Un nouveau modèle pour la classification non supervisée sous contraintes

Thi-Bich-Hanh Dao, Khanh-Chuong Duong, Christel Vrain

Univ. Orléans, INSA Centre Val de Loire
LIFO EA 4022, F-45067, Orléans, France
(thi-bich-hanh.dao,khanh-chuong.duong,christel.vrain)@univ-orleans.fr

RÉSUMÉ. *La classification non supervisée sous contraintes utilisateur a connu un essor important en fouille de données. Dans les dix dernières années, beaucoup de travaux se sont attachés à étendre les algorithmes classiques pour prendre en compte des contraintes utilisateur, mais ils sont en général limités à un seul type de contraintes. Dans de précédents travaux (Dao* et al.*, 2013a), nous avons proposé un cadre générique et déclaratif, fondé sur la programmation par contraintes, qui permet de modéliser différentes tâches de clustering sous contraintes. L'utilisateur peut spécifier un parmi plusieurs critères d'optimisation et combiner différents types de contraintes. Ce modèle exige que le nombre de classes soit fixé à l'avance. Dans ce papier, nous présentons un nouveau modèle pour le clustering sous contraintes où le nombre de classes est seulement borné par des bornes inférieure et supérieure. Ce modèle, tout en offrant plus de flexibilité, est plus efficace et les expérimentations montrent que les performances obtenues sont meilleures que les méthodes complètes existantes traitant des mêmes critères.*

ABSTRACT. *Constrained clustering is an important task in Data Mining. In the last ten years, many works have been done to extend classical clustering algorithms to handle user-defined constraints, but they are in general limited to one kind of constraints. In our previous work (Dao* et al.*, 2013a), we have proposed a declarative and general framework, based on Constraint Programming, which enables to design a clustering task by specifying an optimization criterion and different kinds of user-constraints. The model is designed for a clustering task, where data is divided in exactly k clusters. In this paper, we present a new model for constrained clustering tasks where the number of clusters is only bounded. It offers more flexibility, while being more efficient and experiments show that it has a better performance, when compared to existing complete methods dealing with the same criteria.*

MOTS-CLÉS : *classification non supervisée, contraintes utilisateur, programmation par contraintes.*

KEYWORDS: *clustering, constrained clustering, constraint programming.*

DOI:10.3166/RIA.28.523-545

1. Introduction

La classification non supervisée (aussi appelée clustering) sous contraintes est une tâche importante de fouille de données, permettant de modéliser plus finement une tâche de clustering en intégrant des contraintes utilisateur. Plusieurs types de contraintes peuvent être considérés ; elles peuvent porter sur les clusters, comme par exemple sur leur diamètre ou sur leur taille, ou porter sur des paires d'objets qui doivent être ou pas dans une même classe. Néanmoins, concevoir un algorithme capable de traiter différents types de contraintes est difficile et la plupart des approches classiques étendues au clustering sous contraintes ne prennent en compte qu'un seul type de contraintes (Basu *et al*., 2008). Depuis quelques années, les approches déclaratives, offrant des cadres génériques, se sont avérées être intéressantes pour la fouille de données (De Raedt *et al*., 2008 ; 2010 ; Métivier *et al*., 2012). Dans cette lignée, dans de précédents travaux (Dao *et al*., 2013c ; 2013a), nous avons proposé un cadre déclaratif et générique, fondé sur la programmation par contraintes (PPC) qui permet de modéliser différentes tâches de clustering sous contraintes, étant donnée une mesure de dissimilarités entre les objets. L'utilisateur peut spécifier un parmi plusieurs critères d'optimisation et combiner différents types de contraintes utilisateur, portant sur les clusters ou sur des paires d'objets. Le modèle proposé était basé sur une représentation à deux niveaux : un ensemble de variables affectant un point représentatif à chaque classe [1] et un ensemble de variables affectant un point représentatif à chaque objet. Il nécessitait que le nombre de classes soit fixé *a priori*. Dans ce papier, nous proposons un nouveau modèle plus simple, qui est composé d'un seul ensemble de variables affectant à chaque objet le numéro de la classe à laquelle il appartient. Dans ce modèle, l'utilisateur a la flexibilité de ne pas fixer le nombre de clusters à l'avance, mais seulement de donner une borne supérieure k_{max} et une borne inférieure k_{min} sur le nombre de clusters. Le modèle trouve, s'il existe une solution, une partition en k classes, avec $k_{min} \leq k \leq k_{max}$, satisfaisant toutes les contraintes et optimisant globalement le critère spécifié. De plus, ce nouveau modèle, tout en offrant plus de flexibilité, est plus efficace en temps et en espace que le modèle précédent.

Il est bien connu que l'efficacité d'un modèle fondé sur la PPC dépend fortement du choix des variables qui représentent les éléments du problème et du choix des contraintes utilisées. Pour exploiter pleinement la puissance de la PPC, il est également nécessaire de renforcer la propagation de contraintes et la capacité à casser les symétries (différentes représentations d'une même solution). Nous étudions des améliorations du modèle reposant sur des résultats théoriques, permettant de l'alléger sans changer la sémantique. Nous avons développé un algorithme de filtrage pour le critère de la somme des dissimilarités intra-cluster, ce qui permet de mieux exploiter les connaissances lors d'une affectation partielle des variables. Pour les contraintes qui cassent la symétrie, nous avons développé un algorithme de filtrage afin d'avoir une propagation plus rapide. Des expérimentations sur des bases de données classiques

1. Le point représentatif est celui ayant le plus petit indice parmi les points du cluster. En général ce n'est pas le barycentre du cluster.

montrent que notre modèle est meilleur que les méthodes complètes existantes traitant les mêmes critères dans le cas sans contraintes utilisateur. De plus, la possibilité de combiner différents types de contraintes utilisateur et la flexibilité de choix sur le nombre de clusters permettent d'obtenir des solutions de meilleure qualité.

Le papier est organisé comme suit. Dans la section 2, nous rappelons des notions de base sur la classification non supervisée, le clustering sous contraintes et la PPC. Différents travaux connexes sont discutés dans la section 3. La section 4 est consacrée au nouveau modèle et la section 5 à des améliorations permettant d'améliorer les performances du modèle. Des expérimentations sont présentées dans la section 6 et une discussion sur les travaux futurs est donnée dans la section 7.

2. Préliminaires

2.1. Classification non supervisée

La classification non supervisée, souvent appelée par son équivalent anglais clustering, est une tâche de fouille de données qui a pour objectif de regrouper des données en un ensemble de classes ou clusters, de façon à ce que les objets d'une même classe aient une forte similarité entre eux et diffèrent fortement des objets des autres classes. La classification non supervisée est souvent vue comme un problème d'optimisation, i.e., trouver une partition des objets qui optimise un critère donné. Plus formellement, nous considérons une base composée de n objets $\mathcal{O} = \{o_1, \ldots, o_n\}$ et une mesure de dissimilarité $d(o_i, o_j)$ entre deux objets o_i et o_j de $\mathcal{O}$. Par définition, une partition des objets en k classes $C_1, \ldots, C_k$ est telle que : (1) $\forall c \in [1, k]$ [2], $C_c \neq \emptyset$, (2) $\cup_c C_c = \mathcal{O}$ et (3) $\forall c \neq c'$, $C_c \cap C_{c'} = \emptyset$. Les critères à optimiser souvent utilisés sont :

– Minimiser le plus grand diamètre, où le diamètre d'une classe est défini comme la dissimilarité maximale entre deux objets de cette classe :

$$\text{minimise} \max_{c \in [1,k], o_i, o_j \in C_c} (d(o_i, o_j)).$$

– Maximiser la plus petite marge, où la marge entre deux classes $C_c, C_{c'}$ distinctes est définie comme la plus petite des dissimilarités $d(o_i, o_j)$, avec $o_i \in C_c$ et $o_j \in C_{c'}$:

$$\text{maximise} \min_{c < c' \in [1,k], o_i \in C_c, o_j \in C_{c'}} (d(o_i, o_j)).$$

– Minimiser la somme des dissimilarités intra-clusters (WCSD, pour Within-Cluster Sum of Dissimilarities) :

$$\text{minimise} \sum_{c=1}^{k} \frac{1}{2} \sum_{o_i, o_j \in C_c} d(o_i, o_j).$$

2. Nous utilisons la notation $[1, k]$ pour désigner l'ensemble $\{1, .., k\}$

– Minimiser le critère des moindres carrés (WCSS, pour Within-Cluster Sum of Squares), ce critère est souvent défini dans un espace euclidien avec la dissimilarité mesurée par la distance euclidienne au carré et m_c le centre de la classe C_c :

$$\text{minimise} \sum_{c=1}^{k} \sum_{o_i \in C_c} d(o_i, m_c).$$

Lorsque la distance euclidienne au carré est utilisée pour la mesure de dissimilarité, nous avons la relation suivante entre WCSD et WCSS :

$$\sum_{c=1}^{k} \sum_{o_i \in C_c} d(o_i, m_c) = \sum_{c=1}^{k} \frac{1}{2|C_c|} \sum_{o_i, o_j \in C_c} d(o_i, o_j)$$

– Minimiser l'erreur absolue, avec $r_c \in \mathcal{O}$ l'objet représentatif de la classe C_c :

$$\text{minimise} \sum_{c=1}^{k} \sum_{o_i \in C_c} d(o_i, r_c).$$

La classification non supervisée est un problème NP-difficile pour tous ces critères d'optimisation sauf le dernier, la plupart des algorithmes classiques utilisent donc des heuristiques et trouvent un optimum local. On peut citer par exemple, l'algorithme des k-moyennes pour le critère WCSS, les k-médoides pour le critère d'erreur absolue (Han *et al*., 2011) ou FPF (Furthest Point First) (Gonzalez, 1985) pour la minimisation du diamètre maximal. Certains algorithmes ne reposent pas sur un critère d'optimisation, comme par exemple DBSCAN (Ester *et al*., 1996), fondé sur la notion de densité, mais nécessitant de fixer deux paramètres d'entrée un rayon ϵ et un seuil $MinPts$ afin d'ajuster la notion de densité.

2.2. Classification non supervisée sous contraintes

La plupart des algorithmes de classification non supervisée reposent sur un critère d'optimisation, et pour des raisons de complexité ne cherchent qu'un optimum local. Plusieurs optima peuvent exister, certains pouvant être plus proches de celui recherché par l'utilisateur. Afin de mieux modéliser la tâche d'apprentissage, mais aussi dans l'espoir de réduire la complexité, des contraintes définies par l'utilisateur peuvent être ajoutées. On parle alors de classification non supervisée sous contraintes dont le but est de trouver des clusters satisfaisant les contraintes utilisateur. Ces contraintes peuvent porter sur les classes (contrainte de classe) ou sur les objets (contrainte objets). Beaucoup de travaux ont porté sur les contraintes objets, introduites dans (Wagstaff, Cardie, 2000). Deux types de contraintes objets sont couramment utilisés : must-link ou cannot-link. Une contrainte must-link spécifie que deux objets o_i et o_j doivent apparaître dans la même classe : $\forall c \in [1, k],\ o_i \in C_c \Leftrightarrow o_j \in C_c$. Une contrainte cannot-link spécifie que deux objets ne doivent pas être dans la même classe : $\forall c \in [1, k],\ \neg(o_i \in C_c \wedge o_j \in C_c)$.

Les contraintes de classe imposent des restrictions sur les classes. La contrainte de capacité minimale (resp. maximale) exige que chaque classe ait au moins (resp. au plus) un nombre donné α (resp. β) d'objets : $\forall c \in [1,k],\ |C_c| \geq \alpha$ (resp. $\forall c \in [1,k],\ |C_c| \leq \beta$). La contrainte de diamètre maximum spécifie une borne supérieure γ sur le diamètre de chaque cluster : $\forall c \in [1,k], \forall o_i, o_j \in C_c,\ d(o_i,o_j) \leq \gamma$. La contrainte de marge minimale, aussi appelée δ-contrainte dans (Davidson, Ravi, 2005b), spécifie une borne inférieure δ sur la marge entre deux classes : $\forall c \in [1,k]$, $\forall c' \neq c, \forall o_i \in C_c, o_j \in C_{c'}, d(o_i,o_j) \geq \delta$. La ϵ-contrainte introduite dans (Davidson, Ravi, 2005b) demande que chaque point o_i ait dans son voisinage de rayon ϵ au moins un autre point de la même classe : $\forall c \in [1,k], \forall o_i \in C_c, \exists o_j \in C_c, o_j \neq o_i \wedge d(o_i,o_j) \leq \epsilon$. Cette contrainte tente de capturer la notion de densité, introduite dans DBSCAN. Nous proposons une nouvelle contrainte fondée sur la densité, plus forte que la ϵ-contrainte: pour tout point o_i, son voisinage de rayon ϵ doit contenir au moins m points appartenant à la même classe que o_i.

Dans les dix dernières années, beaucoup de travaux ont étendu les algorithmes classiques pour traiter des contraintes must-link et cannot-link, comme par exemple une extension de COBWEB (Wagstaff, Cardie, 2000), des k-moyennes (Wagstaff *et al.*, 2001 ; Bilenko *et al.*, 2004), de la classification hiérarchique (Davidson, Ravi, 2005a) ou de la classification spectrale (Lu, Carreira-Perpinan, 2008 ; Wang, Davidson, 2010). Ceci est effectué en modifiant soit la mesure de dissimilarité, soit la fonction objectif à optimiser ou encore la stratégie de recherche. Ces algorithmes sont spécifiques pour un critère d'optimisation ou pour un type de contraintes utilisateur. Notre approche fondée sur la programmation par contraintes permet d'intégrer différents critères d'optimisation et différents types de contraintes utilisateur.

2.3. Programmation par contraintes

La programmation par contraintes (PPC) est un paradigme puissant pour résoudre des problèmes combinatoires, fondé sur des méthodes issues de l'intelligence artificielle ou de la recherche opérationnelle. Un *problème de satisfaction de contraintes (CSP)* est un triplet $\langle X, D, C \rangle$ où $X = \{x_1, x_2, \ldots, x_n\}$ est un ensemble de variables, Dom une fonction associant à chaque variable $x_i \in X$ son domaine $Dom(x_i)$, $C = \{C_1, C_2, ..., C_t\}$ est un ensemble de contraintes où chaque contrainte C_i exprime une condition sur un sous-ensemble de X. Une solution d'un CSP est une affectation complète de valeurs $a_i \in Dom(x_i)$ à chaque variable x_i qui satisfait toutes les contraintes de C. Un *problème d'optimisation de contraintes (COP)* est un CSP avec de plus une fonction objectif à optimiser. Une solution optimale d'un COP est une solution du CSP qui optimise la fonction objectif. Résoudre un CSP est en général NP-difficile. Néanmoins les méthodes utilisées par les solveurs permettent de résoudre efficacement un grand nombre d'applications réelles. Elles reposent sur la propagation de contraintes et des stratégies de recherche adaptées.

La propagation d'une contrainte c enlève du domaine des variables de c des valeurs dont on est certain qu'elles ne pourront pas apparaître dans une solution. Un

ensemble de propagateurs est associé à chaque contrainte, dépendant de la consistance souhaitée pour cette contrainte (par exemple la consistance d'arc enlève toutes les valeurs inconsistantes, alors que la consistance de bornes modifie les bornes du domaine) ; la consistance est choisie par le programmeur. Les relations arithmétiques et logiques sont disponibles comme contraintes, ainsi que des relations plus complexes exprimées sous forme de contraintes globales. Les contraintes globales représentent des relations sur des ensembles de variables et en général, peuvent être exprimées par une conjonction de contraintes élémentaires. Cependant, définies d'une façon globale, les contraintes globales bénéficient de propagations beaucoup plus efficaces, qui sont effectuées par des algorithmes de filtrage. Ces algorithmes se basent souvent sur la théorie des graphes ou sur la recherche opérationnelle.

Un solveur de contraintes itère une étape de propagation suivie d'une étape de branchement jusqu'à ce qu'une solution soit trouvée. Les contraintes sont propagées jusqu'à atteindre un état stable, dans lequel les domaines des variables ont été réduits autant que possible. Si les domaines de toutes les variables sont réduits à un singleton, une solution est trouvée. Si le domaine d'une variable devient vide, alors il n'existe aucune solution avec l'actuelle affectation partielle des variables et le solveur effectue un retour arrière. Dans les autres cas, le solveur choisit une variable dont le domaine n'est pas réduit à un singleton et divise le domaine de la variable en plusieurs parties, créant ainsi plusieurs nouvelles branches dans l'arbre de recherche. Le solveur explore chaque branche suivant une stratégie, activant la propagation de contraintes puisque le domaine d'une variable a été modifié.

La stratégie de recherche est déterminée par le programmeur. Elle est indiquée par le choix des variables et des valeurs pour créer les branches. Pour trouver une ou toutes les solutions, l'exploration en profondeur d'abord est souvent utilisée ; le solveur crée et ordonne les branches, suivant la stratégie donnée par le programmeur et explore en profondeur chaque branche. Pour un problème d'optimisation, la recherche *branch-and-bound* est intégrée à la recherche en profondeur : dès qu'une solution est trouvée, la valeur de la fonction objectif pour cette solution est calculée et une nouvelle contrainte est ajoutée, exprimant que toute nouvelle solution doit être meilleure que celle-ci. En conséquence, seule la première meilleure solution trouvée est retournée par le solveur. Il est important de remarquer que le solveur effectue une recherche complète, n'élaguant des branches que si elles ne peuvent pas conduire à une meilleure solution, et trouve donc une solution optimale. Le choix des variables et des valeurs à chaque branchement est important, puisqu'il peut réduire drastiquement l'espace de recherche et donc le temps de calcul. Pour plus de détails voir (Rossi *et al.*, 2006).

3. Travaux connexes

Depuis quelques années, les approches déclaratives qui offrent des cadres génériques se sont avérées prometteuses pour la classification non supervisée. Dans le cadre du clustering conceptuel où à chaque objet est associé l'ensemble des propriétés qu'il satisfait, des travaux récents (Guns *et al.*, 2013 ; Métivier *et al.*, 2012) ont

déjà proposé l'utilisation de la PPC. Le problème est formalisé comme la recherche d'un ensemble de k motifs fréquents, où un motif est un ensemble de propriétés, deux à deux non recouvrants, qui couvrent toutes les données ; les motifs trouvés peuvent alors être vus comme des définitions en intension des classes. Plusieurs critères d'optimisation sont considérés comme maximiser la taille minimale d'une classe ou minimiser la différence entre les tailles des classes. Ces approches sont adaptées aux bases de données qualitatives, alors que notre approche peut traiter tout type de données dès lors que l'on dispose d'une mesure de dissimilarité entre les données. Une autre approche basée sur la programmation linéaire en nombres entiers a été proposée (Mueller, Kramer, 2010 ; Schmidt *et al.*, 2011). Dans cette approche, un ensemble de clusters candidats doit être connu à l'avance et le modèle cherche le meilleur clustering formé par des candidats. Pour le clustering conceptuel, les clusters candidats sont définis par les motifs fréquents. L'approche n'est cependant pas appropriée aux tâches de clustering en général, car trouver un bon ensemble de clusters candidats est difficile, vu que le nombre de clusters candidats est exponentiel par rapport au nombre d'objets. Dans le cadre appelé "correlation clustering", une approche basée sur MaxSAT est proposée (Berg, Jarvisalo, 2013). Elle vise à minimiser la somme des dissimilarités intra-clusters et des similarités inter-clusters et permet d'intégrer directement des contraintes must-link ou cannot-link.

Dans le cadre de la classification non supervisée à base de distance, le problème avec la minimisation de la somme des dissimilarités intra-cluster (WCSD) est NP-difficile car le problème max-cut, qui est NP-complet (Garey, Johnson, 1979), est un cas particulier de ce problème avec $k = 2$. Le clustering avec minimisation de la somme des moindres carrés (WCSS) est aussi un problème NP-difficile (Aloise *et al.*, 2009), même avec $k = 2$. Le clustering avec minimisation du diamètre maximal est polynomial pour $k = 2$, mais devient NP-difficile dès que $k \geq 3$ (Hansen, Delattre, 1978). Si on considère le critère de maximiser la marge entre clusters, le problème est polynomial (Delattre, Hansen, 1980), cependant il devient NP-difficile avec des contraintes utilisateur comme par exemple des contraintes cannot-link, (Davidson, Ravi, 2007). Les algorithmes utilisés sont souvent heuristiques ; par exemple k-means trouve un optimum local pour le critère WCSS. Il existe peu d'algorithmes exacts, ils se basent en général sur des bornes inférieures, mais trouver de telles bornes est souvent un problème difficile. Pour le critère du diamètre, l'algorithme de (Hansen, Delattre, 1978) se base sur la coloration de graphes, alors que celle de (Brusco, Stahl, 2005) utilise la recherche *branch-and-bound*. Pour le critère WCSD, la meilleure méthode exacte à notre connaissance est un algorithme de recherche *branch-and-bound* répétitif (Brusco, 2006). L'algorithme résout d'abord le problème avec $k + 1$ objets, puis avec $k + 2$ objets et ainsi de suite, jusqu'à ce que les n objets soient considérés. Lors de la résolution d'un problème de taille plus grande, les résultats des sous-problèmes sont utilisés pour calculer des bornes inférieures. Pour le critère WCSS, la meilleure méthode est un algorithme par génération de colonnes (Aloise *et al.*, 2012). Il est à noter que chacune de ces méthodes est conçue pour un critère particulier.

Des approches déclaratives offrent des cadres de modélisation plus généraux et flexibles. Une approche déclarative pour le clustering à base de distance est un cadre

SAT pour des problèmes à deux classes ($k = 2$) (Davidson *et al.*, 2010). Il permet d'intégrer les contraintes must-link ou cannot-link et des contraintes de classe (diamètre des clusters, marge entre clusters) et l'algorithme converge vers un optimum global. Avec $k = 2$ le problème est encodé par un problème 2-SAT pour lequel la complexité est polynomiale. Une extension pour des valeurs $k > 2$ conduirait à des problèmes SAT plus généraux et nécessiterait des encodages adaptés. Notre approche est plus générale dans la mesure où différents autres critères sont traités, un ensemble plus grand de contraintes est considéré et le nombre de classes n'est pas limité à 2. Considérant le critère WCSS, une approche basée sur la programmation linéaire, qui étend l'algorithme par génération de colonne, est proposée dans (Babaki *et al.*, 2014). Cette approche intègre différents types de contraintes utilisateur (must-link, cannot-link et contraintes anti-monotones) et trouve un optimum global. Cependant, à notre connaissance, il n'existe pas de cadre générique pour les critères de diamètre, de marge ou WCSD et qui intègrent des contraintes utilisateur.

4. Un nouveau modèle en PPC pour le clustering sous contraintes

Nous disposons d'une collection de n points et d'une mesure de dissimilarité d entre paires de points. Sans perdre de généralité, nous supposons que les points sont indexés et nommés par leur indice (ainsi 1 représente le 1er point). Le modèle a pour but de trouver une partition des données en k classes, satisfaisant un ensemble de contraintes spécifié par l'utilisateur et optimisant un critère donné. Dans (Dao *et al.*, 2013a ; 2013c), nous avons présenté un modèle en PPC pour cette tâche imposant que le nombre k de classes soit fixé d'avance par l'utilisateur. Dans cette section, nous présentons un nouveau modèle dans lequel le nombre k n'est plus fixé mais borné entre k_{min} et k_{max}, paramètres fixés par l'utilisateur. Ce modèle, tout en offrant plus de flexibilité, est plus efficace que le modèle précédent.

4.1. Variables

Les classes (clusters) sont identifiées par leur numéro, qui varie de 1 au nombre de classes. Pour représenter l'affectation des points aux classes, nous utilisons des variables $G_1, \ldots, G_n$, dont le domaine est l'ensemble des entiers de $[1, k_{max}]$. Une affectation $G_i = c$ signifie que le point i est affecté à la classe numéro c. Nous utilisons $\mathcal{G}$ pour désigner la collection de variables $G_1, \ldots, G_n$.

Une variable est aussi introduite pour représenter le critère à optimiser. Elle est écrite D pour le diamètre maximal, S pour la marge minimale et V pour la somme des dissimilarités intra-clusters. Elle est à valeur réelle, puisque les distances sont réelles. Les domaines de D et S sont $[\min_{i,j\in[1,n]}(d(i,j)), \max_{i,j\in[1,n]}(d(i,j))]$. Le domaine de V est $[0, \sum_{i<j\in[1,n]} d(i,j)]$.

4.2. Contraintes

La tâche de classification non supervisée sous contraintes utilisateur est modélisée par les contraintes de la PPC. Ces contraintes modélisent la recherche d'une partition, différents critères d'optimisation et des contraintes utilisateur.

4.2.1. Contraintes de partition

Ces contraintes expriment les relations entre les points et leurs clusters. Une affectation complète des variables G_i (une solution) représente une partition. Cependant, une partition peut se représenter par plusieurs solutions, à cause des permutations des numéros de cluster. Par exemple, à partir d'une solution (qui représente une partition), échanger le numéro de deux clusters crée une nouvelle solution mais qui représente toujours la même partition lorsqu'on ignore les numéros de cluster. Une permutation de numéros peut apparaître lors de la création d'un nouveau cluster, si un numéro quelconque parmi les numéros restants lui est attribué. Pour casser cette symétrie entre les partitions, les clusters sont numérotés de telle manière qu'un nouveau numéro c soit utilisé pour un nouveau cluster si et seulement si tous les autres numéros $c' < c$ sont déjà utilisés. Le premier cluster doit contenir le premier point. Si le deuxième point doit être placé dans un nouveau cluster, ce cluster portera le numéro 2. Si nous avons déjà 2 clusters et qu'un nouveau cluster doit être créé, il portera le numéro 3 et ainsi de suite. Une partition en clusters est modélisée par les contraintes suivantes :

– Le premier point appartient au premier cluster : $G_1 = 1$.

– La condition que le nombre de clusters ne doive pas dépasser k_{max} est trivialement satisfaite puisque le domaine de chaque variable G_i est l'ensemble des entiers compris entre 1 et k_{max}, ne permettant que k_{max} numéros de clusters.

– Que le nombre minimal de clusters soit k_{min} signifie que tous les numéros entre 1 et k_{min} doivent être utilisés dans les affectations de G_i. Autrement dit, pour tout $c \in [1, k_{min}]$, la valeur c doit être prise au moins une fois par des variables de $G_1, \ldots, G_n$. Pour chaque valeur $c \in [1, k_{min}]$, nous posons une contrainte globale

$$\mathit{AtLeast}(1, \mathcal{G}, c)$$

qui exige qu'au moins une variable de $\mathcal{G}$ prenne la valeur c. Si l'utilisateur souhaite exactement k classes, il suffit de fixer $k_{min} = k_{max} = k$.

– Un numéro c de cluster est utilisé si et seulement si tous les numéros $c' < c$ sont déjà utilisés : $\forall i \in [2, n]$

$$G_i \leq \max_{j \in [1, i-1]} (G_j) + 1.$$

Cette contrainte impose que chaque point i doit être dans le même cluster qu'un point précédent, ou dans un nouveau cluster de numéro $c = c' + 1$, où c' est le plus grand des numéros déjà utilisés.

4.2.2. Modélisation du critère à optimiser

Pour le critère du diamètre maximal, étant donné que D représente le diamètre maximal des clusters, nous posons la fonction objectif *minimize* D. De plus, deux

points ayant une dissimilarité supérieure au diamètre maximal doivent être dans des clusters différents. Ceci est représenté par les contraintes réifiées suivantes [3] : $\forall i < j \in [1, n]$,

$$d(i, j) > D \rightarrow (G_i \neq G_j). \quad (1)$$

Pour le critère de maximisation de la marge minimale, nous posons la fonction objectif *maximize* S. De plus, deux points ayant une dissimilarité inférieure à la marge minimale doivent être dans le même cluster : $\forall i < j \in [1, n]$,

$$d(i, j) < S \rightarrow G_i = G_j. \quad (2)$$

Pour la minimisation de la somme des dissimilarités intra-classe (WCSD), qui est représentée par V, nous posons la fonction objectif *minimize* V et

$$V = \sum_{i<j\in[1,n]} (G_i == G_j) d(i, j). \quad (3)$$

4.2.3. Modélisation des contraintes utilisateur

Les contraintes utilisateur classiques peuvent être intégrées directement :

– Taille minimale α des clusters : cela signifie que chaque point doit se trouver dans un cluster avec au moins α points (en comptant lui-même). Pour chaque $i \in [1, n]$, au moins α variables de $\mathcal{G}$ doivent prendre la valeur affectée à G_i. Nous utilisons donc la contrainte : *AtLeast*$(\alpha, \mathcal{G}, G_i)$.

– Taille maximale β des clusters : chaque numéro $c \in [1, k_{max}]$ doit être affecté à au plus β variables dans $\mathcal{G}$ (ceci est vrai aussi pour un numéro $c \in [k_{min} + 1, k_{max}]$, qui n'est pas utilisé, car dans ce cas c apparaît 0 fois). Pour chaque valeur $c \in [1, k_{max}]$, nous posons une contrainte : *AtMost*$(\beta, \mathcal{G}, c)$.

– Une δ-contrainte exprime que la marge entre deux clusters doit être au moins δ. Pour l'exprimer, pour chaque couple (i, j), $i < j \in [1, n]$ satisfaisant $d(i, j) < \delta$, nous posons la contrainte : $G_i = G_j$.

– Une contrainte de diamètre exprime que le diamètre d'un cluster ne doit pas dépasser γ. Pour l'exprimer, pour chaque couple (i, j), $i < j \in [1, n]$ tel que $d(i, j) > \gamma$, nous posons la contrainte : $G_i \neq G_j$.

– La contrainte de densité que nous avons introduite exprime que chaque point doit avoir dans son voisinage de rayon ϵ, au moins m points appartenant au même cluster que lui. Pour tout point $i \in [1, n]$, l'ensemble de points dans son ϵ-voisinage est calculé $\mathcal{X}_i = \{G_j \mid d(i, j) \leq \epsilon\}$ et cette contrainte est posée : *AtLeast*$(m, \mathcal{X}_i, G_i)$.

– Une contrainte must-link entre deux points i et j est exprimée par : $G_i = G_j$.

– Une contrainte cannot-link sur i et j est exprimée par : $G_i \neq G_j$.

3. Une contrainte réifiée $A \rightarrow B$ représente la relation implication, la contrainte B devient effective si la contrainte A est satisfaite, et si la contrainte B n'est pas satisfaite alors $\neg A$ doit être satisfaite.

4.3. Stratégie de recherche

Le branchement est réalisé sur les variables de $\mathcal{G}$. Le choix dépend du critère à optimiser. Pour le critère minimiser le diamètre maximal ou maximiser la marge minimale, une variable G_i avec le plus petit domaine restant est choisie en premier. Quand G_i est choisie, toutes les valeurs c du domaine de G_i sont examinées et nous choisissons le numéro de la classe la plus proche du point i. La distance entre un point i et une classe c est définie comme la distance maximale entre i et tous les points j pour lesquels c est déjà affectée à G_j. Si une classe c est vide (il n'existe pas de point instancié G_j tel que $G_j = c$) la distance entre i et cette classe est nulle. Cela signifie que l'on privilégie d'affecter le point à une nouvelle classe s'il reste un numéro ; de plus, le premier numéro de classe vide suffit. La classe c_0 la plus proche de i est choisie et le branchement sur G_i fait deux alternatives $G_i = c_0$ et $G_i \neq c_0$. Cette stratégie diffère de celle utilisée dans le premier modèle, où le branchement dépendait de la distance entre le point i et le représentant de chaque classe.

Pour la somme des dissimilarités intra-classe, une stratégie mixte est utilisée. Puisque déterminer une borne supérieure raisonnable pour la somme des dissimilarités intra-classe est nécessaire, une recherche gloutonne est utilisée en premier pour trouver rapidement une solution. A cette étape, lors de chaque branchement, on choisit une variable G_i et une valeur c telles que l'affectation $G_i = c$ fait augmenter V le moins possible. La solution trouvée est en général assez bonne. Une fois cette solution trouvée, la stratégie de recherche est changée en une stratégie "first-fail" qui tend à détecter des échecs rapidement. La somme de dissimilarités s_{ic} entre un point i et une classe c est définie comme la somme de dissimilarités entre i et tous les points j tels que G_j est instanciée à c. On calcule pour chaque point i une valeur $s_{min} = \min_{c \in Dom(G_i)} s_{ic}$. Lors de chaque branchement, on choisit une variable G_i correspondant à un point i qui a la valeur s_{min} maximale. La valeur c choisie est celle qui donne la valeur s_{min} au point i.

5. Améliorations du modèle

Des solveurs de PPC cherchent des solutions en itérant deux phases : propagation de contraintes et branchement. Améliorer la propagation de contraintes ou des stratégies de recherche a donc un grand impact sur l'efficacité du modèle. Différentes études concernant ces aspects ont été réalisées afin de rendre le modèle plus performant.

5.1. Réorganisation de points

Lors de branchement, une variable G_i ayant le plus petit domaine est choisie. S'il existe plusieurs candidats, la variable de plus petit indice i sera choisie. Nous réordonnons donc les points de telle façon que les points "extrêmes" aient des indices faibles. Les premiers points sont alors le plus probablement des points appartenant à des classes différentes. A ces fins, nous utilisons l'algorithme FPF (Furthest Point First) (Gonzalez, 1985) pour réordonner les points. Cet algorithme, pour trouver k

clusters, choisit itérativement k points, chaque point choisi est appelé "tête" et lui sont affectés les points non encore choisis les plus proches de lui. Plus précisément, la première tête choisie t_1 est le point le plus loin des autres (la somme des dissimilarités de ce point avec les autres est maximale) et tous les points lui sont initialement rattachés, il devient donc la tête de ces points. A chaque étape i, le point le plus éloigné de sa tête est choisi, il devient une nouvelle tête t_i. Tous les points qui ne sont pas une tête sont alors examinés, si un point est plus proche de t_i que de sa tête alors il change de tête pour être rattaché à t_i. Lorsque k têtes sont choisies, les rattachements forment k clusters. Nous appliquons cet algorithme avec $k = n$, choisissant ainsi n points ; l'ordre selon lequel les points sont choisis est l'ordre pour réordonner nos points.

5.2. Critère du diamètre maximal

Dans le cas sans contraintes utilisateur, avec un k fixé, il est prouvé dans (Gonzalez, 1985) que le diamètre d_{FPF} calculé par l'algorithme FPF vérifie $d_{opt} \leq d_{FPF} \leq 2d_{opt}$, avec d_{opt} le diamètre optimal. Cette connaissance implique des bornes inférieure $D_l = d_{FPF}/2$ et supérieure $D_u = d_{FPF}$ pour le domaine de la variable D. Comme k varie entre k_{min} et k_{max}, la borne D_l sera $d_{FPF}/2$ pour $k = k_{max}$ et la borne D_u sera d_{FPF} pour $k = k_{min}$. De plus, pour chaque couple i, j :

– si $d(i, j) < D_l$, la contrainte réifiée (1) sur i, j est trivialement satisfaite, elle ne sera pas posée,

– si $d(i, j) > D_u$, la contrainte (1) est remplacée par une contrainte $G_i \neq G_j$,

– sinon, la contrainte (1) est posée.

Ce résultat permet de supprimer plusieurs contraintes réifiées sans modifier la sémantique du modèle. Puisque les contraintes réifiées nécessitent la gestion de variables supplémentaires, leur suppression permet d'améliorer l'efficacité du modèle.

Dans le cas où s'ajoutent des contraintes utilisateur, le diamètre optimal est en général supérieur à d_{opt}. La borne supérieure D_u du domaine de D n'est plus valable, par contre nous avons toujours la borne inférieure $D_l = d_{FPF}/2$ pour $k = k_{max}$. Ainsi, pour chaque couple i, j, tel que $d(i, j) < D_l$, la contrainte (1) ne sera pas posée.

5.3. Critère de la marge minimale entre clusters

Dans le cas sans contraintes utilisateur, trouver k clusters maximisant la marge minimale entre clusters est un problème polynomial. Il est prouvé dans (Delattre, Hansen, 1980) que, quelque soit la partition, quelque soit le nombre k, le nombre des valeurs possibles pour la marge minimale est au plus $n-1$. Pour trouver ces valeurs, on considère le graphe complet où les points sont des sommets et le poids de chaque arête $\{i, j\}$ est $d(i, j)$. On calcule un arbre couvrant de poids minimal pour le graphe. Les poids des arêtes de l'arbre couvrant forme l'ensemble des valeurs possibles pour la marge. Les arêtes de l'arbre sont ordonnées dans l'ordre croissant des poids, ce qui donne une suite $d_1, \ldots, d_{n-1}$. Pour trouver k clusters, on supprime les $k-1$ arêtes de poids

plus grands de l'arbre, ce qui crée k composantes connexes qui forment k clusters. La valeur d_{n-k+1} est la marge minimale entre clusters. Lorsque $k_{min} \leq k \leq k_{max}$, la marge varie entre $d_{n-k_{max}+1}$ et $d_{n-k_{min}+1}$.

Avec des contraintes utilisateur, le problème devient NP-difficile (Davidson, Ravi, 2007). La marge optimale avec contraintes utilisateur est en général inférieure à la marge optimale sans contraintes utilisateur. Nous pouvons donc toujours fixer la borne supérieure S_u de la variable S à $d_{n-k_{min}+1}$. Ainsi, pour chaque couple i, j tel que $d(i, j) > S_u$, la contrainte réifiée (2) ne sera pas posée. Plusieurs contraintes réifiées peuvent être supprimées sans modifier la sémantique du modèle.

5.4. Filtrage pour le critère de la somme des dissimilarités

En utilisant des contraintes prédéfinies, la contrainte (3) peut se réaliser par une contrainte linéaire sur des variables booléennes $V = \sum_{i<j} B_{ij} d(i, j)$, avec B_{ij}, pour $1 \leq i < j \leq n$, des variables booléennes de domaine $\{0, 1\}$, qui sont liées par les contraintes réifiées $B_{ij} \leftrightarrow (G_i = G_j)$. Cependant ces contraintes, qui sont considérées indépendamment, n'offrent pas suffisamment de propagation. Par exemple, prenons 4 points de 1 à 4, avec $k = 2$ et une affectation partielle où $G_1 = 1$ et $G_2 = G_3 = 2$. Nous avons trois variables booléennes instanciées $B_{12} = B_{13} = 0$, $B_{23} = 1$ et trois variables non instanciées $B_{14}, B_{24}, B_{34} \in \{0, 1\}$. Supposons qu'une solution avec $V = 5$ soit trouvée. Avec la recherche par *branch-and-bound*, une nouvelle contrainte $\sum_{i<j} B_{ij} d(i, j) < 5$ est ajoutée. Supposons que $d(1, 4) = d(2, 3) = 1$ et $d(2, 4) = 2$ et $d(3, 4) = 3$. La contrainte devienne $B_{14} + 2B_{24} + 3B_{34} < 4$. On constate que B_{24} et B_{34} doivent être égales car $G_2 = G_3$ et ne peuvent donc pas valoir 1, sinon la contrainte serait violée. On devrait conclure que le point 4 ne peut pas être dans le même cluster que les points 2 et 3, et supprimer ainsi la valeur 2 du domaine de G_4. Ce filtrage n'est malheureusement pas fait par la propagation de la contrainte somme.

En PPC, des filtrages plus efficaces pour la contrainte somme ont été proposés, lorsqu'elle est associée avec d'autres contraintes. Dans le cas d'une contrainte $y = \sum x_i$ avec des contraintes $x_j - x_i \leq c$, un algorithme peut réduire le domaine des x_i lorsque les bornes de y sont réduites (Régin, Rueher, 2005). Des algorithmes de réduction de bornes du domaine de variables sont proposés pour le cas d'une contrainte somme avec des contraintes d'ordre $x_i \leq x_{i+1}$ (Petit *et al.*, 2011) ou avec une contrainte *alldifferent*$(x_1, \ldots, x_n)$ (Beldiceanu *et al.*, 2012). Cependant la contrainte (3) n'en fait pas partie. Un algorithme générique de réduction de borne qui exploite une contrainte somme et un ensemble de contraintes est proposé dans (Régin, Petit, 2011). Dans notre cas, le domaine restant d'une variable G_i est constitué de valeurs qui sont des indices de clusters et n'est donc pas en général un intervalle continu.

Nous avons donc développé un algorithme de filtrage pour une nouvelle contrainte globale sur une variable V, une collection $\mathcal{G}$ de n variables et des constantes a_{ij} (avec $a_{ij} = a_{ji} \geq 0$ pour $i, j \in [1, n]$), qui est de la forme :

$$V = \sum_{1 \leq i < j \leq n} (G_i == G_j) a_{ij}. \tag{4}$$

La valeur de V dans une solution trouvée constitue une borne supérieure stricte V_u de la variable V. Supposons que nous ayons une instanciation partielle des variables de $\mathcal{G}$. L'ensemble $\{1, .., n\}$ est donc constitué de deux sous-ensembles disjoints, K l'ensemble des i avec G_i déjà instanciée et U l'ensemble des i avec G_i non encore instanciée. La somme (4) est séparée en trois parties :

- V_1 la somme des a_{ij} pour $i, j \in K$ et $G_i = G_j$, $i < j$,
- V_2 la somme des a_{ij} pour $i \in U$, $j \in K$ et $G_i = G_j$,
- V_3 la somme des a_{ij} pour $i, j \in U$ et $G_i = G_j$, $i < j$.

La valeur de V_1 peut se calculer exactement. Pour V_2 la valeur exacte est inconnue, mais une borne inférieure est calculée par la somme des contributions minimales des points non instanciés par rapport aux points instanciés :

$$V_{2l} = \sum_{i \in U} \min_{c \in Dom(G_i)} (\sum_{j \in K, G_j = c} a_{ij}).$$

La valeur exacte de V_3 est également inconnue, mais une borne inférieure peut être calculée. Soient $p = |U|$, $k = | \cup_{i \in U} Dom(G_i)|$, m le quotient et m' le reste de la division entière de p par k. Soit $f(p) = (km^2 + 2mm' - km)/2$. Une borne inférieure de V_3 est égale à la somme des $f(p)$ plus petites valeurs de a_{ij}, avec $i < j \in U$. Une borne inférieure V_l de V est donc la somme des bornes inférieures de V_1, V_2 et V_3. Ces bornes sont proposées dans (Klein, Aronson, 1991) et sont également utilisées dans (Brusco, 2003) dans le mécanisme de *branch-and-bound*, où l'échec est détecté si $V_l \geq V_u$. Nous exploitons davantage ces bornes pour construire un algorithme de filtrage pour réduire le domaine des variables G_i non encore instanciées.

L'algorithme examine chaque variable G_i non encore instanciée et chaque valeur c du domaine restant de G_i. Avec l'hypothèse que la valeur c soit affectée à la variable G_i, les nouvelles bornes inférieures V'_{1l} et V'_{2l} de V_1 et de V_2 peuvent être révisées en temps constant. Une nouvelle borne inférieure V'_{3l} de V_3 peut être la somme des $f(p-1)$ plus petites valeurs de a_{uv}, avec $u < v \in U - \{i\}$. Cependant, pour réduire la complexité du filtrage, nous prenons V_4 à la place de V'_{3l}. Ici V_4 est la somme des $f(p-1)$ plus petites valeurs de a_{uv}, avec $u < v \in U$. Il est évident que $V'_{3l} \geq V_4$. Si $V'_{1l} + V'_{2l} + V_4 \geq V_u$, sachant que le domaine de V est $[V_l, V_u)$, alors la valeur c est inconsistante, nous pouvons supprimer c du domaine de G_i. L'algorithme de filtrage a pour complexité $O(n^2 + nk)$. Pour plus de détails et de preuves, nous renvoyons le lecteur à l'article (Dao *et al.*, 2013b).

5.5. Filtrage pour contrainte de partition

Dans ce nouveau modèle, pour casser la symétrie entre les ensembles de clusters, on impose que le numéro d'un nouveau cluster soit inférieur ou égal à $c' + 1$, où c' est le plus grand des numéros déjà utilisés. Une implémentation directe est de poser pour chaque $i \in [2, n] : G_i \leq \max_{j \in [1, i-1]}(G_j) + 1$. Dans ce cas, pour une variable G_i, la contrainte est activée chaque fois le domaine d'une variable G_j ($j < i$) change.

Par conséquent, cette contrainte peut être activée plusieurs fois dans une étape de recherche. De plus, le modèle ne peut pas profiter des relations entre les contraintes individuelles pour réduire le nombre de calculs. Nous réalisons donc un algorithme de filtrage qui considère l'ensemble de ces contraintes comme une contrainte. L'algorithme est présenté dans l'algorithme 1, dans lequel les variables sont considérées tout ensemble en une passe.

Dans cet algorithme, nous utilisons la valeur max des domaines des variables déjà considérées. Pour chaque variable G_i, si G_i est instanciée et la valeur de G_i (numéro du cluster contenant i) est plus grande que max, nous changeons max. Si G_i est non encore instanciée, toutes les valeurs du domaine de G_i qui sont plus grandes que $max + 1$ sont inconsistantes et sont supprimées. Si $max + 1$ est dans le domaine de G_i, c'est-à-dire qu'il est possible d'affecter i au cluster numéro $max + 1$, la valeur de max est augmentée de 1 pour la considération du point suivant. Si $max = k - 1$, l'algorithme s'arrête car nous ne pouvons plus filtrer davantage.

L'algorithme fait au plus n itérations. A chaque itération i, il vérifie les valeurs du domaine restant de G_i et la taille maximale du domaine est k. La complexité dans le pire des cas est donc $O(nk)$. Cependant, l'algorithme s'arrête si $max = k - 1$ et dans la plupart de cas, max atteint $k - 1$ après moins de $2k$ itérations.

$i \leftarrow 1$;
$max \leftarrow 1$;
tant que $i < n$ *et* $max < k - 1$ **faire**
  si G_i *est instanciée* **alors**
    si $max < G_i$ **alors** $max = G_i$;
  sinon
    pour *chaque valeur* $a \in$ *Dom*(G_i) *et* $a > max + 1$ **faire**
      supprimer a du $Dom(G_i)$;
    si $max + 1 \in$ *Dom*(G_i) **alors** $max \leftarrow max + 1$;
  $i \leftarrow i + 1$

Algorithm 1: Algorithme de filtrage pour la contrainte anti-symétrie

6. Expérimentations

Onze bases de données du dépôt *UCI Machine Learning Repository* [4] sont utilisées dans nos expériences. Elles varient dans le nombre d'objets et le nombre de clusters. Pour la base *Wine Quality*, le nombre de classes est inconnu et nous choisissons $k = 4$ pour les tests. Les deux premières colonnes du tableau 1 résument les informations sur ces bases, qui sont présentées suivant le nombre croissant d'objets.

4. http://archive.ics.uci.edu/ml

Notre modèle ainsi que le modèle précédent (Dao *et al*., 2013a) sont implémentés en utilisant la bibliothèque Gecode [5] version 4.2.1. Les expérimentations sont réalisées sur un processeur 3.4 GHz Core i5 Intel sous Ubuntu 12.04.

6.1. Clustering avec différents critères

Pour le critère de diamètre maximal, nous comparons notre nouveau modèle (noté CP1) avec l'ancien modèle (noté CP2), l'algorithme basé sur la recherche *branch-and-bound* (Brusco, Stahl, 2005) (noté BaB) [6] et l'algorithme basé sur la coloration de graphe (Delattre, Hansen, 1980) (noté CdG) [7]. Nous considérons la classification non supervisée sans contraintes utilisateur car les algorithmes auxquels nous nous comparons ne peuvent pas traiter de contraintes et à notre connaissance, il n'existe pas d'algorithme exact avec des contraintes utilisateur pour ce critère. Dans les expérimentations, le temps est limité à 1 heure et la distance euclidienne est utilisée pour calculer la dissimilarité entre objets. Le nombre de classes k est fixé au nombre de classes réel pour les quatre algorithmes (pour le nouveau modèle, $k_{min} = k_{max} = k$).

Le tableau 1 montre les résultats des expérimentations. Pour chaque base nous présentons la valeur D_{opt} de diamètre optimal trouvé ainsi que le temps d'exécution en secondes de chaque système. Le signe - est utilisé lorsque le système ne trouve pas la solution optimale après 1 heure. Tous les systèmes trouvent naturellement la même valeur de diamètre optimal.

Tableau 1. Comparaison de performance avec la minimisation du diamètre maximal

Bases de données	n	k_{rel}	D_{opt}	CP1	CP2	CdG	BaB
Iris	150	3	2,58	<0,1	<0,1	1,8	1,4
Wine	178	3	458,13	<0,1	0,3	8,1	2
Glass	214	7	4,97	0,4	0,9	8,1	42
Ionosphere	351	2	8,6	0,3	0,4	0,6	-
User Knowledge	403	4	1,17	15,6	75	3,7	-
Breast Cancer	569	2	2 377,96	0,7	0,7	1,8	-
Synthetic Control	600	6	109,36	23,9	56,1	-	-
Vehicle	846	4	264,83	12,1	14,3	-	-
Yeast	1 484	10	0,67	559,1	2 446,7	-	-
Wine Quality	1 599	4	69,78	825,9	-	-	-
Image Segmentation	2 000	7	436,4	224,4	589,2	-	-

On peut constater que le nouveau modèle est le plus performant dans la plupart des cas, à l'exception de la base *User Knowledge Modeling*. Parmi les programmes, l'algorithme BaB est le moins efficace, il n'est pas capable de résoudre les bases de

5. http://www.gecode.org

6. Nous utilisons le programme disponible sur la page de l'auteur http://mailer.fsu.edu/~mbrusco/

7. Le programme est codé par nous-mêmes en C++.

données avec plus de 300 objets. La performance de CdG est rapidement réduite si le nombre d'objets n passe à 500. Les algorithmes BaB et CdG n'ont pas de mécanisme de réduction de domaines de variables. L'algorithme BaB se base sur les bornes pour détecter les échecs pendant la recherche, tandis que l'algorithme CdG examine toutes les distances disponibles dans l'ordre décroissant pour trouver le diamètre optimal. Notre nouveau modèle, qui est plus simple et qui exploite les avantages de la propagation de contraintes et des stratégies de recherche de la PPC, est plus performant.

Pour le critère de marge minimale, sans contraintes utilisateur, lorsque le nombre de classes k n'est pas fixé, $k_{min} \leq k \leq k_{max}$, la meilleure solution trouvée est toujours une solution avec $k = k_{min}$. Cependant, ce n'est plus le cas lorsqu'on ajoute des contraintes utilisateur, comme par exemple une contrainte de diamètre. Dans les expérimentations, pour créer une contrainte de diamètre, nous utilisons les résultats donnés dans le tableau 1 : le diamètre de chaque cluster ne doit pas dépasser $1,5D_{opt}$. Le nombre de classes k varie, $2 \leq k \leq k_{rel}$, avec k_{rel} le nombre réel de classes. Le tableau 2 reporte les résultats pour chaque base : la marge optimale S_{opt}, le nombre de classes de la solution k_{sol}, le nombre réel de classes k_{rel} et le temps d'exécution en secondes. Même en faisant varier le nombre de classes, le modèle est capable de résoudre presque tous les jeux de données, à l'exception de la base la plus grande.

Tableau 2. Critère de la marge avec contrainte de diamètre

Bases de données	S_{opt}	k_{sol}	k_{rel}	Temps (s)
Iris	0,53	3	3	<0,01
Wine	53,33	3	3	<0,01
Glass	1,78	7	9	0,5
Ionosphere	9,27	2	2	2,4
User Knowledge Modeling	0,32	4	4	1,0
Breast Cancer	422	2	2	5,4
Synthetic Control	45,16	5	6	2,9
Vehicle	27,06	4	4	17,2
Yeast	0,15	10	10	497,1
Wine Quality	4,11	4	4	702,0
Image Segmentation	-	-	7	-

Concernant le critère WCSD, la propagation de la contrainte WCSD est moins efficace que celle de diamètre ou de marge. Nous nous comparons avec l'algorithme de recherche *branch-and-bound* répétitif (Repetitive Branch and Bound Algorithm RBBA) (Brusco, 2006). A notre connaissance, c'est le meilleur algorithme exact pour ce critère, mais il n'intègre pas des contraintes utilisateur. Sans contraintes utilisateur, notre modèle et RBBA ne peuvent trouver la solution optimale que pour la base de données *Iris*. Notre modèle prend 4 174 secondes pour compléter la recherche alors que RBBA prend 3 249 secondes. Cependant, notre modèle est capable de traiter des bases plus grandes en présence de contraintes utilisateur, comme illustré dans le tableau 3. Dans ces exemples, d_{max} est la dissimilarité maximale entre objets.

Tableau 3. WCSD avec contraintes utilisateur

Base de données	Contraintes utilisateur	WCSD	Temps (s)
Wine	separateur $\geq 0,015 d_{max}$ taille de classe ≥ 30	$1,40 \times 10^6$	11,2
WDBC	diamètre $\leq 0,8 d_{max}$	$1,82 \times 10^{10}$	2,9
Vehicle	marge $\geq 0,03 d_{max}$ diamètre $\leq 0,4 d_{max}$	$1,93 \times 10^9$	1,6

6.2. Clustering avec des contraintes d'utilisateurs

Nous évaluons l'influence des contraintes utilisateur sur la qualité de la partition obtenue et sur la performance de notre modèle. Pour mesurer la qualité d'une partition, nous considérons l'indice ARI (Adjusted Rand Index). Cet indice mesure la similarité entre une partition P et la partition réelle P^* de la base de données. Il est défini par :

$$ARI = \frac{2(ab - cd)}{(a+d)(d+b) + (a+c)(c+b)}$$

où a et b sont les nombres de paires de points pour lesquelles P et P^* s'accordent (a nombre de paires qui se retrouvent dans la même classe dans P et dans P^*, b nombre de paires dans différentes classes), c et d sont les nombres de paires de points pour lesquelles P et P^* ne s'accordent pas (même classe dans P mais différentes classes dans P^* et vice versa). L'indice ARI varie entre 0 et 1 et plus les deux partitions s'accordent, plus il est proche de 1. Pour mesurer la performance, nous retenons le nombre de nœuds de l'arbre de recherche.

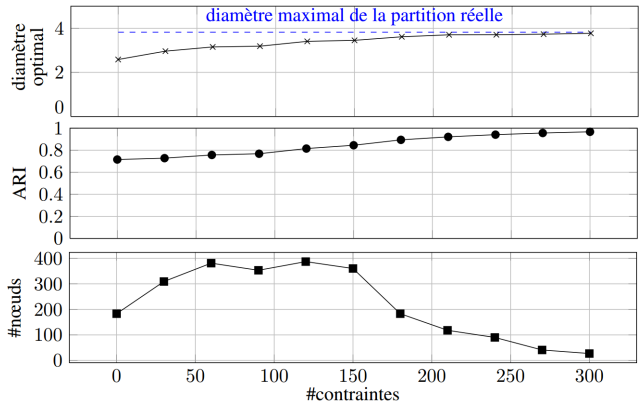

Figure 1. Influence des contraintes utilisateur sur la base Iris

Nous considérons la minimisation de diamètre maximal et des contraintes must-link (ML) et cannot-link (CL) sur la base Iris. Les contraintes sont générées suivant

la méthode utilisée dans (Wagstaff, Cardie, 2000). Deux points sont choisis au hasard dans la base de données, s'ils sont dans le même groupe dans P^*, une contrainte ML est générée, sinon ce sera une contrainte CL. Le nombre de contraintes varie de 0 à 300, qui correspond à 2,68 % des paires. Le test est répété 100 fois pour chaque nombre de contraintes et les valeurs présentées dans la figure 1 montrent la moyenne pour chaque cas. Les trois courbes indiquent la moyenne du diamètre optimal, de l'indice ARI, et du nombre de nœuds de l'arbre de recherche. Les deux premières courbes montrent une amélioration de la qualité des solutions avec des contraintes utilisateur. L'ajout de contraintes ML/CL n'implique pas toujours une réduction de l'espace de recherche, comme le montre la dernière courbe. On constate que le nombre de nœuds de l'arbre de recherche augmente lorsque le nombre de contraintes varie entre 30 et 150. Ceci s'explique par le fait qu'avec les contraintes ML/CL, le diamètre optimal augmente, comme illustré par la première courbe. La propagation des contraintes de diamètre est donc moins efficace tandis que la propagation des contraintes ML/CL n'est pas encore suffisante. Lorsqu'il y a plus de 200 contraintes ML/CL, la propagation de ces contraintes devient plus importante, ce qui permet de réduire l'espace de recherche.

6.3. Analyse de la robustesse du modèle

Nous évaluons la robustesse du modèle par rapport aux bornes inférieure k_{min} et supérieure k_{max}. Les expérimentations sont réalisées avec le critère de diamètre maximal, sans contraintes utilisateur et pour toutes les bases. On sait que le critère de diamètre favorise un grand nombre de clusters, puisque plus il y a de clusters dans une partition, plus le diamètre maximal est petit. Par conséquence, la solution optimale est toujours obtenue avec k_{max} clusters. Cependant pour tester la robustesse de la méthode, deux cas sont considérés, faisant varier soit k_{min}, soit k_{max}. Dans le premier cas, $k_{max} = 10$ et k_{min} varie de 2 à 10. Nous constatons que pour toutes ces bases, le nombre de nœuds de l'arbre de recherche ne change pas lorsque k_{min} varie. En effet, grâce à la propagation des contraintes pour le critère de diamètre, après avoir trouvé une solution avec k_{max} clusters, le solveur peut détecter qu'il n'existe pas de meilleure solution avec moins de k_{max} clusters.

Dans le deuxième cas, $k_{min} = 2$ et k_{max} varie de 2 à 10. Le nombre de nœuds des arbres de recherche est reporté dans la figure 2. En général, quand k_{max} augmente, il y a plus de partitions à prendre en considération. Cependant, la figure 2 montre un comportement intéressant : dans la plupart des bases, le nombre de nœuds de l'arbre de recherche n'augmente pas systématiquement. En effet, plus k_{max} est grand, plus le diamètre optimal est petit et la propagation des contraintes de diamètre est plus efficace pour réduire l'espace de recherche. Par conséquence, dans certains cas, l'algorithme prend moins de temps quand k_{max} augmente. Quant au temps d'exécution pour le cas où $k_{min} = 2$ et $k_{max} = 10$, la base *User Knowledge Modeling* dont l'arbre de recherche est le plus grand nécessite 4 499 secondes, la base *Image Segmentation* avec le plus grand nombre de points nécessite 944,5 secondes. Le temps varie entre

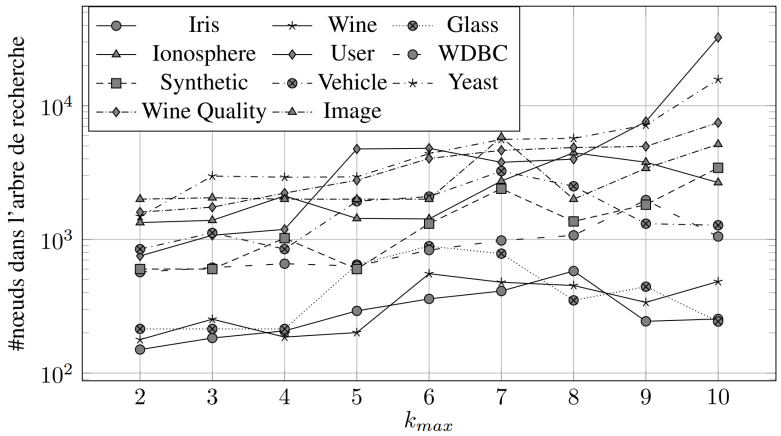

Figure 2. Taille de l'arbre de recherche avec $k_{min} = 2$ *et* $2 \leq k_{max} \leq 10$

moins de 0,1 secondes pour les trois premières bases et environ 500 secondes pour les bases *Yeast* et *Wine Quality*.

7. Conclusion

Dans ce papier, nous présentons un nouveau modèle basé sur la programmation par contraintes pour la classification non supervisée sous contraintes. Le modèle est générique dans le sens où un critère d'optimisation parmi différents critères peut être choisi et différent types de contraintes utilisateur peuvent s'intégrer directement. De plus, il offre à l'utilisateur la flexibilité de ne pas fixer le nombre de classes à l'avance, mais seulement des bornes sur le nombre de classes souhaité. Du fait que le modèle se base sur la dissimilarité entre objets, des données qualitatives ou quantitatives peuvent être considérées. Des expérimentations sur des bases de données classiques montrent que notre modèle est meilleur par rapport aux approches exacts existantes.

Des études sur des stratégies de recherche ou des algorithmes de filtrage appropriés améliorent considérablement l'efficacité du modèle. Nous continuons à étudier ces aspects afin de rendre le modèle capable de traiter des bases de données plus importantes. Du point de vue de la fouille de données, nous envisageons d'intégrer d'autres critères d'optimisation, comme par exemple le critère des moindres carrés.

Bibliographie

Aloise D., Deshpande A., Hansen P., Popat P. (2009). Np-hardness of euclidean sum-of-squares clustering. *Machine Learning*, vol. 75, n° 2, p. 245–248.

Aloise D., Hansen P., Liberti L. (2012). An improved column generation algorithm for minimum sum-of-squares clustering. *Mathematical Programming*, vol. 131, n° 1-2, p. 195-220.

Babaki B., Guns T., Nijssen S. (2014). Constrained clustering using column generation. In *Proceedings of the 11th international conference on integration of ai and or techniques in constraint programming for combinatorial optimization problems*, p. 438-454.

Basu S., Davidson I., Wagstaff K. (2008). *Constrained clustering: Advances in algorithms, theory, and applications* (1re éd.). Chapman & Hall/CRC.

Beldiceanu N., Carlsson M., Petit T., Régin J.-C. (2012). An O(nlog n) Bound Consistency Algorithm for the Conjunction of an alldifferent and an Inequality between a Sum of Variables and a Constant, and its Generalization. In *Proceedings of the 20th European Conference on Artificial Intelligence*, p. 145-150.

Berg J., Jarvisalo M. (2013). Optimal Correlation Clustering via MaxSAT. In *Proceedings of the 13th ieee international conference on data mining workshops*, p. 750-757.

Bilenko M., Basu S., Mooney R. J. (2004). Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of the 21st International Conference on Machine Learning*, p. 11–18.

Brusco M. (2003). An enhanced branch-and-bound algorithm for a partitioning problem. *British Journal of Mathematical and Statistical Psychology*, p. 83-92.

Brusco M. (2006). A repetitive branch-and-bound procedure for minimum within-cluster sum of squares partitioning. *Psychometrika*, p. 347-363.

Brusco M., Stahl S. (2005). *Branch-and-Bound Applications in Combinatorial Data Analysis (Statistics and Computing)* (1re éd.). Springer. Hardcover.

Dao T.-B.-H., Duong K.-C., Vrain C. (2013a). A Declarative Framework for Constrained Clustering. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, p. 419–434.

Dao T.-B.-H., Duong K.-C., Vrain C. (2013b). A Filtering Algorithm for Constrained Clustering with Within-Cluster Sum of Dissimilarities Criterion. In *Proceedings of the 25th International Conference on Tools with Artificial Intelligence*, p. 1060-1067.

Dao T.-B.-H., Duong K.-C., Vrain C. (2013c). Un modèle général pour la classification non supervisée sous contraintes utilisateur. In *Neuvième Journées Francophones de Programmation par Contraintes.*

Davidson I., Ravi S. S. (2005a). Agglomerative hierarchical clustering with constraints: Theoretical and empirical results. In *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, p. 59–70.

Davidson I., Ravi S. S. (2005b). Clustering with Constraints: Feasibility Issues and the k-Means Algorithm. In *Proceedings of the 5th SIAM International Conference on Data Mining*, p. 138–149.

Davidson I., Ravi S. S. (2007). The Complexity of Non-hierarchical Clustering with Instance and Cluster Level Constraints. *Data Mining Knowledge Discovery*, vol. 14, no 1, p. 25–61.

Davidson I., Ravi S. S., Shamis L. (2010). A SAT-based Framework for Efficient Constrained Clustering. In *Proceedings of the 10th SIAM International Conference on Data Mining*, p. 94-105.

Delattre M., Hansen P. (1980). Bicriterion Cluster Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no 4, p. 277-291.

De Raedt L., Guns T., Nijssen S. (2008). Constraint programming for itemset mining. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p. 204–212.

De Raedt L., Guns T., Nijssen S. (2010). Constraint Programming for Data Mining and Machine Learning. In *Proc. of the 24th AAAI Conference on Artificial Intelligence.*

Ester M., Kriegel H. P., Sander J., Xu X. (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, p. 226–231.

Garey M. R., Johnson D. S. (1979). *Computers and intractability: A guide to the theory of np-completeness*. W. H. Freeman and Company.

Gonzalez T. (1985). Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, vol. 38, p. 293–306.

Guns T., Nijssen S., De Raedt L. (2013). k-Pattern set mining under constraints. *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, n° 2, p. 402–418.

Han J., Kamber M., Pei J. (2011). *Data mining: Concepts and techniques* (3rd éd.). San Francisco, CA, USA, Morgan Kaufmann Publishers Inc.

Hansen P., Delattre M. (1978). Complete-link cluster analysis by graph coloring. *Journal of the American Statistical Association*, vol. 73, n° 362, p. 397-403.

Klein G., Aronson J. E. (1991). Optimal clustering: A model and method. *Naval Research Logistics*, vol. 38, n° 3, p. 447–461.

Lu Z., Carreira-Perpinan M. A. (2008). Constrained spectral clustering through affinity propagation. In *Proceedings of the 2008 IEEE Conference on Computer Vision and Pattern Recognition*, p. 1–8.

Métivier J., Boizumault P., Crémilleux B., Khiari M., Loudni S. (2012). A constraint language for declarative pattern discovery. In *Proceedings of the ACM symposium on applied computing*, p. 119–125.

Métivier J.-P., Boizumault P., Crémilleux B., Khiari M., Loudni S. (2012). Constrained Clustering Using SAT. In *Proceedings of the 11th International Symposium on Advances in Intelligent Data Analysis*, p. 207-218.

Mueller M., Kramer S. (2010). Integer Linear Programming Models for Constrained Clustering. In *Proceedings of the 13th International Conference on Discovery Science*, p. 159-173.

Petit T., Régin J.-C., Beldiceanu N. (2011). A $\theta(n)$ Bound-Consistency Algorithm for the Increasing Sum Constraint. In *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming*, p. 721-728.

Régin J.-C., Petit T. (2011). The Objective Sum Constraint. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, p. 190-195.

Régin J.-C., Rueher M. (2005). Inequality-sum: a global constraint capturing the objective function. *RAIRO - Operations Research*, vol. 39, n° 2, p. 123-139.

Rossi F., Beek P. van, Walsh T. (Eds.). (2006). *Handbook of Constraint Programming*. Amsterdam, Netherlands, Elsevier B.V.

Schmidt J., Brändle E. M., Kramer S. (2011). Clustering with Attribute-Level Constraints. In *ICDM*, p. 1206-1211.

Wagstaff K., Cardie C. (2000). Clustering with instance-level constraints. In *Proceedings of the 17th International Conference on Machine Learning*, p. 1103–1110.

Wagstaff K., Cardie C., Rogers S., Schrödl S. (2001). Constrained K-means Clustering with Background Knowledge. In *Proceedings of the 18th international conference on machine learning*, p. 577–584.

Wang X., Davidson I. (2010). Flexible constrained spectral clustering. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p. 563-572.