ell) + \sum_{j=1}^{m_c} r_{3,j}(\ell)}{|C_c| + m_c} \qquad (5.11)
\end{aligned}$$

La contribution du ℓ-ème point dans 5.10 rassemble les quantités suivantes :

- la somme des dissemblances entre celui-ci et les composants du groupe C_c : $r_{2,c}(\ell)$;
- la moitié des dissemblances entre celui-ci et le reste des $m-1$ points libres, qui est supérieure ou égale à $\sum_{j=1}^{m_c} r_{3,j}(\ell)$;
- l'autre moitié des dissemblances entre celui-ci et le reste des $m-1$ points libres, qui est supérieure ou égale à $\sum_{i=1}^{m_c-1} r_{3,m_c}(i)$.

Par ailleurs, on remarque dans 5.11 que $\underline{Z}_1(C_c)$ est tel que :

$$\underline{Z}_1(C_c) = \frac{Z(C_c) \cdot |C_c| + \sum_{i=1}^{m_c-1} R_i}{|C_c| + m_c - 1} \qquad \text{tel que} \quad R_i = r_{2,c}(i) + \sum_{j=1}^{m_c} r_{3,j}(i) \qquad (5.12)$$

dont la majeure différence avec $\underline{Z}_0(C_c)$ est qu'on considère $m_c - 1$ termes de R_i au lieu de m_c (voir équations 5.2, 5.6 et 5.3). Le fait que $\underline{Z}_0(C_c)$ et $\underline{Z}_1(C_c)$ soient si semblables veut dire qu'on peut les calculer tous les deux en même temps avec la même complexité.

Il suffit d'utiliser 5.9 avec 5.11 pour toutes les combinaisons de $o^\star$ et de $c \in [0, k-1[$ possibles. Si $\underline{Z}(\mathcal{C}) > \text{UB}(\mathtt{Z})$, la valeur c est filtrée du domaine de la variable qui représente $o^\star$.

Le filtrage opéré comme stipulé dans les formules 5.9 et 5.11 permet, à l'instar de la section 5.2.2 :

1. d'éviter un calcul fastidieux relatif au cas général du partitionnement de données dont les complexités temporelle et spatiale respectives sont $\Theta(q)$ et $\Theta(q)$ (équations 3.5 et 3.6), en le remplaçant par un autre de complexités respectives $\Theta(1)$ et $\Theta(1)$ (équation 5.11) ;
2. de produire des bornes inférieures plus serrées en ne considérant qu'un cas particulier de répartition des points libres entre les groupes, plutôt que le cas général.

On s'attend encore une fois, pour les mêmes raisons avancées à la section 5.2.2, que les points discutés plus haut aient un impact mesurable sur les performances de résolution du ccMSSC.

5.2.4 Déclenchement de la contrainte

En CP, on peut contrôler les conditions qui font déclencher la propagation d'une contrainte (voir section 1.6). Dans le cas de la présente contrainte :

— On choisit un évènement domaine pour les variables `x` car tout changement de domaine affecte le calcul des bornes pour tous les cas de figure ;
— On choisit un évènement borne pour la variable `Z` car toute modification du domaine de celle-ci ne se produit qu'à ses deux bouts.

Les algorithmes de propagation qu'on développe tiennent compte de ces conditions afin d'engager un filtrage adéquat quand ils sont appelés à opérer.

5.2.5 Nouvel algorithme standard de filtrage

On présente dans cette section la méthode `propagate` de la contrainte qu'on propose pour résoudre le ccMSSC. Il s'agit ici d'un résumé algorithmique des sections précédentes. `propagate`, qui s'occupe des filtrages à chaque noeud de l'arbre de recherche en cas de réveil, se divise ici en deux parties. D'une part, le filtrage de `Z` présenté à l'algorithme 5.2.1 et, de l'autre, le filtrage des variables `x` présenté à l'algorithme 5.2.2.

Algorithme 5.2.1 `propagate` : filtrage de `Z`

1: **for** $c \leftarrow 0\,..\,k-1$ **do**
2: **for** $i \leftarrow 0\,..\,n-1 \mid |D(\texttt{x[}i\texttt{]})| > 1$ **do** ▷ on itère sur les q points libres
3: **if** $c \in D(\texttt{x[}i\texttt{]})$ **then**
4: $r_2[c,i] \leftarrow \sum_{o_j \in C_c} \|o_i - o_j\|^2$
5: **else**
6: $r_2[c,i] \leftarrow \infty$
7: **for** $i \leftarrow 0\,..\,n-1 \mid |D(\texttt{x[}i\texttt{]})| > 1$ **do** ▷ idem, q^2 itérs avec ces 2 boucles car q pts libres
8: **for** $j \leftarrow 0\,..\,n-1 \mid |D(\texttt{x[}j\texttt{]})| > 1$ **do**
9: $r_3[i,j] \leftarrow \frac{1}{2} \cdot \|o_i - o_j\|^2$
10: `sort` $(r_3[i, j \in 0..n \mid |D(\texttt{x[}j\texttt{]})| > 1])$ ▷ usuellement, complexité temporelle $\mathcal{O}(q \log q)$
11: **for** $j \leftarrow 1\,..\,\left(\max_{c \in [0,k[} m_c\right) - 1$ **do**
12: $r_3[i,j] \leftarrow r_3[i,j] + r_3[i,j-1]$ ▷ r_3 ici représente la somme dans 5.3 directement
13: **for** $c \leftarrow 0\,..\,k-1$ **do**
14: **for** $v \leftarrow 0\,..\,1$ **do**
15: **for** $i \leftarrow 0\,..\,n-1 \mid |D(\texttt{x[}i\texttt{]})| > 1$ **do**
16: **if** $m_c - v > 0$ **then**
17: $R[i] \leftarrow r_2[c,i] + r_3[i, m_c - 1]$ ▷ voir équation 5.3
18: `sort` $(R[i \in 0..n \mid |D(\texttt{x[}i\texttt{]})| > 1])$ ▷ idem, complexité temporelle $\mathcal{O}(q \log q)$
19: $\underline{Z}[c,v] \leftarrow \frac{Z(C_c) \cdot |C_c| + \sum_{i=0}^{m_c - v - 1} R[i]}{m_c + |C_c| - v}$ ▷ voir équations 5.2, 5.6 et 5.12
20: $\mathrm{LB}(\texttt{Z}) \leftarrow \sum_{c=0}^{k-1} \underline{Z}[c,0]$ ▷ voir équations 5.7 et 5.8

Analyse et commentaires sur l'algorithme 5.2.1 :

- les lignes entre 1 et 12 permettent de calculer tous les termes intervenants dans les suites présentées à la section 5.2.1 ;
- la boucle aux lignes 1–6 permet de calculer les $r_{2,c}$ pour tous les points libres et tous les groupes. Si un point donné ne peut pas faire partie d'un groupe quelconque (dû à une propagation antérieure), une dissemblance de valeur infinie est attribuée pour éliminer cette affectation. En moyenne, les groupes comportent n/k éléments : complexité temporelle $\mathcal{O}(kq(n/k)) = \mathcal{O}(qn)$;
- la boucle aux lignes 7–12 permet de calculer les sommes de $r_{3,j}(i)$ pour tous les points libres. Complexité temporelle $\mathcal{O}(q(q + q \log q + q)) = \mathcal{O}(q^2 \log q)$;
- la boucle aux lignes 13–19 permet de calculer les $\underline{Z}_0(C_c)$ et $\underline{Z}_1(C_c)$ pour tous les groupes. Complexité temporelle $\mathcal{O}(k \cdot 1 \cdot (q + q \log q + q)) = \mathcal{O}(kq \log q)$;

— la ligne 20 rassemble les contributions minimales des groupes pour calculer la nouvelle borne inférieure de Z. Comme vu à la section 5.2.2, la complexité temporelle est $\mathcal{O}(k)$;
— la complexité temporelle globale est $\mathcal{O}(qn + q^2 \log q + kq \log q) = \mathcal{O}(qn + q^2 \log q)$ car il y a toujours autant ou plus de points libres que de groupes vides ;
— la complexité spatiale globale est $\mathcal{O}(n^2)$.

Algorithme 5.2.2 `propagate` : filtrage des variables du tableau `x`

1: **for** $c \leftarrow 0 \,..\, k-1$ **do**
2: $\quad \mathrm{LB}_E \leftarrow \mathrm{LB}(\mathtt{Z}) - \underline{Z}[c, 0]$
3: $\quad$ **for** $i \leftarrow 0 \,..\, n-1 \mid |D(\mathtt{x}[i])| > 1$ **do** ▷ idem qu'avant, q itérs pour q pts libres
4: $\quad\quad$ **if** $c \in D(\mathtt{x}[i])$ **then**
5: $\quad\quad\quad \mathrm{LB}_P \leftarrow \frac{(|C_c|+m_c-1)\cdot \underline{Z}[c,1]+r_2[c,i]+r_3[i,m_c-1]}{|C_c|+m_c}$ ▷ voir équations 5.9 et 5.11
6: $\quad\quad\quad$ **if** $(\mathrm{LB}_E + \mathrm{LB}_P \geq \mathrm{UB}(\mathtt{Z}))$ **then**
7: $\quad\quad\quad\quad D(\mathtt{x}[i]) \leftarrow D(\mathtt{x}[i]) \setminus \{c\}$ ▷ filtrer si borne dépassée

Analyse et commentaires sur l'algorithme 5.2.2 :
— filtrer le domaine des variables à la ligne 7 se fait en temps constant. Ceci est possible car le domaine des variables est un ensemble épars (*sparse set*, où les opérations de retrait se font en complexité temporelle dans $\Theta(1)$ [43]) ;
— la complexité temporelle globale est $\Theta(kq)$;
— la complexité spatiale globale est $\Theta(1)$.

Les complexités spatiale et temporelle de `propagate` pour cette contrainte sont $\mathcal{O}(qn + q^2 \log q + kq) = \mathcal{O}(qn + q^2 \log q)$ et $\mathcal{O}(n^2)$, respectivement.

5.2.6 Limites

Bien que cet algorithme soit extrêmement rapide en pratique, il souffre d'une limite importante. Pour cela, on attire le lecteur à la situation décrite à la figure 5.5.

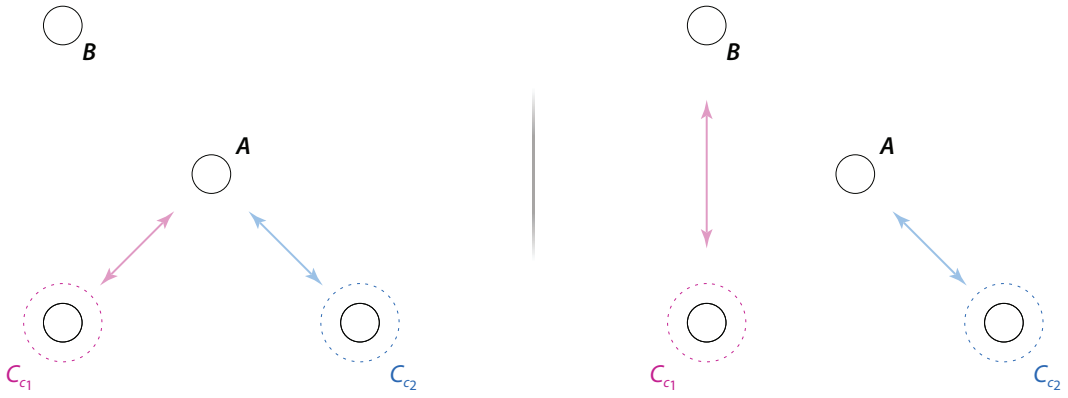

Figure 5.5 Observations considérées lors du calcul de la borne inférieure d'une instanciation partielle

Compte tenu du fait que les contributions minimales de chaque groupe sont calculées de façon locale et indépendante du reste des groupes de la partition, un même point libre peut donc être considéré dans le calcul de la contribution minimale de deux (ou plusieurs) groupes distincts à la fois. Dans le cas de la figure 5.5 (gauche), le même point libre A sera considéré pour les deux groupes c_1 et c_2, ce qui porte atteinte à la qualité des bornes calculées (en effet une meilleure borne serait, dans le cas de la figure 5.5, de considérer le point B pour le groupe c_1 et le point A pour le groupe c_2, voir schéma de droite).

La section qui suit présente un algorithme destiné à éliminer cette limite à travers une méthode de calcul alternative.

5.3 Deuxième algorithme de filtrage avancé

5.3.1 Idée

Au lieu de considérer les affectations et les groupes de façon indépendante, on essaye ici de considérer le problème comme un tout.

D'une part, on a une liste de points libres qui, une fois assignés à un groupe donné, augmentent l'objectif d'une quantité dont on connaît une borne inférieure (suite 5.3 divisée par la cardinalité voulue du groupe, n_c). De l'autre, on a une liste de groupes partiellement remplis

qui doivent accueillir chacun m_c de ces points libres. En outre, on impose qu'un point ne puisse être assigné qu'à un seul groupe à la fois.

Pour illustrer le dernier paragraphe, on considère le graphe biparti suivant :

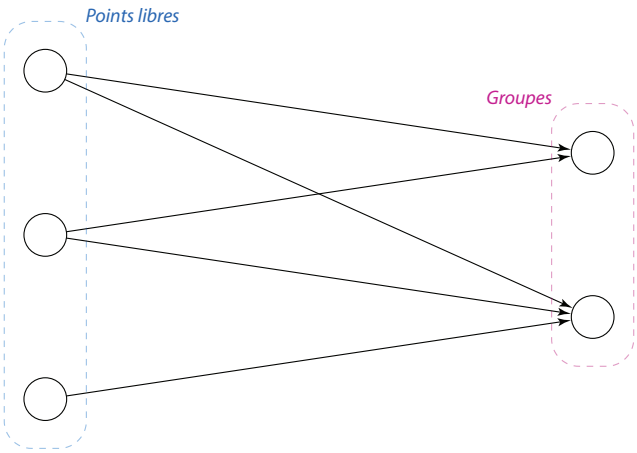

Figure 5.6 Graphe d'affectation des observations libres

Dans le graphe plus haut :

- les sommets de gauche représentent les q points libres ;
- les sommets de droite représentent les groupes qui ne se sont toujours pas complètement remplis (i.e., $m_c \neq 0$) ;
- un arc n'existe que pour relier un point à un groupe qui peut l'accueillir (e.g., le 3e point libre dans la figure 5.6 ne peut pas appartenir au premier groupe) ;
- chaque arc est muni d'un poids égal à la contribution minimale au problème du i-ème point libre à son origine quand il est assigné au groupe C_c à sa destination (voir équations 5.3 et 5.2) :

$$\frac{r_{2,c}(i) + \sum_{j=1}^{m_c} r_{3,j}(i)}{|C_c| + m_c} = \frac{r_{2,c}(i) + \sum_{j=1}^{m_c} r_{3,j}(i)}{n_c} \tag{5.13}$$

- seul un des arcs qui ont une même origine peut être utilisé.

Le problème revient désormais à trouver le sous-ensemble d'arcs de poids minimal qui relie les sommets de droite à ceux de gauche et qui respecte les règles prescrites. Ce sous-ensemble nous permettra de calculer une borne inférieure globale $\underline{Z}(\{C_0, ..., C_{k-1}\})$ qui prend en compte

tous les points libres et tous les groupes partiels, à la fois, sans jamais considérer un même point plus d'une fois.

De prime abord, cela semble comme un problème difficile. Cependant le lecteur attentif se rappellera la section 3.5 dans laquelle on parle d'un problème similaire.

5.3.2 Comparaison avec GCC avec coûts

De manière superficielle, le présent problème revient à une GCC avec coûts. Cependant, ceci n'est pas tout à fait le cas. En l'occurrence, dans une GCC avec coûts, on a affaire à des coûts fixes, décidés avant la résolution du problème. Ici, les coûts :

— ne sont pas disponibles avant d'avoir entamé la résolution du problème (on rappelle la constatation 4.1) ;
— sont dynamiques et dépendent des affectations courantes à chaque noeud de l'arbre de recherche.

Cela veut dire que l'application d'une GCC avec coûts à ce problème est inadéquate. De plus, la possibilité de tirer profit des calculs incrémentiels de celle-ci est largement perdue. Néanmoins, on peut s'inspirer du fonctionnement général de celle-ci pour dériver un autre algorithme, certes semblable, mais plus spécialisé et surtout compatible avec le problème en main.

5.3.3 Calcul de la borne inférieure globale à travers un réseau de flot

De même que pour la GCC avec coûts, il est simple de transformer le problème de la section 5.3.1 en un problème de réseau de flot dans lequel on cherche un flot de coût minimal.

D'abord, on commence par alimenter chacun des sommets relatifs aux points libres par une unité de fluide, délivrée par une source commune. Vu que chaque point libre n'est alimenté que par une seule unité de fluide, on a l'assurance que chaque point ne sera compté qu'une seule fois dans la contribution totale. Ensuite, on relie tous les sommets relatifs aux groupes à un puits commun pour récolter le flot qui transite à travers le réseau.

Par ailleurs, les arcs supplémentaires ajoutés sont tous de coût nul. Les arcs existants portent le coût de la contribution minimale discutée à la section 5.3.1. Tous les arcs ont une capacité contrainte à être entre 0 et 1 sauf pour les arcs qui relient les groupes au puits commun : ceux-ci ont une capacité limitée non pas à 1 mais au m_c correspondant (on ne veut pas qu'un groupe se fasse assigner plus de points qu'il ne peut en accueillir).

Au total, on fait transmettre q unités de flot pour les q points libres à travers le réseau. S'il existe une répartition possible des points sur les groupes jusqu'à leurs complétions respectives, la totalité du flot va passer de la source h (de capacité q) au puits t (de capacité $-q$), à travers tous les sommets. Sinon le problème n'admet pas de solution et devra provoquer un échec et un retour en arrière dans la recherche entamée par le moteur CP.

La figure 5.7 ci-dessous montre les modifications discutées plus haut appliquées à l'exemple de la figure 5.6.

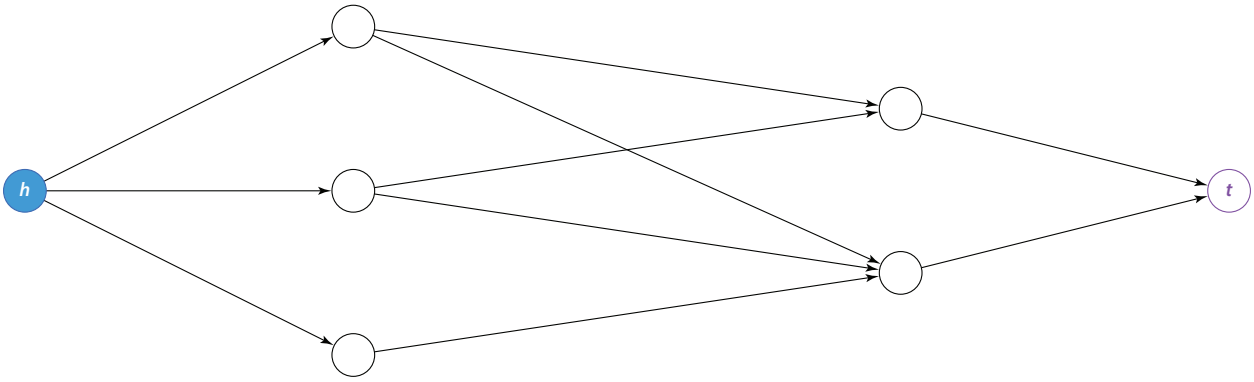

Figure 5.7 Réseau dérivé du graphe d'affectation

Une fois le problème transformé en un autre de MCF équivalent, on peut le résoudre avec Network Simplex pour obtenir une borne inférieure globale de meilleure qualité, égale à :

$$\underline{Z}(\{C_0, ..., C_{k-1}\}) = \sum_{c \in [0,k[} \frac{Z(C_c) \cdot |C_c|}{n_c} + \text{MCF}^*_{\text{obj}} \tag{5.14}$$

avec $\text{MCF}^*_{\text{obj}}$ le coût de la solution optimale du MCF.

Contrairement à GCC avec coûts, un MCF doit être résolu intégralement à chaque fois (aucune possibilité de faire un calcul incrémentiel à cause des coûts dynamiques). Cependant, la taille du problème diminue avec le nombre d'observations fixées : moins il y a d'observations libres, plus facile est le MCF. On choisit aussi d'utiliser Network Simplex compte tenu de l'absence de calcul incrémentiel ainsi que ses performances réelles meilleures comparé à l'algorithme des chemins successifs de poids minimaux [44].

Par ailleurs, il est à noter que la solution du LP relatif au MCF discuté plus haut est nécessairement entière car (voir section 2.2.3) :

- la matrice $\mathbf{A}$ est totalement unimodulaire (toutes ses colonnes contiennent un seul 1, un seul -1 et que des 0) ;
- le vecteur b n'est composé que d'entiers (tous les s_v sont soit égaux à 0, à q ou à $-q$) ;
- toutes les bornes sont entières (égales soit à 0, soit à 1 soit à m_c).

Cela veut dire qu'aucun des arcs du réseau n'accueille qu'une partie du flot : ils sont soit complètement occupés soit complètement vides. Cela veut dire aussi qu'un point libre ne peut qu'être entièrement affecté à un groupe donné, jamais divisé entre deux.

De façon analogue aux méthodes précédentes, on s'attend, malgré la charge de calcul complémentaire, à ce que cette nouvelle méthode mène à des gains de performance conséquents compte tenu des bornes dont la qualité est encore meilleure et qui tirent profit au maximum de la structure des ccMSSC.

5.3.4 Nouvelle procédure de Cost-Based Filtering

Vu que l'on considère maintenant le problème comme un tout (par contraste à un ensemble d'éléments indépendants), on ne peut plus utiliser les mêmes techniques de recyclage de calculs pour accélérer le processus de filtrage des variables représentatives `x` (comme vu précédemment). Toute méthode de recalcul des bornes doit désormais prendre en compte, à son tour, le problème dans son entièreté.

Une première façon naïve de faire cela est d'introduire, à chaque fois et pour chaque affectation possible (i.e., chaque couple point libre/groupe), une contrainte supplémentaire au MCF qui viendrait obliger l'affectation du point en particulier à un groupe donné. On résoudrait ce nouveau MCF augmenté par cette contrainte et on comparerait la nouvelle borne calculée à la borne supérieure de `Z`. Si on la dépasse, on procède au filtrage correspondant à la supposition faite. La même remarque sur les réseaux sans solutions s'applique ici aussi.

En pratique cette solution résulte en un gâchis considérable des ressources et doit être écartée. On considère, à la place, la solution optimale du MCF sans la contrainte supplémentaire (exemple à la figure 5.8, gauche), laquelle on modifie pour supposer une certaine affectation en particulier (figure 5.8 droite).

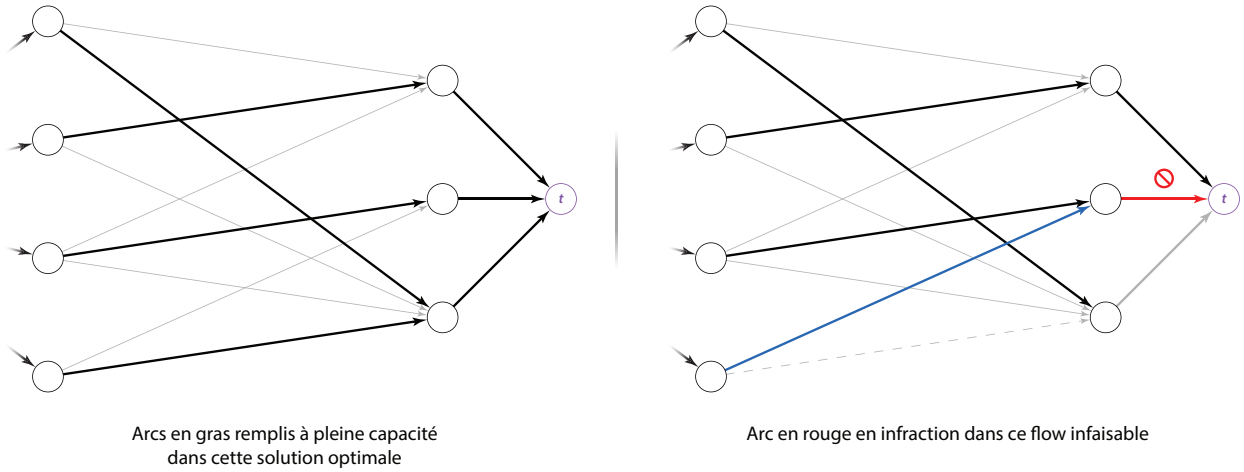

Figure 5.8 Fausse modification de la solution optimale d'un MCF

Le réseau de droite représente une mauvaise solution au problème vu qu'un groupe en particulier accueille plus de flot qu'il ne peut en admettre (une unité en trop de fluide). En contrepartie, un autre groupe en accueille moins qu'il ne doit en avoir (une unité en moins de fluide).

La façon de régler ce problème (et tomber sur une solution optimale au MCF augmenté) est de retirer un point au groupe qui a un excès de flot. Ce point devra être affecté à un autre groupe. Peut-être celui à qui il manque une unité de flot ? Pas nécessairement. La solution trouvée sera faisable, mais pas obligatoirement optimale. Afin de retrouver la solution optimale, il est nécessaire d'évaluer tous les retraits et affectations qui résultent en le coût supplémentaire minimal.

On procède alors à une série alternée de retraits et d'affectations qui viendra ramener l'unité de fluide en trop vers le groupe qui en a une de moins et qui va résulter en le coût induit le plus bas (assurant ainsi l'optimalité de la solution). Afin d'illustrer cela, considérons l'exemple 5.3.1.

Exemple 5.3.1. Soit le problème de MCF à la figure 5.9 où tous les arcs ont une capacité permise entre 0 et 1, sauf celui qui relie le sommet B3 au puits t où la capacité doit se trouver entre 0 et 2.

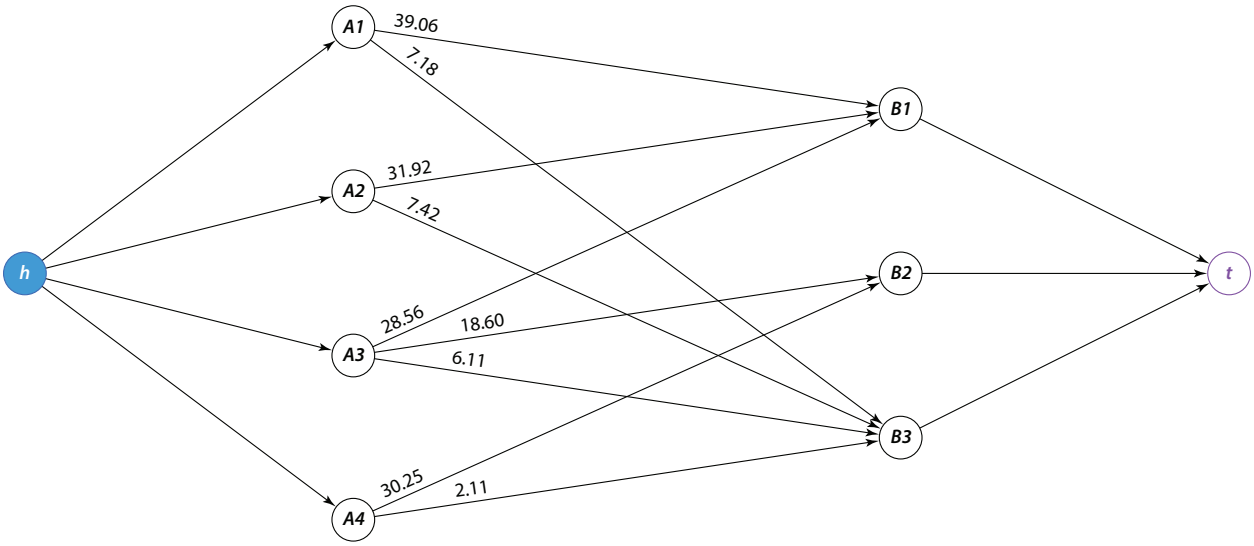

Figure 5.9 Réseau dérivé d'un exemple de graphe d'affectation

À la figure 5.9, les quantités indiquées sur les arcs représentent les coûts de ceux-ci. Les arcs non marqués ont un coût nul. Le problème indiqué en haut admet la solution optimale suivante, de coût 59.81 :

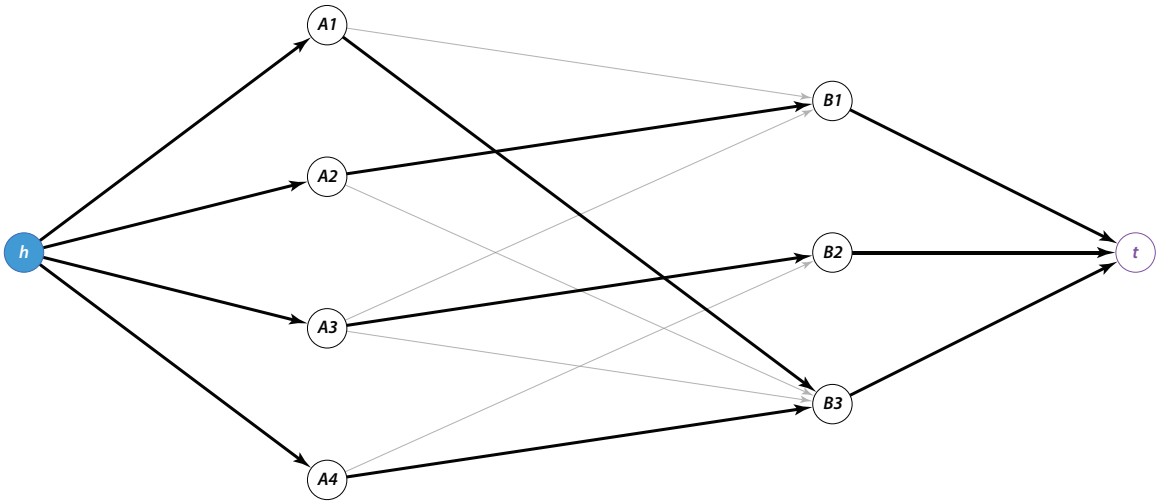

Figure 5.10 Flot minimal sur le réseau dérivé

Les arcs en gras sont occupés à pleine capacité. Les arcs fins et transparents ne transportent aucun flot (on rappelle la remarque sur la nature entière des solutions de ce problème, voir section 5.3.3).

Supposons maintenant que l'on veuille contraindre l'arc (A4, B2) à transporter du flot (dans le contexte de ce travail de recherche, on rappelle que cela reviendrait à obliger l'affectation d'un point donné à un groupe en particulier). On se retrouve avec un excès de flot au sommet B2.

Afin de retrouver une solution optimale après imposition de la contrainte, il suffit de procéder, dans l'ordre, à :

1. enlever une unité de flot de (A3, B2)
2. mettre cette unité de flot dans (A3, B1)
3. enlever une unité de flot de (A2, B1)
4. mettre cette unité de flot dans (A2, B3)

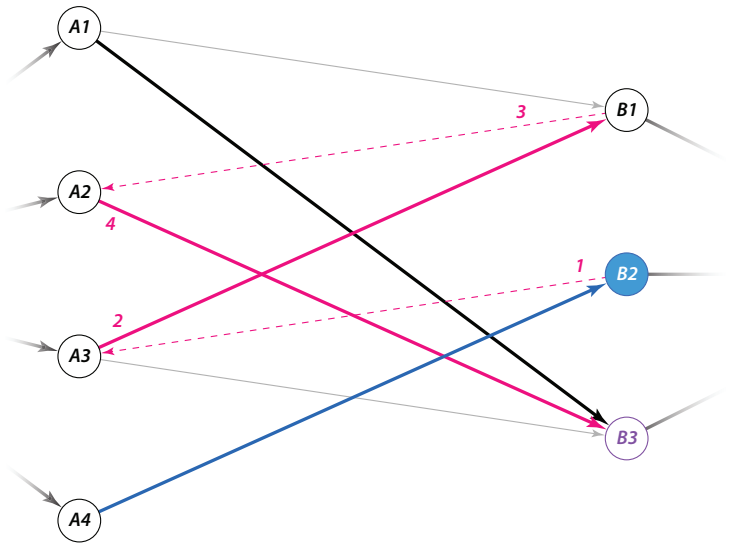

Figure 5.11 Solution optimale du problème augmenté

Cette série d'opérations assure un coût supplémentaire minimal, conduisant à une nouvelle solution optimale (de coût 73.41, une fois l'arc correspondant à la contrainte supplémentaire comptabilisé). Ces opérations ressemblent en fait à un chemin qui alterne entre les deux groupes, gauche et droit, des sommets et à travers duquel on envoie l'unité de flot d'un endroit (le noeud où il y a excès) à un autre (le noeud où il y a déficit).

Si on construit le graphe résiduel — dans lequel les arcs occupés sont inversés et leurs poids remplacés par leurs opposés — suivant, issu de la solution à la figure 5.10 :

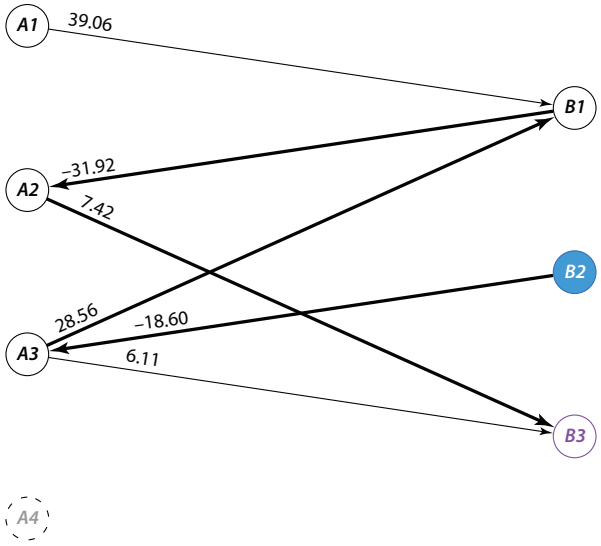

Figure 5.12 Graphe résiduel issu de la solution optimale au problème MCF

on y constante que les opérations énumérées plus haut pour retrouver la solution optimale représentent un chemin de poids minimal dans celui-ci, entre les groupes en infraction. Ce à quoi les lecteurs attentifs se doutaient déjà : retrouver la solution optimale revient à envoyer l'excès de flot à l'endroit qui peut l'accepter en empruntant le chemin dont le poids est le plus petit pour assurer l'optimalité de la nouvelle solution. Sur la figure 5.12, si on se dirige vers la droite, on ajoute un point à un groupe (et donc sa contribution) alors que l'inverse se produit dans l'autre sens (d'où les poids négatifs à travers ces arcs). △

De façon tout à fait analogue à l'exemple 5.3.1, étant donné le réseau de la section 5.3.3 ainsi que sa solution MCF optimale, on propose les étapes suivantes pour mettre à jour cette dernière, successivement et de façon incrémentielle, à la suite d'une introduction de contrainte qui suppose une affectation donnée :

1. construire le graphe biparti résiduel dans lequel les arcs occupés sont inversés et leur sont assignés des poids opposés (négatifs) ;
2. identifier les deux sommets dans lesquels il y a excès et déficit de flot après réaffectation du point concerné par la contrainte supplémentaire ;

3. éliminer tous les arcs entrants au sommet où il y a excès de flot ;
4. éliminer tous les arcs sortants du sommet où il y a déficit de flot ;
5. trouver le chemin de poids minimal qui connecte le sommet où il y a déficit de flot à celui où il y a excès avec un algorithme approprié ;
6. calculer le delta objectif minimal $\Delta\text{MCF}^*_{\text{obj}}$ en faisant la somme de la contribution due à l'affectation du point au nouveau groupe avec le poids du chemin calculé (qui peut être une quantité négative) ;
7. la valeur de l'objectif du problème augmenté par la contrainte est égale à

$$\text{MCF}^*_{\text{obj}} - \text{ancienne contribution du point} + \Delta\text{MCF}^*_{\text{obj}}$$

On applique les étapes ci-dessus pour tous les couples variable-valeur qui n'interviennent pas dans le MCF d'origine (section 5.3.3). Si la valeur de l'objectif calculée pour un certain couple dépasse UB(Z), on filtre la valeur du domaine de la variable concernée.

Cette série d'opérations ressemble à ce qu'une GCC avec coûts ferait pour filtrer les variables. Cela dit, la méthode discutée ici est plus spécialisée car on sait que tous les groupes seront remplis à pleine capacité (dans une GCC avec coûts, cela n'est pas nécessairement vrai dans la mesure où les cardinalités ne sont pas égales à une valeur mais doivent appartenir à un intervalle défini).

Aucun chemin n'existe à l'étape 5 plus haut si, et seulement si, le nouveau problème MCF augmenté est irréalisable. Si cela se produit, l'affectation étudiée est incohérente et on procède au filtrage.

Théorème 5.3.1. Si un chemin existe dans le graphe résiduel entre les sommets où il y a excès et déficit de flot, alors celui-ci ne comporte aucun cycle de poids négatif. ◀

Démonstration. Supposons qu'un tel chemin existe et comporte un cycle de poids négatif, on pourrait emprunter ce dernier autant de fois que nécessaire jusqu'à générer une solution de coût meilleur que celui du problème principal. Or l'ajout d'une contrainte ne peut résulter qu'en une solution équivalente ou pire. Celle-ci sera équivalente si la contrainte est redondante ou pire si celle-ci change la solution (qui ne peut être meilleure sinon la solution initiale n'est pas optimale vu qu'on aura trouvé une amélioration). ■

A l'instar du théorème 5.3.1, les étapes 3 et 4 éliminent des arcs qui ne peuvent être occupés qu'en cas de présence de cycles négatifs. Voir exemple 5.3.2 pour une illustration de cela.

Exemple 5.3.2. Soit les chemins abstraits à la figure 5.13, reliant un sommet h à un autre t.

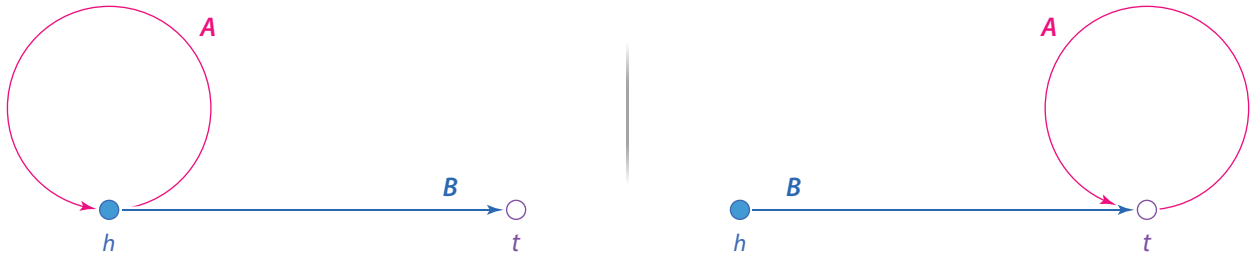

Figure 5.13 Illustration abstraite de chemins munis de cycles

Si ces deux chemins sont de poids minimaux, c'est que, forcément, dans les deux, la boucle A a un poids non positif. Ceci est parce que si son poids était positif, on pourrait l'éliminer dans les deux cas et retrouver un chemin de moindre coût, ce qui va à l'encontre de notre supposition initiale stipulant que ces deux chemins sont de coûts minimaux.

Or, selon le théorème 5.3.1, le chemin recherché dans le graphe résiduel pour le rétablissement de la solution optimale dans le MCF ne comporte pas de cycles négatifs. Ainsi, il n'est pas possible d'entrer dans le sommet de départ (respectivement de sortir du sommet d'arrivée). Ceci reviendrait à emprunter le chemin A dans les deux cas ; un chemin qui ne peut pas exister. Dans le cas où A a un poids nul, celui-ci n'a pas lieu d'être emprunté dans tous les cas.

Par conséquent, ne pas considérer à la fois les arcs incidents au départ ainsi que les arcs émanant de la destination représente une action sure qui ne provoque aucune erreur. △

5.3.5 Recyclage d'une solution antérieure de MCF

Il arrive parfois qu'un choix dans l'arbre ne change pas la solution d'un MCF courant. Cela se produit, par exemple, quand des affectations potentielles qui n'ont jamais figuré dans la solution optimale de celui-ci sont écartées sans que des observations ne soient fixées entre-temps. Si une telle situation se présente, on peut éviter de relancer Network Simplex et réutiliser la solution antérieure, économisant ainsi des ressources. Cette vérification se fait en $\mathcal{O}(n)$, très largement moins que Network Simplex.

5.3.6 Nouvel algorithme avancé de filtrage

On présente ici le résumé algorithmique de la section 5.3. Les différences avec l'algorithme standard, présenté à la section 5.2.5, sont montrées en bleu.

Algorithme 5.3.1 `propagate` : filtrage de `Z` avancé

1: **for** $c \leftarrow 0\,..\,k-1$ **do**
2: **for** $i \leftarrow 0\,..\,n-1 \mid |D(\texttt{x[}i\texttt{]})| > 1$ **do** ▷ on itère sur les q points libres
3: **if** $c \in D(\texttt{x[}i\texttt{]})$ **then**
4: $r_2[c,i] \leftarrow \sum_{o_j \in C_c} \|o_i - o_j\|^2$
5: **for** $i \leftarrow 0\,..\,n-1 \mid |D(\texttt{x[}i\texttt{]})| > 1$ **do** ▷ idem, q^2 itérs avec ces 2 boucles car q pts libres
6: **for** $j \leftarrow 0\,..\,n-1 \mid |D(\texttt{x[}j\texttt{]})| > 1$ **do**
7: $r_3[i,j] \leftarrow \frac{1}{2} \cdot \|o_i - o_j\|^2$
8: `sort` $(r_3[i, j \in 0..n \mid |D(\texttt{x[}j\texttt{]})| > 1])$ ▷ usuellement, complexité temporelle $\mathcal{O}(q \log q)$
9: **for** $j \leftarrow 1\,..\,\left(\max_{c\in[0,k[} m_c\right) - 1$ **do**
10: $r_3[i,j] \leftarrow r_3[i,j] + r_3[i,j-1]$ ▷ r_3 ici représente la somme dans 5.3 directement
11: **if** `impactfulChangeHasOccurred()` **then** ▷ voir section 5.3.5
12: `makeMCFModel()` ▷ voir figure 5.7, complexité $\mathcal{O}(kq)$
13: `solution` $\leftarrow$ `solveMCFModel()` ▷ Network Simplex, voir section 2.2.2
14: **if** `solution` $= \varnothing$ **then**
15: `fail()` ▷ si modèle irréalisable, échec + retour arrière
16: LB(`Z`) $\leftarrow$ WCSS partiel + `solution.cost()` ▷ voir équation 5.14

Algorithme 5.3.2 `propagate` : filtrage avancé des variables du tableau `x`

1: **for** $c \leftarrow 0\,..\,k-1$ **do**
2: **for** $i \leftarrow 0\,..\,n-1 \mid |D(\texttt{x[}i\texttt{]})| > 1$ **do** ▷ idem qu'avant, q itérs pour q pts libres
3: **if** $c \in D(\texttt{x[}i\texttt{]}) \wedge \neg$`solution.hasFlow(x[`$i$`], `$c$`)` **then**
4: $\delta \leftarrow$ `minPathWeightUpdate(solution, x[`i`], `c`)` ▷ voir section 5.3.4
5: **if** $\delta = \infty \vee$ LB(`Z`) $+ \delta \geq$ UB(`Z`) **then**
6: $D(\texttt{x[}i\texttt{]}) \leftarrow D(\texttt{x[}i\texttt{]}) \setminus \{c\}$ ▷ filtrer si borne dépassée ou modèle irréalisable

Le nouvel algorithme avancé est similaire au dernier avec quelques différences majeures :

— la boucle aux lignes 13–19 ainsi que la ligne 20, dans l'algorithme 5.2.1 précédent, sont remplacées par la construction ainsi que résolution du MCF correspondant à la configuration courante du problème, tel que prescrit plus haut dans la section 5.3.3.

Selon la section 2.2.2, on parle d'une complexité temporelle théorique de

$$\mathcal{O}\left((q+k)\cdot qk\cdot\log(q+k)\cdot\log\left((q+k)\cdot\max_{c\in[0,k[,i\in[0,n[|o_i\notin C_c\forall c}\frac{r_2[c,i]+r_3[i,m_c-1]}{|C_c|}\right)\right)$$

représentant, de loin, le calcul le plus lourd de l'algorithme ;

— la ligne 5 dans l'algorithme 5.2.2 est remplacée par le calcul du coût du chemin de poids minimal discuté à la section 5.3.4. On préconise l'usage de l'algorithme de Bellman-Ford pour cette tâche (étant donné la présence d'arcs de poids négatifs, les algorithmes comme celui de Dijkstra sont proscrits). La complexité temporelle de celui-ci dans le cas moyen est $\mathcal{O}(|A||V|)$ et au meilleur des cas $\mathcal{O}(|A|)$ [45]. Cela revient à $\mathcal{O}(qk(q+k))$ (moins que Network Simplex). Le test à la ligne 4 de l'algorithme 5.2.2 doit être révisé pour éliminer les suppositions redondantes (i.e., les affectations qui sont considérées dans le MCF d'origine, voir ligne 3 dans l'algorithme 5.3.2). La complexité temporelle finale est de $\mathcal{O}(q^2k^2(q+k))$.

De la même façon, on s'attend à ce que cette méthode mène à des gains considérables de performances compte tenu des meilleures bornes et ce malgré la charge de travail supplémentaire. En outre, la détection des incohérences lors du Cost-based Filtering devrait aussi aider à éliminer davantage de valeurs que la méthode précédente, dont la portée locale ne permet pas de déceler ces problèmes (voir fin de la section 5.3.4).

5.4 Autres remarques et considérations

5.4.1 Niveau de cohérence maintenu

Cette contrainte (tous algorithmes confondus) intervient sur les bornes de l'objectif ainsi que sur toutes les valeurs de toutes les variables. Elle permet alors de maintenir une cohérence de bornes vis-à-vis de l'objectif ou, au choix, une cohérence de domaine ou de bornes en ce qui concerne les variables entières représentatives.

La version étudiée ici maintient une cohérence de domaine pour les variables représentatives étant donné l'avantage lié à la réduction des domaines de la façon la plus agressive possible. De plus, rarement sont les domaines assez larges pour qu'il y ait une réduction des temps de calculs relatifs aux filtrages dans tous les cas.

Le Cost-based Filtering des valeurs doit cependant être relancé à chaque fois car l'élimination d'un arc peut provoquer des chemins différents dans le graphe résiduel lors du calcul de chemins de poids minimaux.

5.4.2 Contrainte de cardinalité externe

Afin d'assurer un filtrage rigoureux, il est important d'avoir une contrainte GCC qui surveille les cardinalité des groupes indépendamment de la présente contrainte. Celle-ci viendrait éliminer les branches qui résulteraient en une affectation qui ne respecte pas les cardinalités voulues. Celle-ci viendrait aussi éliminer les valeurs relatives à un groupe en particulier du domaine des variables lorsque celui-ci devient plein.

5.5 Résumé

On propose dans ce chapitre deux versions d'une contrainte globale destinée à réduire l'espace de recherche d'une solution de ccMSSC. Cette contrainte filtre à la fois l'objectif ainsi que les variables représentatives des observations.

La première version de celle-ci, adaptée de CPC (section 3.3.1) emploie des calculs rapides mais génère des bornes de moindre qualité. La deuxième version, quant à elle, met en oeuvre des calculs plus lourds pour produire des bornes de meilleure qualité.

CHAPITRE 6 RÉSULTATS

On définit dans cette section la méthodologie servant à tester les contributions de ce travail de recherche. On y précise alors les tests à mener ainsi que les éléments qui servent de comparaison. On y présente les résultats de façon succincte, accompagnés de commentaires. Enfin, on analyse ces résultats et explique ce qu'ils impliquent.

6.1 État actuel de résolution du ccMSSC

Comme on l'a vu au chapitre 3, il existe deux méthodes qui permettent de résoudre le MSSC en CP : CPC (section 3.3.1) et CP RBBA (section 3.3.2). Étant donné l'aspect déclaratif de la CP, il est facilement possible d'étendre ces algorithmes à la résolution du ccMSSC en introduisant des contraintes de cardinalité au modèle. Les auteurs de ces deux méthodes affirment que leurs performances dépassent celles de l'état de l'art relatif à leur période de parution [5, 10].

On aura aussi vu à la section 3.2.3 une méthode numérique, basée sur l'optimisation conique, qui peut parfois prouver l'optimalité pour certains exemplaires en ccMSSC. Par ailleurs, il existe un autre cadre de résolution de MSSC sous contraintes basé sur une méthode de génération de colonnes [36]. Celui-ci peut théoriquement être étendu au ccMSSC. Cependant, les auteurs l'on spécialisé à un autre type de contraintes (ML/CL plutôt que cardinalités).

CPC et CP RBBA représentent alors la meilleure concurrence dans le domaine et il est nécessaire de comparer le présent travail à ces deux méthodes de résolution.

De plus, bien que différents, une brève comparaison avec la méthode numérique basée sur l'optimisation conique sera donnée dans cette section pour rendre cette analyse complète.

6.2 Exemplaires choisis

La meilleure façon de comparer ces algorithmes est d'utiliser des exemplaires du problème à la fois variés mais qui sont aussi connus et souvent utilisés dans la communauté. Ceci sert un objectif double : cadrer les performances dans un référentiel connu et assurer le respect du principe de « Nothing up my sleeve, » éliminant la possibilité de générer des exemplaires destinés à cacher les faiblesses de la présente approche.

Les exemplaires retenus sont les suivants :

Tableau 6.1 Résumé des exemplaires choisis

Code	Nom	n	s	k	Cardinalités visées	Notes
AI3	Acute Inflammations	120	6	**3**	40 40 40	1, 2, 3
AI4	Acute Inflammations	120	6	**4**	30 30 30 30	1, 2
BC	Breast Cancer Coimbra	116	9	3	38 39 39	1, 2
BT	Breast Tissue	106	9	6	18 17 17 18 18 18	1, 2
CB	Connectionist Bench	208	60	2	111 97	1
CS	Concrete Slump Test	103	7	3	34 34 35	1, 2
GI	Glass Identification	214	9	6	70 17 9 29 76 13	1
HA	Hatco	100	14	3	34 33 33	2, 4
FI2	Fisher's Iris	150	4	**2**	60 90	1, 3
FI3	Fisher's Iris	150	4	**3**	50 50 50	1, 2
PA	Parkinson's	195	22	2	147 48	1
PR	Planning Relax	182	12	2	130 52	1
RU	Ruspini	75	2	4	18 19 19 19	5, 2
SE	Seeds	210	7	3	70 70 70	1, 2
SY	Soybean	47	35	4	11 12 12 12	1, 2, 6
TH	Thyroid Disease	215	5	3	72 71 72	1, 2, 7
UL	Urban Land Cover	168	147	9	14 16 14 25 15 23 17 15 29	1
WNO	Wine	178	13	3	**60 40 78**	1, 3
WNB	Wine	178	13	3	**59 60 59**	1, 2

Légende pour les notes :

1. Tiré de UCI Machine Learning Repository [46].
2. Instance avec classes balancées.
3. Modification aléatoire des caractéristiques d'un même exemplaire (cf. tableau pour le détail des aspects modifiés).
4. Tiré de *Multivariate Data Analysis* (livre, éditions Pearson) [47].
5. Tiré de *Numerical methods for fuzzy clustering* (article, Information Sciences) [48].
6. Multiples versions disponibles, ici version *Small*.
7. Multiples versions disponibles, ici version avec 215 observations.

6.3 Détails techniques

6.3.1 Implantation des algorithmes

Nouveaux algorithmes développés Tous les algorithmes développés dans le cadre de ce travail sont implantés en C++ en utilisant IBM ILOG CPLEX Optimization Studio 12.9.0, une suite d'optimisation avec plusieurs interfaces (dont une en C++).

Le problème ccMSSC est modélisé en utilisant Concert Technology for C++, une couche de modélisation en C++ faisant partie d'IBM ILOG CPLEX Optimization Studio. Le modèle ainsi créé est alors résolu à travers CP Optimizer, un moteur de résolution CP faisant partie de la même suite logicielle.

La nouvelle contrainte ainsi que la stratégie de recherche développées dans ce travail sont implantées en utilisant CP Optimizer Extensions en C++, après quoi on a accès à elles à travers Concert Technology en utilisant le même modèle.

Le problème MCF intervenant dans la version avancée de la nouvelle contrainte est lui aussi modélisé en utilisant Concert Technology for C++. On utilise CPLEX Optimizer, un solveur inclus dans IBM ILOG CPLEX Optimization Studio, pour le résoudre en utilisant Network Simplex.

CPC La contrainte figurant dans CPC a été réimplantée en utilisant CP Optimizer Extensions en C++ et utilisée dans le même modèle que les nouvelles contraintes.

CP RBBA On a eu accès au code source C++ de CP RBBA [1]. Ce dernier a été développé sur GECODE, un autre moteur de résolution CP libre et open-source très utilisé par la communauté CP. De ce fait, on a dû utiliser un nouveau modèle séparé du reste pour tester cet algorithme (GECODE et la suite d'IBM étant incompatibles). Le code retrouvé a été très légèrement modifié pour y incorporer les contraintes de cardinalité nécessaires. Malgré la différence du moteur de résolution, on ne s'attend pas à ce que cela ait un impact sur les performances.

6.3.2 Lancement des algorithmes

Tous les algorithmes ont été compilés en utilisant Intel C++ Compiler 19.0.3.199 et lancés sur un seul coeur d'un processeur Intel Xeon Gold 6148 @ 2.40 GHz. Tous les processus se sont faits allouer un maximum de 1 GiB de mémoire vive. Le temps maximal d'exécution permis a été fixé à 86400 secondes (i.e., 1 jour).

6.4 Premier axe : Heuristique de recherche

La première des deux contributions de ce travail est une heuristique de recherche rehaussée pour le MSSC dans toutes ses formes.

1. https://cp4clustering.github.io/

6.4.1 Bris d'égalité dynamique

Afin de démontrer les gains du bris d'égalité, on propose de faire un partitionnement simple avec CPC, sans contrôle de cardinalité, sur quelques-uns des exemplaires présentés à la section 6.2. On compare ici le cas avec et sans bris d'égalités. Les hypothèses émises au chapitre 4 stipulent qu'il devrait y avoir un gain de performances généralisé en brisant les égalités.

Afin de tester l'impact de cette stratégie, on exécute CPC tel que présenté à la section 3.3.1 avec six des exemplaires au tableau 6.1. Il a fallu utiliser des exemplaires assez simples compte tenu des performances restreintes de CPC.

Aucun contrôle de la cardinalité n'est introduit dans cette section pour isoler l'effet du bris d'égalités sur les performances. On n'exécute pas cette stratégie avec CP RBBA car ce dernier emploie une recherche fondamentalement différente, axée sur RBBA.

Tableau 6.2 Impact du bris d'égalités sur les performances de CPC

	Aucun bris d'égalité			Bris d'ég. sur la somme des carrés		
Exmp.	Durée [s]	Échecs	Branches	Durée [s]	Échecs	Branches
BC	61.2	8.30k	16.60k	**25.6**	**3.89k**	**7.77k**
BT	—	—	—	**54.6**	**7.26k**	**14.51k**
HA	224.3	40.34k	80.70k	**183.5**	**33.38k**	**66.79k**
FI3	624.5	56.27k	112.54k	**420.8**	**40.96k**	**81.91k**
RU	0.4	83	166	**0.8**	**82**	**168**
SY	1.7	1.66k	3.31k	**1.5**	**1.38k**	**2.77k**

Comment lire ce tableau Les exemplaires au tableau 6.2 sont identifiés par le code qui leur a été assigné au tableau 6.1. Pour chacune des configurations (avec et sans bris d'égalités), on rapporte la durée d'exécution en secondes ainsi que la taille de l'arbre de recherche à travers deux quantités : le nombre d'échecs (*fails*) ainsi que le nombre de branches. Une branche représente une décision prise. Un échec représente un noeud de l'arbre à partir duquel il n'est plus possible de retrouver une solution. Quand il y a échec, un retour en arrière au plus proche noeud valide se produit pour poursuivre la recherche. Par ailleurs, les lignes dans lesquelles il n'y a pas d'information représentent une exécution qui a été interrompue après atteinte du temps limite. Enfin, les lignes en vert et en gras représentent le meilleur résultat pour un exemplaire donné.

Commentaire Le tableau 6.2 est clair : on observe un impact conséquent sur les performances de résolution en introduisant la stratégie de bris d'égalités basée sur la somme des carrés des observations libres. Un exemplaire en particulier, Breast Tissue, ne peut être résolu dans le temps alloué qu'avec le bris d'égalités en place, ramenant le temps de résolution

de plus de 24 heures à moins d'une minute. Par contraste à cela, Ruspini n'a presque pas bénéficié du bris d'égalité. Pour cet exemplaire, on ne voit qu'une réduction non-significative d'un seul échec. En moyenne, pour les exemplaires résolus, on observe des arbres environ 32% plus petits en appliquant le bris d'égalités. Si on raisonne sur la médiane des arbres, celle-ci est réduite d'environ 33%. Les temps de résolution suivent une tendance similaire avec des réductions respectives de 34% et de 37%.

6.4.2 Initialisation heuristique

Il est question de deux ordres de branchements différents : *Distances décroissantes au centre* et *distances minimales décroissantes aux autres centres*. Étant donné qu'on ne sait pas a priori laquelle de ces méthodes est la meilleure, on propose de procéder à la résolution de ccMSSC, avec les algorithmes développés, en utilisant les deux méthodes et voir laquelle se distingue.

Cette étude est réalisée en bas, en même temps que les tests relatifs à la résolution du ccMSSC par les deux versions de la contrainte globale développée.

6.5 Deuxième axe : Contrainte globale pour ccMSSC

La seconde contribution de ce travail prend la forme de deux versions d'une contrainte globale en CP pour la résolution du ccMSSC. La première version utilise des calculs simples et rapides mais produit des bornes moins rigoureuses. La seconde utilise des calculs plus puissants résultants en des bornes de meilleure qualité. Cependant, celle-ci souffre d'une complexité de calcul plus grande.

On ne sait pas dans quel cas le compromis qualité des bornes contre rapidité est le meilleur. On ne sait pas non plus si celui-ci dépend des exemplaires résolus. Pour cela, on propose de résoudre tous les exemplaires discutés à la section 6.2 par les deux versions, en utilisant les deux initialisations heuristiques (voir section 6.4.2 plus haut).

Comme expliqué à la section 6.1, il est nécessaire de comparer ce travail à l'état de l'art en CP pour ce qui concerne le ccMSSC. Pour cela, on lance CPC et CP RBBA munis de contraintes de cardinalité et on compare ces deux algorithmes augmentés aux deux versions de la contrainte développée au chapitre 5. On avait émis l'hypothèse que les méthodes spécialisées développées dans ce travail de recherche devraient être largement meilleures que les méthodes générales existantes.

On n'utilise pas la stratégie de recherche rehaussée (chapitre 4) avec CP RBBA. En effet, la structure de celui-ci ne permet pas de l'y introduire (cela étant dû à la division du problème en plusieurs sous-problèmes spécialisés). Pour ce dernier algorithme, on fait usage des meilleures recommandations émises par les auteurs de celui-ci quant à la stratégie de recherche ainsi qu'à l'ordre des points (on rappelle que RBBA est sensible à l'ordre des observations, voir section 3.4.1). Deux ordres des points distincts sont alors retenus pour CP RBBA : *Farthest First* et *Nearest Neighbor* (voir section 3.4.1). Les deux doivent être explorés.

Par ailleurs, les solutions heuristiques du ccMSSC utilisées pour l'initialisation de la recherche, dans le cas de CPC et des nouveaux algorithmes, sont obtenus à travers deux algorithmes distincts, selon le cas :

— balancé : on utilise LIMA-VNS (voir section 3.2.2) pour lequel on a accès à un exécutable ;
— non balancé : on utilise Constrained K-Means Clustering [49] à travers une implantation modifiée de Babaki, B. [2].

Ces algorithmes génèrent une solution en moins d'une seconde pour les exemplaires considérés. On alimente leur générateur de nombres aléatoires par `/dev/random` et on sélectionne, pour chaque exemplaire, la solution à la médiane parmi 10 solutions indépendantes (i.e., la 5e meilleure).

2. `https://github.com/Behrouz-Babaki/MinSizeKmeans`

Tableau 6.3 Performances de résolution du ccMSSC à l'optimalité par tous les algorithmes

	CP RBBA & GCC				CPC & GCC				Standard CC			Network CC		
Exmp.	Ord.	Durée [s]	Échecs	Branches	Ord.	Durée [s]	Échecs	Branches	Durée [s]	Échecs	Branches	Durée [s]	Échecs	Branches
AI3	NN	—	—	—	OC	—	—	—	—	—	—	—	—	—
	FF	—	—	—	EC	—	—	—	—	—	—	—	—	—
AI4	NN	—	—	—	OC	—	—	—	—	—	—	—	—	—
	FF	—	—	—	EC	—	—	—	—	—	—	—	—	—
BC	NN	—	—	—	OC	—	—	—	42.6	29.22k	58.45k	34.7	6.82k	13.64k
	FF	—	—	—	EC	—	—	—	12.2	5.52k	11.05k	**5.4**	**787**	**1.57k**
BT	NN	—	—	—	OC	—	—	—	23.0	12.98k	25.96k	**9.1**	**886**	**1.77k**
	FF	—	—	—	EC	—	—	—	—	—	—	—	—	—
CB	NN	—	—	—	OC	—	—	—	—	—	—	—	—	—
	FF	—	—	—	EC	—	—	—	—	—	—	—	—	—
CS	NN	—	—	—	OC	—	—	—	—	—	—	—	—	—
	FF	—	—	—	EC	—	—	—	—	—	—	—	—	—
GI	NN	—	—	—	OC	—	—	—	—	—	—	—	—	—
	FF	—	—	—	EC	—	—	—	—	—	—	—	—	—
HA	NN	877.1	195.78k	391.65k	OC	60.1	14.46k	28.92k	6.0	2.58k	5.16k	4.0	559	1.12k
	FF	2.0	161.82k	323.81k	EC	46.0	11.31k	22.63k	3.7	1.72k	3.45k	**1.4**	**127**	**254**
FI2	NN	24.7	19.11k	38.32k	OC	380.3	43.51k	87.02k	10.8	2.53k	5.07k	9.4	1.02k	2.03k
	FF	2.0	91.56k	183.24k	EC	23.9	6.17k	12.33k	3.2	706	1.41k	**1.5**	**142**	**284**
FI3	NN	18.9	1.30M	2.60M	OC	2498.7	207.59k	415.18k	321.6	58.07k	116.13k	147.6	9.94k	19.88k
	FF	—	—	—	EC	350.6	33.49k	66.98k	54.3	11.93k	23.85k	**7.8**	**530**	**1.06k**
PA	NN	—	—	—	OC	—	—	—	—	—	—	27022.1	1.74M	3.49M
	FF	—	—	—	EC	—	—	—	—	—	—	**24867.4**	**1.66M**	**3.32M**
PR	NN	—	—	—	OC	—	—	—	—	—	—	—	—	—
	FF	—	—	—	EC	—	—	—	—	—	—	—	—	—
RU	NN	*1056.9*	*103.27M*	*206.54M*	OC	5290.6	3.52M	7.04M	157.5	167.28k	334.55k	107.1	16.58k	33.16k
	FF	*5192.1*	*586.64M*	*1.17G*	EC	3341.4	2.21M	4.41M	38.8	50.73k	101.47k	**9.3**	**1.74k**	**3.48k**
SE	NN	—	—	—	OC	—	—	—	—	—	—	—	—	—
	FF	—	—	—	EC	—	—	—	—	—	—	**37197.0**	**1.69M**	**3.38M**
SY	NN	*13543.4*	*2.51G*	*5.02G*	OC	442.3	766.06k	1.53M	6.0	16.43k	32.86k	13.1	3.54k	7.07k
	FF	*58074.8*	*11.67G*	*23.34G*	EC	99.7	283.89k	567.77k	**0.9**	**2.84k**	**5.68k**	1.8	643	1.29k
TH	NN	—	—	—	OC	—	—	—	—	—	—	—	—	—
	FF	—	—	—	EC	—	—	—	—	—	—	—	—	—
UL	NN	—	—	—	OC	—	—	—	—	—	—	—	—	—
	FF	—	—	—	EC	—	—	—	—	—	—	—	—	—
WNO	NN	—	—	—	OC	—	—	—	32239.3	6.60M	13.19M	**6420.8**	**567.35k**	**1.13M**
	FF	—	—	—	EC	—	—	—	58373.0	9.28M	18.55M	11335.3	755.86k	1.51M
WNB	NN	—	—	—	OC	—	—	—	497.7	154.16k	308.31k	240.5	31.71k	63.43k
	FF	—	—	—	EC	—	—	—	95.4	15.69k	31.38k	**13.7**	**853**	**1.71k**

Comment lire ce tableau Ce tableau 6.3 se lit comme le tableau 6.2 avec certaines distinctions majeures. CP RBBA & GCC et CPC & GCC représentent les algorithmes existants augmentés de contraintes de cardinalité pour la résolution du ccMSSC. Ces deux algorithmes représentent l'état de l'art en CP. Standard CC représente le premier algorithme de filtrage standard développé dans le présent travail. Network CC représente quant-à-lui le deuxième algorithme avancé. On sépare, d'une part, CP RBBA et, de l'autre, le reste des algorithmes. Cette séparation rend compte de la différence fondamentale de CP RBBA par rapport aux

autres méthodes, notamment en ce qui concerne l'ordre des observations (colonnes *Ord.* sur le tableau). Pour CP RBBA, `NN` représente *Nearest Neighbor* alors que `FF` représente *Farthest First*. Pour le reste des algorithmes, `OC` représente *Distances décroissantes au centre* alors que `EC` représente *Distances minimales décroissantes aux autres centres*. Enfin, les résultats en rouge et en italique représentent une solution erronée.

Commentaire En général, Network CC présente les meilleures performances, malgré les calculs plus lourds, avec CP RBBA montrant, curieusement et souvent, les pires.

- Aucun de Acute Inflammations, Connectionist Bench, Concrete Slump Test, Glass Identification, Planning Relax, Thyroid Disease et Urban Land Cover n'a pu être résolu à optimalité par n'importe laquelle des méthodes.
- Pour tous les autres exemplaires, Network CC présente les meilleures performances sauf pour Soybean où Standard CC est meilleur, mais de peu.
- Dans l'ensemble, *Distances minimales décroissantes aux autres centres* se distingue comme le meilleur ordre des observations à choisir, menant à des performances plus élevées pour CPC & GCC, Standard CC et Network CC pour quasiment tous les exemplaires sauf Wine (version non balancé) et Breast Tissue. Pour ce dernier, seul *Distances décroissantes au centre* mène à une solution.
- Standard CC arrive toute de même à résoudre les mêmes exemplaires que Network CC sauf deux (Parkinson's et Seeds).
- CPC & GCC présente de meilleures performances pour ccMSSC que CP RBBA malgré les bornes moins bonnes. En contre partie, CP RBBA se distingue grandement pour Hatco et Fisher's Iris (version 2 classes) pour lesquels il a des performances qui sont très proches de Network CC.
- Pour les exemplaires résolus par les trois algorithmes n'incluant pas CP RBBA, Network CC produit l'arbre de recherche le plus petit, suivi, dans l'ordre, de Standard CC puis de CPC & GCC. Standard CC produit des arbres en moyenne 9 fois plus grands. CPC & GCC produit quant-à-lui des arbres plus de 200 fois plus grands.
- CP RBBA maintient des arbres significativement plus grands à cause de la résolution de $n-k$ problèmes à chaque fois. Aussi, CP RBBA ne peut retourner de solution intermédiaire car les sous-problèmes sont naturellement incomplets. Ceci est par contraste aux autres méthodes qui montrent toutes les solutions améliorantes entre le début et la fin de la recherche.
- Pour CP RBBA, il ne semble pas y avoir un ordre de points à privilégier. Tantôt *Farthest First* produit les meilleurs résultats, tantôt *Nearest Neighbor* le fait à son tour.

— Six des 19 exemplaires ne peuvent être résolus qu'avec les nouvelles méthodes. Deux d'entre eux (Parkinson's et Seeds) ne peuvent être résolus qu'avec Network CC.
— Il ne semble y avoir qu'une très faible corrélation entre les caractéristiques n, s et k des exemplaires et leur difficulté. Par exemple, Acute Inflammations avec $n = 120, s = 6, k = 3$ ne peut être résolu par aucun des algorithmes alors que Wine $n = 178, s = 13, k = 3$ peut être résolu par Standard CC et Network CC sans problème.
— En moyenne sur les exemplaires résolus, CP Optimizer est capable de prendre 250 décisions par seconde (branches par secondes) avec Network CC. Ceci est par opposition à 1476 pour Standard CC, soit presque 6 fois plus pour ce dernier.

Autres remarques En examinant les résultats de plus près, on remarque que :
— pour tous sauf un des exemplaires résolus, les solutions de départ générées par LIMA-VNS et par Constrained K-Means Clustering se trouvent être les solutions optimales pour leurs exemplaires respectifs. Dans ce contexte, les algorithmes de CP présentés ici servent à prouver l'optimalité. Seul Parkinson's a démarré d'une solution non optimale. Pour ce dernier, on réfère le lecteur au graphique à la figure 6.1 pour l'évolution de l'objectif au cours de la recherche ;
— pour les exemplaires qui ne peuvent pas être résolus, aucun des algorithmes n'améliore la solution initiale donnée par les heuristiques ;
— CP RBBA retourne de mauvaises solutions pour Ruspini et Soybean. On pense que cela est probablement dû à une accumulation d'instabilités de calculs.

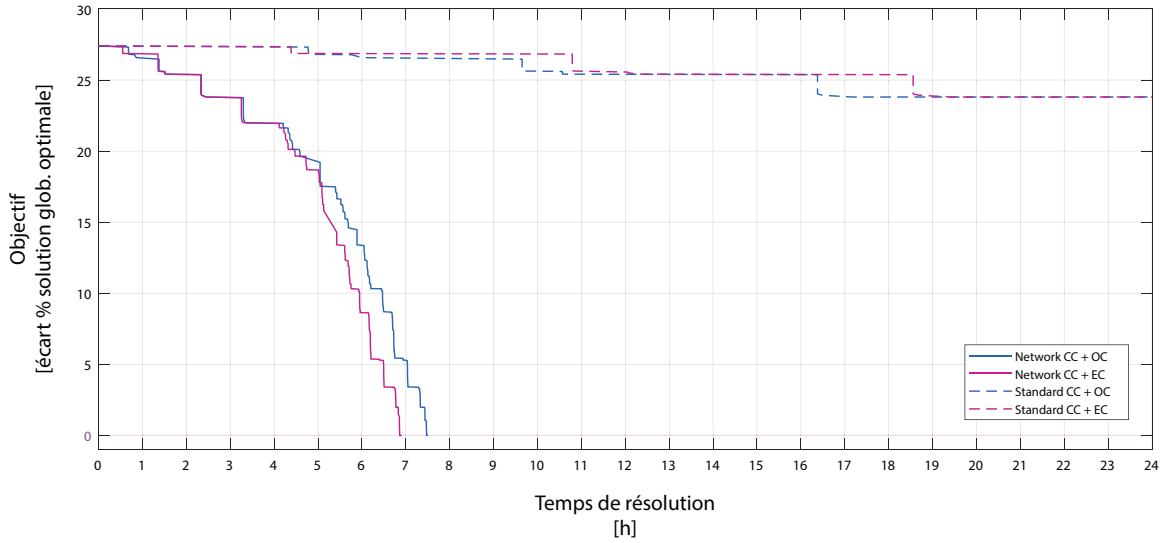

Figure 6.1 Évolution de l'objectif au cours de la recherche d'une solution pour Parkinson's

La figure 6.1 montre l'évolution de l'écart par rapport à la solution optimale pour Standard CC et Network CC. En général, les deux suivent le même chemin. Par contre, Network CC est capable d'avancer plus rapidement. Au final, Network CC prouve l'optimalité en moins de 7 heures alors que Standard CC reste bloqué à 24% de l'objectif jusqu'à la fin du temps permis.

On peut voir aussi dans le graphique l'impact du meilleur ordre des points sur les performances de façon plus claire. En effet, pour Network CC, à partir de 5 heures, *Distances minimales décroissantes aux autres centres* permet de retrouver des solutions améliorantes plus vite, ramenant la recherche de plus en plus près de la solution optimale plus tôt.

Par ailleurs, pour les expérimentations qui n'aboutissent pas à un optimum global, on présente le tableau 6.4 qui résume les écarts observés entre la meilleure solution trouvée et la borne inférieure de la branche incomplète la plus prometteuse du Branch and Bound.

Tableau 6.4 Écarts observés dans la résolution du ccMSSC par tous les algorithmes

	Ordre OC [écart %]			Ordre EC [écart %]		
Exmp.	CPC & GCC	Standard CC	Network CC	CPC & GCC	Standard CC	Network CC
AI3	53.2	53.2	38.0	55.4	55.4	**37.7**
AI4	57.9	57.8	30.0	55.7	55.7	**29.6**
BC	81.6	—	—	81.6	—	—
BT	99.9	—	—	99.9	4.0	3.7
CB	36.2	35.9	17.4	35.9	35.6	**17.3**
CS	45.7	45.7	**28.0**	45.5	45.5	28.1
GI	96.5	86.4	**55.2**	96.5	90.3	58.9
HA	—	—	—	—	—	—
FI2	—	—	—	—	—	—
FI3	—	—	—	—	—	—
PA	79.6	68.0	—	79.6	68.0	—
PR	49.3	40.4	**25.2**	49.3	40.4	**25.2**
RU	—	—	—	—	—	—
SE	47.7	46.8	24.2	47.7	46.8	—
SY	—	—	—	—	—	—
TH	87.4	83.3	**37.8**	87.4	83.3	**37.8**
UL	92.2	60.8	**30.3**	92.2	60.8	**30.3**
WNO	70.6	—	—	70.6	—	—
WNB	69.4	—	—	69.4	—	—

Comment lire ce tableau Toutes les valeurs du tableau 6.4 représentent, en pourcentage et pour chaque configuration, l'écart entre le coût de la meilleure solution trouvée (i.e., le plus petit UB(Z)) et la borne inférieure de la branche incomplète la plus prometteuse (i.e., le plus petit LB(Z) relatif à une solution partielle d'un noeud de l'arbre qui n'as pas été exploré à terme). Les valeurs absentes représentent un écart de 0%, ce qui revient à dire que la solution globalement optimale a été trouvée pour la configuration correspondante (et rapportée au

tableau 6.3). Les valeurs en vert et en gras représentent le meilleur écart calculé pour un exemplaire qui n'a pu être résolu par aucune des méthodes. Enfin, pour les mêmes raisons rapportées avant, CP RBBA ne fait pas partie de cette analyse.

Commentaire Le tableau 6.4 montre des écarts significativement meilleurs pour Network CC en ce qui concerne la totalité des exemplaires. Celui-ci est suivi de Standard CC et ensuite de CPC & GCC. Dans l'ensemble, les écarts sont assez élevés pour les exemplaires non résolus avec en moyenne 31% pour Network CC, 55% pour Standard CC et 71% pour CPC & GCC. De plus, CPC & GCC montre parfois des performances qui se démarquent par leur qualité extrêmement mauvaise. On attire l'attention par exemple à Breast Tissue pour lequel on a un écart de presque 100%, voulant dire que le mécanisme de filtrage a un impact essentiellement inexistant (alors que les deux autres méthodes présentent des écarts très largement meilleurs d'environ 4% quand ils ne réussissent pas à prouver l'optimalité de cet exemplaire). Enfin, aucun de `OC` et `EC` ne se démarque comme le meilleur ordre des points si on ne se fie qu'aux écarts observés.

6.6 Stratégie de recherche rehaussée vs stratégie par défaut de CP Optimizer

Les lecteurs familiers avec IBM CP Optimizer (CPO), le moteur de résolution CP utilisé dans l'implantation de ce travail, savent que celui-ci est muni d'une des stratégies de branchements automatiques les plus performantes en CP.

Celle-ci prend la forme d'une recherche en profondeur (Depth-First Search, DFS) dirigée par les échecs (Failure-directed Search, FDS) à l'aide de plusieurs heuristiques [50]. Elle représente en fait une recherche alternative qui vient après une *Large Neighborhood Search* (ou LNS) [51]. De ce fait, l'idée derrière cette stratégie est qu'une meilleure solution n'existe pas ou est très dure à trouver. Cette supposition est au coeur de son fonctionnement dans le sens où elle priorise non pas l'amélioration de la solution mais plutôt l'élimination rapide du plus grand nombre de branches infructueuses [51]. Par défaut, CP Optimizer redémarre cette DFS après un certain nombre d'échecs pour échapper aux sous-arbres de recherche infructueux plus vite.

Afin de montrer où se situe la stratégie personnalisée développée dans ce travail par rapport à celle de CPO, on teste cette dernière avec Network CC pour la résolution du ccMSSC de six exemplaires.

Tableau 6.5 Comparaison de la stratégie personnalisée rehaussée avec celle de CPO pour la résolution du ccMSSC avec Network CC

	Stratégie personnalisée rehaussée			Stratégie CPO auto, DFS + relances		
Exmp.	Durée [s]	Échecs	Branches	Durée [s]	Échecs	Branches
BC	**5.4**	**787**	**1.57k**	181.3	18.38k	41.28k
BT	**9.1**	**886**	**1.77k**	140.3	5.03k	11.67k
HA	**1.4**	**127**	**254**	121.6	13.67k	32.05k
FI3	**7.8**	**530**	**1.06k**	589.3	51.50k	112.65k
RU	**9.3**	**1.74k**	**3.48k**	107.8	13.68k	30.37k
SY	**1.8**	**643**	**1.29k**	50.1	13.12k	27.21k

Comment lire ce tableau Ce tableau 6.5 est identique à 6.2 en ce qui concerne sa lecture. Les valeurs relatives à la stratégie personnalisée ont été reprises des meilleurs résultats rapportés au tableau 6.3 pour ce qui concerne Network CC afin de servir de référence.

Commentaire La stratégie personnalisée rehaussée discutée dans ce travail présente un avantage substantiel par rapport à celle de CPO où les arbres de recherche sont en moyenne 27 fois plus grands et les temps de résolution environ 20 fois plus longs. La médiane est augmentée d'un facteur de 21 et de 34, respectivement, pour les deux quantités.

6.7 Comparaison avec la méthode d'optimisation conique

On parle dans cette partie de la méthode discutée à la section 3.2.3.

Avant de continuer, il est impératif de rappeler que cette méthode ne garantit pas l'optimalité de la solution obtenue. Elle permet par contre d'émettre des garanties sous forme de bornes sur les solutions retrouvées. La méthode par CP discutée dans ce travail permet aussi de fournir des garanties à travers des bornes inférieures, mais celle-ci ne s'achève qu'avec une preuve d'optimalité globale, contrairement à la méthode basée sur l'optimisation conique qui peut s'achever sans elle. De plus, cette dernière méthode ne permet pas d'incorporer des contraintes utilisateur supplémentaires autres que celles relatives aux cardinalités finales. Dans notre cas, ajouter des contraintes supplémentaires est extrêmement facile en utilisant le paradigme déclaratif de la CP (voir chapitre 1) en conjonction avec une heuristique de départ adéquate (voir section 4.1.3). Enfin, celle-ci a été testée sur un ordinateur différent (Multi-core Intel Core i7 @ 3.40 GHz et 16 GiB de mémoire vive).

Néanmoins, les auteurs arrivent à résoudre `FI3`, `SE`, `PR` et `PA` avec preuve d'optimalité en 584, 3823, 2637 et 2000 secondes respectivement [17]. Ceci représente des performances bien

meilleures que celles atteintes dans ce travail sauf pour FI3 où Network CC se démarque très largement.

Par exemple, on n'arrive pas à résoudre PR par n'importe laquelle des méthodes développées dans ce travail. En contrepartie, la méthode basée sur l'optimisation conique permet de fournir une solution à cet exemplaire, avec preuve d'optimalité, en un peu moins de 45 minutes. Celle-ci permet aussi de donner une solution à CB et à UL avec un écart de 0.001% et de 3%, respectivement, alors que le meilleur que Network CC peut faire est de 17% et 30%, respectivement.

6.8 Analyse et interprétation des résultats

6.8.1 Bris des égalités

Le bris des égalités dont il est question ici favorise l'échec. On rappelle qu'il y est question de toujours démarrer les nouveaux groupes à partir des points dont la somme des carrés avec les autres points libres est la plus grande. Or, une solution de coût moindre généralement requiert le contraire, son coût étant directement relié aux distances carrées entre les points. De ce fait, cette stratégie représente une sorte de fail-first. Les lecteurs attentifs se rappelleront de ce principe discuté à la section 4.1 dont les résultats sont des arbres plus petits dans lesquels les branches infructueuses sont ramenées à la racine de ceux-là.

Les statistiques observées viennent appuyer cette analyse avec des résultats universellement meilleurs quand le bris d'égalité est actif.

6.8.2 Méthodes actuelles vs nouvelles méthodes

On avait émis l'hypothèse que les nouvelles méthodes développées allaient présenter des performances meilleures pour la résolution du ccMSSC, résultat de leur spécialisation au problème avec contraintes de cardinalité. En effet, on a montré deux méthodes pour faire valoir ces contraintes pour le calcul de meilleures bornes, Standard CC et Network CC. De plus, on a aussi proposé une stratégie de recherche mise à jour pour le problème.

De meilleures bornes permettent d'identifier les branches infructueuses plus tôt et permettent de réduire l'espace de recherche pour une résolution efficace. Les statistiques sur la taille des arbres de recherche relatifs à chacune des méthodes le démontrent. De plus, l'effet des meilleures bornes est directement mesurable à travers les écarts calculés et rapportés au tableau 6.4. Au final, celles-ci ont permis la résolution de plus d'exemplaires dans un temps de plus en plus réduit.

CP RBBA est sensé prendre en compte les contraintes externes dans le calcul de ses bornes [10]. Vraisemblablement, cela n'a pas été suffisant pour lui permettre de se démarquer. Il est possible que les contraintes supplémentaires introduisent une charge de calcul significativement plus grande pour la détermination des bornes. En effet, CP RBBA sans contraintes est considéré comme un des algorithmes les plus rapides en CP pour résoudre le MSSC.

Par ailleurs, dans le cas de CP RBBA, ne pas résoudre un exemplaire dans le temps imparti veut dire qu'on n'a aucune solution (en effet, la solution de tous les sous-problèmes est requise pour cela). Ceci est par contraste aux trois autres méthodes pour lesquelles on a accès à des solutions intermédiaires même si on n'est pas en mesure de pouvoir prouver un optimum global avant la limite de temps imposée.

De toute autre part, cet avantage dû à la spécialisation des algorithmes au cas du ccMSSC s'étend au delà des contraintes, jusqu'à la stratégie de recherche, comme on l'a vu à la section 6.6. En effet, notre stratégie arrive a présenter des gains clairs par rapport à une stratégie générale, même aussi bonne que celle d'IBM CP Optimizer.

Enfin, les écarts observés entre l'approche avec optimisation conique et la nôtre (section 6.7) sont la manifestation des différences fondamentales entre celles-ci. L'une (approche CP) explore l'ensemble de l'espace des solutions et garantit tout le temps une solution optimale quand elle est retournée. L'autre (optimisation conique) trouve une solution à travers des méthodes approximatives et heuristiques et fournit une garantie a posteriori sur celle-ci à travers des bornes. Ces méthodes ne sont alors pas tout à fait équivalentes et une comparaison entre celles-ci n'est pas forcément claire et facile.

6.8.3 Algorithme de filtrage standard vs. algorithme avancé

Network CC présente l'avantage de calculer des bornes encore plus serrées (voir tableau 6.4). Cela vient cependant avec un coût non négligeable (une propagation en moyenne 6 fois plus lente comme le démontrent les résultats). Malgré cela, ces bornes de meilleure qualité permettent de réduire l'espace de recherche par un plus grand facteur, faisant de l'effort supplémentaire de calcul tout sauf un gâchis.

Au final, les meilleures bornes nous permettent de réduire le temps de calcul pour tous sauf les plus simples exemplaires (dans lesquels la différence se mesure en dixièmes de secondes). Network CC est donc facilement le meilleur algorithme.

6.8.4 Ordre d'initialisation heuristique

On aura examiné deux ordres d'initialisation heuristique dans ce travail : *Distances décroissantes au centre* (aussi connu sous le nom de `OC`) et *distances minimales décroissantes aux autres centres* (appelé aussi `EC`).

Dans l'ensemble, `EC` présente les résultats les plus satisfaisants. Cependant, il n'est pas universellement meilleur (on pense par exemple à Breast Tissue et à Wine non balancé ou encore aux écarts observés sur les exemplaires dont on ne prouve pas l'optimalité). Cela laisse entendre soit la présence d'une meilleure méthode qui puisse allier les forces des deux soit une forte dépendance à l'agencement des observations dans l'espace, rendant le choix de la méthode plus difficile.

Dans le premier cas, des investigations futures sont à prévoir. Dans le deuxième cas, il serait nécessaire de développer des procédures pour étudier les ensembles de données a priori afin de pouvoir sélectionner la meilleure stratégie. Malheureusement, les données obtenues ne permettent pas de tirer des conclusions définitives. Néanmoins, il est clair que, dans l'immédiat, `EC` devrait être privilégié compte tenu du nombre d'exemplaires résolus à travers lui.

6.8.5 Qu'est-ce qui rend un exemplaire difficile à résoudre ?

On aura vu qu'il n'y a qu'une très faible corrélation entre, d'une part, le nombre de points, la dimensionnalité et le nombre de classes et, d'autre part, la difficulté à résoudre un exemplaire donné. Il est sûr que ces quantités jouent un rôle, cependant celui-ci ne devrait pas être aussi grand qu'on peut le penser.

Les bornes calculées dépendent des contributions individuelles de chaque point (voir section 5.1). Celles-ci dépendent, à leur tour, des voisins de ces points. Ainsi, il paraît raisonnable d'arriver à la conclusion suivante : la difficulté d'un exemplaire a un rapport plus fort avec la séparation de ses classes. Un exemplaire avec des classes bien séparées permet le calcul de bornes plus fidèles. Au contraire, un exemplaire avec des classes confondues rend le calcul des bornes moins précis à cause de la proximité de toutes les observations avec toutes les autres. Ces exemplaires ont souvent aussi la caractéristique d'avoir un espace des solutions relativement plat : la solution optimale se trouve à proximité de plusieurs autres non optimales de coûts à peu près égaux.

Ce dernier aspect explique aussi pourquoi, pour un même groupe d'observations, changer le nombres de classes et/ou les cardinalités désirées change de façon significative l'effort de calcul à entreprendre. Cela est dû au fait que changer ces caractéristiques modifie la forme et la séparation des classes dans la solution optimale. Par exemple, si un exemplaire comporte 3

classes balancées séparées, le fait de les traiter à la place comme 2 classes balancées viendrait scinder un des groupes originaux en deux, provoquant une proximité des classes et donc une perte de la séparation. On pense par exemple à Wine. Le fait de changer les cardinalités désirées mène à un problème largement différent : `WN0` est à peu près 665 fois plus difficile (si on se fie à la taille de l'arbre) que `WNB`. De plus, pour le premier, c'est `OC` qui mène aux meilleures performances contrairement au second où `EC` se démarque.

CHAPITRE 7 CONCLUSION

On aura proposé dans ce travail une approche de résolution exacte du problème de partitionnement avec minimisation de variance sous contraintes de cardinalité, ccMSSC. Ce dernier étant généralement résolu par des heuristiques vu sa difficulté en pratique, le présent travail montre alors une des rares méthodes exactes pour résoudre ce problème. Pour cela, on fait appel à la programmation par contraintes dont la flexibilité habilite le développement de stratégies efficaces pour la résolution jusqu'à un optimum global.

7.1 Synthèse des contributions

7.1.1 Une stratégie de branchement rehaussée

La première contribution de ce travail prend la forme d'une stratégie de branchement en CP visant la résolution efficace du ccMSSC. Celle-ci a été inspirée d'un travail antérieur et représente une version renforcée de celui-ci.

Premièrement, on a identifié un bris d'égalité qui permet de réduire la taille des arbres de recherche en assurant un démarrage des groupes plus futé lors du processus de partitionnement. Les résultats expérimentaux montrent un clair avantage du bris d'égalité sur l'ensemble des expériences faites.

Deuxièmement, on a proposé deux initialisations gloutonnes de la recherche qui font valoir les multiples heuristiques déjà en place dans la littérature. Celles-ci sont conçues encore une fois pour réduire la taille des arbres de recherche en contrôlant, à la fois, la borne supérieure initiale de la solution mais aussi l'agencement des branches de recherche. Là encore, les résultats expérimentaux viennent appuyer l'efficacité des méthodes prescrites quand on les confronte à l'une des meilleures stratégies de recherche disponibles sur le marché.

7.1.2 Deux algorithmes de filtrage pour ccMSSC

La deuxième contribution de ce travail représente une contrainte globale pour la réduction de l'espace de recherche d'une solution optimale pour le ccMSSC. On propose deux versions de celle-ci avec deux algorithmes de filtrage distincts.

Un premier algorithme de filtrage proposé représente une simple adaptation d'un algorithme général antérieur pour traiter le cas particulier du ccMSSC. Cet algorithme a été modifié

pour prendre en compte les caractéristiques du ccMSSC dans les calculs afin de tirer plein profit de la structure du problème en main pour la réduction des domaines des variables.

Un deuxième algorithme de filtrage proposé représente, quant à lui, une reconception plus approfondie qui fait appel à la résolution d'un problème de flot pour des calculs plus fidèles et donc une réduction de l'espace de recherche plus puissante. Ceci induit cependant une charge de calcul considérable.

Les expérimentations faites viennent confirmer l'efficacité des deux algorithmes développés par rapport à d'autres approches en CP. En particulier, on a remarqué que l'approche avancée impliquant la résolution du problème de flot était globalement meilleure malgré la charge de calcul supplémentaire car elle provoque une plus grande réduction des domaines.

Enfin, on ne connaît qu'une seule autre méthode destinée explicitement à la résolution du ccMSSC et capable de produire, parfois, une solution globalement optimale (section 3.2.3). Alors qu'on peut résoudre quelques exemplaires plus simples beaucoup plus rapidement, cette méthode rivale représente une meilleure alternative à la nôtre pour les exemplaires beaucoup plus compliqués ou pour les cas où on n'a besoin que d'une assez bonne garantie sur la solution retrouvée.

7.2 Limites et avenues de recherche futures

7.2.1 Stratégie de recherche

On rappelle qu'on propose deux initialisations gloutonnes de la recherche, sous forme de deux ordres des points initiaux : *Distances décroissantes au centre* et *distances minimales décroissantes aux autres centres* (respectivement dénommés `OC` et `EC`).

Or, on aura vu qu'aucune de ces deux méthodes ne se démarque suffisamment pour la recommander pour tous les exemplaires, signifiant la présence de caractéristiques cachées qui ne sont pas prises en compte dans la présente méthode (voir section 6.8.4). Ce travail ne permet pas de tirer des conclusions suffisantes à cet égard. Ceci représente alors une potentielle avenue d'expansion dans laquelle on développerait soit une meilleure stratégie d'initialisation soit des outils qui permettent de déterminer, avant le début de la résolution, laquelle des méthodes d'initialisation est la plus adéquate pour l'exemplaire courant.

Par ailleurs, la stratégie de recherche développée n'incorpore aucun mécanisme de relance. Elle représente une simple DFS guidée. Celle-ci a de la difficulté à échapper aux mauvaises décisions et se perd dans des sous-arbres infructueux larges [50]. Or, il serait possible de

remédier à cela en procédant à des relances périodiques lors desquels la recherche est reprise et guidée vers des endroits plus susceptibles de contenir des solutions [50].

Finalement, il pourrait être judicieux d'explorer la résolution du présent problème de ccMSSC dans un cadre de résolution par CP utilisant le dénombrement des solutions comme stipulé dans [52].

7.2.2 Généralisation de la contrainte globale de ccMSSC

La contrainte développée dans ce travail assure l'égalité stricte des cardinalités des groupes de la solution à ceux dictés par le problème. Mais qu'en est-il du cas où on tolérerait un certain jeu si cela signifiait un meilleur coût ?

Considérons le problème dans lequel on voudrait contraindre les cardinalités résultantes à avoir une moyenne décidée a priori et un écart-type restreint par une limite supérieure. Ou alors, le problème dans lequel on tolérerait une certaine déviation, relative ou absolue, des cardinalités finales par rapport à celles désirées.

On pourrait venir lier ces dernières caractéristiques à des intervalles de cardinalités permises, similaire à l'utilisation d'une GCC avec coûts, pour chacun des groupes (contrairement à seulement une valeur en particulier, désignée par n_c précédemment). Cette liaison pourrait être assurée par une contrainte globale comme celle de dispersion discutée dans [53]. Une fois ces intervalles définis, mettre à jour le modèle MCF de la contrainte développée dans ce travail est chose facile. Cependant, la difficulté réside dans la nécessité de développer de nouvelles méthodes pour le calcul incrémentiel du Cost-based filtering, compte tenu des changements qui seront apportés au graphe résiduel du MCF (qui ne sera plus un simple graphe biparti). Il faudra aussi redéfinir le calcul des contributions minimales des points vu que, maintenant, on ne connaît pas le nombre exact d'observations dans chaque groupe. En effet, ces contributions dépendent du nombre final des points d'un groupe, lequel reste inconnu jusqu'à la fin de la recherche.

RÉFÉRENCES

[1] P.-N. Tan *et al.*, *Introduction to Data Mining*, 2e éd. New York, NY : Pearson, 2019.

[2] T.-B.-H. Dao, K.-C. Duong et C. Vrain, "Constrained clustering by constraint programming," *Artificial Intelligence*, vol. 244, p. 70 – 94, 2017, combining Constraint Solving with Mining and Learning. [En ligne]. Disponible : http://www.sciencedirect.com/science/article/pii/S0004370215000806

[3] M. F. A. Hady et F. Schwenker, *Semi-supervised Learning*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2013, p. 215–239. [En ligne]. Disponible : https://doi.org/10.1007/978-3-642-36657-4_7

[4] K. Apt, *Constraint satisfaction problems : examples*. Cambridge University Press, 2003, p. 8–53.

[5] T.-B.-H. Dao, K.-C. Duong et C. Vrain, "Constrained minimum sum of squares clustering by constraint programming," dans *Principles and Practice of Constraint Programming*, G. Pesant, édit. Cham : Springer International Publishing, 2015, p. 557–573.

[6] F. Rossi, P. van Beek et T. Walsh, *Handbook of Constraint Programming*, 1er éd. Amsterdam, The Netherlands : Elsevier, 2006. [En ligne]. Disponible : https://www.elsevier.com/books/handbook-of-constraint-programming/rossi/978-0-444-52726-4

[7] K. Apt, *Constraint programming in a nutshell*. Cambridge University Press, 2003, p. 54–81.

[8] ——, *Search*. Cambridge University Press, 2003, p. 299–350.

[9] ——, *Constraint propagation algorithms*. Cambridge University Press, 2003, p. 254–298.

[10] T. Guns *et al.*, "Repetitive branch-and-bound using constraint programming for constrained minimum sum-of-squares clustering," dans *Proceedings of the Twenty-second European Conference on Artificial Intelligence*, ser. ECAI'16. Amsterdam, The Netherlands, The Netherlands : IOS Press, 2016, p. 462–470. [En ligne]. Disponible : https://doi.org/10.3233/978-1-61499-672-9-462

[11] W. Lin, Z. He et M. Xiao, "Balanced clustering : A uniform model and fast algorithm," dans *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, p. 2987–2993. [En ligne]. Disponible : https://doi.org/10.24963/ijcai.2019/414

[12] D. Aloise *et al.*, “Np-hardness of euclidean sum-of-squares clustering,” *Machine Learning*, vol. 75, n°. 2, p. 245–248, May 2009. [En ligne]. Disponible : https://doi.org/10.1007/s10994-009-5103-0

[13] M. F. Balcan, S. Ehrlich et Y. Liang, “Distributed k-means and k-median clustering on general topologies,” dans *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS’13. USA : Curran Associates Inc., 2013, p. 1995–2003. [En ligne]. Disponible : http://dl.acm.org/citation.cfm?id=2999792.2999835

[14] A. Banerjee et J. Ghosh, “Scalable clustering algorithms with balancing constraints,” *Data Mining and Knowledge Discovery*, vol. 13, n°. 3, p. 365–395, Nov 2006. [En ligne]. Disponible : https://doi.org/10.1007/s10618-006-0040-z

[15] J. Desrosiers, N. Mladenović et D. Villeneuve, “Design of balanced mba student teams,” *Journal of the Operational Research Society*, vol. 56, n°. 1, p. 60–66, 2005. [En ligne]. Disponible : https://doi.org/10.1057/palgrave.jors.2601775

[16] L. Hagen et A. B. Kahng, “New spectral methods for ratio cut partitioning and clustering,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, n°. 9, p. 1074–1085, Sep. 1992.

[17] N. Rujeerapaiboon *et al.*, “Size matters : Cardinality-constrained clustering and outlier detection via conic optimization,” *SIAM Journal on Optimization*, vol. 29, n°. 2, p. 1211–1239, 2019. [En ligne]. Disponible : https://doi.org/10.1137/17M1150670

[18] D. Aloise et P. Hansen, “Evaluating a branch-and-bound rlt-based algorithm for minimum sum-of-squares clustering,” *Journal of Global Optimization*, vol. 49, n°. 3, p. 449–465, Mar 2011. [En ligne]. Disponible : https://doi.org/10.1007/s10898-010-9571-3

[19] IBM, *IBM ILOG CPLEX Optimization Studio : CPLEX User’s Manual*, International Business Machines Corporation.

[20] D. Jungnickel, *The Network Simplex Algorithm*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2005, p. 321–339. [En ligne]. Disponible : https://doi.org/10.1007/3-540-26908-8_11

[21] L. Vandenberghe, “Lecture 17 : Network flow optimization,” University of California, Los Angeles, automne 2013.

[22] C.-G. Quimper *et al.*, “Improved algorithms for the global cardinality constraint,” dans *Principles and Practice of Constraint Programming – CP 2004*, M. Wallace, édit. Berlin, Heidelberg : Springer Berlin Heidelberg, 2004, p. 542–556.

[23] F. Rossi, P. van Beek et T. Walsh, *Handbook of Constraint Programming*, 1er éd. Amsterdam, The Netherlands : Elsevier, 2006. [En ligne]. Disponible : https://www.elsevier.com/books/handbook-of-constraint-programming/rossi/978-0-444-52726-4

[24] T. Walsh, "Breaking value symmetry," dans *Principles and Practice of Constraint Programming – CP 2007*, C. Bessière, édit. Berlin, Heidelberg : Springer Berlin Heidelberg, 2007, p. 880–887.

[25] ——, "Symmetry breaking constraints : Recent results," dans *AAAI Conference on Artificial Intelligence*, 2012. [En ligne]. Disponible : https://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/4974/5393

[26] Y. C. Law et J. H. M. Lee, "Global constraints for integer and set value precedence," dans *Principles and Practice of Constraint Programming – CP 2004*, M. Wallace, édit. Berlin, Heidelberg : Springer Berlin Heidelberg, 2004, p. 362–376.

[27] G. Pesant, "A regular language membership constraint for finite sequences of variables," dans *Principles and Practice of Constraint Programming – CP 2004*, M. Wallace, édit. Berlin, Heidelberg : Springer Berlin Heidelberg, 2004, p. 482–495.

[28] X. Wu *et al.*, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, nº. 1, p. 1–37, Jan 2008. [En ligne]. Disponible : https://doi.org/10.1007/s10115-007-0114-2

[29] X. Jin et J. Han, *K-Means Clustering*. Boston, MA : Springer US, 2010, p. 563–564. [En ligne]. Disponible : https://doi.org/10.1007/978-0-387-30164-8_425

[30] D. Arthur et S. Vassilvitskii, "How slow is the k-means method ?" dans *Proceedings of the Twenty-second Annual Symposium on Computational Geometry*, ser. SCG '06. New York, NY, USA : ACM, 2006, p. 144–153. [En ligne]. Disponible : http://doi.acm.org/10.1145/1137856.1137880

[31] M. J. Brusco, "A repetitive branch-and-bound procedure for minimum within-cluster sums of squares partitioning," *Psychometrika*, vol. 71, nº. 2, p. 347–363, Jun 2006. [En ligne]. Disponible : https://doi.org/10.1007/s11336-004-1218-1

[32] I. Davidson et S. S. Ravi, "Clustering with constraints : Feasibility issues and the k-means algorithm," dans *Proceedings of the 2005 SIAM International Conference on Data Mining*, ser. SIAM '05, 2005, p. 138–149. [En ligne]. Disponible : https://epubs.siam.org/doi/abs/10.1137/1.9781611972757.13

[33] K. Wagstaff *et al.*, "Constrained k-means clustering with background knowledge," dans *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2001, p. 577–584. [En ligne]. Disponible : http://dl.acm.org/citation.cfm?id=645530.655669

[34] A. Pyatkin, D. Aloise et N. Mladenović, "Np-hardness of balanced minimum sum-of-squares clustering," *Pattern Recognition Letters*, vol. 97, p. 44 – 45, 2017. [En ligne]. Disponible : http://www.sciencedirect.com/science/article/pii/S0167865517301988

[35] L. R. Costa, D. Aloise et N. Mladenović, "Less is more : basic variable neighborhood search heuristic for balanced minimum sum-of-squares clustering," *Information Sciences*, vol. 415-416, p. 247 – 253, 2017. [En ligne]. Disponible : http://www.sciencedirect.com/science/article/pii/S0020025517307934

[36] B. Babaki, T. Guns et S. Nijssen, "Constrained clustering using column generation," dans *Integration of AI and OR Techniques in Constraint Programming*, H. Simonis, édit. Cham : Springer International Publishing, 2014, p. 438–454.

[37] F. Focacci, A. Lodi et M. Milano, "Cost-based domain filtering," dans *Principles and Practice of Constraint Programming – CP'99*, J. Jaffar, édit. Berlin, Heidelberg : Springer Berlin Heidelberg, 1999, p. 189–203.

[38] R. A. Carbonneau, G. Caporossi et P. Hansen, "Extensions to the repetitive branch and bound algorithm for globally optimal clusterwise regression," *Computers Operations Research*, vol. 39, n°. 11, p. 2748 – 2762, 2012. [En ligne]. Disponible : http://www.sciencedirect.com/science/article/pii/S0305054812000330

[39] D. Arthur et S. Vassilvitskii, "K-means++ : The advantages of careful seeding," dans *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '07. Philadelphia, PA, USA : Society for Industrial and Applied Mathematics, 2007, p. 1027–1035. [En ligne]. Disponible : http://dl.acm.org/citation.cfm?id=1283383.1283494

[40] J.-C. Régin, "Arc consistency for global cardinality constraints with costs," dans *Principles and Practice of Constraint Programming – CP'99*, J. Jaffar, édit. Berlin, Heidelberg : Springer Berlin Heidelberg, 1999, p. 390–404.

[41] P. Refalo, "Impact-based search strategies for constraint programming," dans *Principles and Practice of Constraint Programming – CP 2004*, M. Wallace, édit. Berlin, Heidelberg : Springer Berlin Heidelberg, 2004, p. 557–571.

[42] L. Michel et P. Van Hentenryck, "Activity-based search for black-box constraint programming solvers," dans *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, N. Beldiceanu, N. Jussien et É. Pinson, édit. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012, p. 228–243.

[43] P. Briggs et L. Torczon, "An efficient representation for sparse sets," *ACM Lett. Program. Lang. Syst.*, vol. 2, n°. 1-4, p. 59–69, mars 1993. [En ligne]. Disponible : http://doi.acm.org/10.1145/176454.176484

[44] Z. Király et P. Kovács, “Efficient implementations of minimum-cost flow algorithms,” *Acta Universitatis Sapientiae, Informatica*, vol. 4, juill. 2012.

[45] R. Bellman, “On a routing problem,” *Quarterly of Applied Mathematics*, vol. 16, n°. 1, p. 87–90, 1958. [En ligne]. Disponible : http://www.jstor.org/stable/43634538

[46] U. of California Irvine. (2019) Uci machine learning repository. [En ligne]. Disponible : https://archive.ics.uci.edu

[47] J. F. Hair *et al.*, *Multivariate Data Analysis*, 5e éd. New York, NY : Pearson, 1998.

[48] E. H. Ruspini, “Numerical methods for fuzzy clustering,” *Information Sciences*, vol. 2, n°. 3, p. 319 – 350, 1970. [En ligne]. Disponible : http://www.sciencedirect.com/science/article/pii/S0020025570800561

[49] K. P. Bennett, P. S. Bradley et A. Demiriz, “Constrained k-means clustering,” Microsoft Research, Rapport technique MSR-TR-2000-65, May 2000. [En ligne]. Disponible : https://www.microsoft.com/en-us/research/publication/constrained-k-means-clustering/

[50] IBM, *IBM ILOG CPLEX Optimization Studio : CP Optimizer User’s Manual*, International Business Machines Corporation.

[51] P. Vilím, P. Laborie et P. Shaw, “Failure-directed search for constraint-based scheduling,” dans *Integration of AI and OR Techniques in Constraint Programming*, L. Michel, édit. Cham : Springer International Publishing, 2015, p. 437–453.

[52] G. Pesant, “Counting-based search for constraint optimization problems,” dans *Thirtieth AAAI Conference on Artificial Intelligence*, 2016. [En ligne]. Disponible : https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12065

[53] G. Pesant et J.-C. Régin, “Spread : A balancing constraint based on statistics,” dans *Principles and Practice of Constraint Programming - CP 2005*, P. van Beek, édit. Berlin, Heidelberg : Springer Berlin Heidelberg, 2005, p. 460–474.