Une approche en programmation par contraintes pour la classification non supervisée

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

LIFO, Université d'Orléans, 45067 Orléans cedex 02, France
{thi-bich-hanh.dao, khanh-chuong.duong, christel.vrain}@univ-orleans.fr

Résumé. Dans cet article, nous abordons le problème de classification non supervisée sous contraintes fondé sur la programmation par contraintes (PPC). Nous considérons comme critère d'optimisation la minimisation du diamètre maximal des clusters. Nous proposons un modèle pour cette tâche en PPC et nous montrons aussi l'importance des stratégies de recherche pour améliorer son efficacité. Notre modèle basé sur la distance entre les objets permet de traiter des données qualitatives et quantitatives. Des contraintes supplémentaires sur les clusters et les instances peuvent directement être ajoutées. Des expériences sur des ensembles de données classiques montrent l'intérêt de notre approche.

1 Introduction

La problématique de classification non supervisée (aussi appelée clustering) a été longuement étudiée pendant de nombreuses années avec des approches comme k-means et k-médoïdes. En général, le problème consiste à partitionner un ensemble de n objets en k classes non vides et deux à deux disjointes. C'est un champ de recherche difficile pour plusieurs raisons : le choix de la mesure de dissimilarité entre les objets dépendant principalement de l'application mais influant fortement sur les résultats, la définition du critère à optimiser, la taille de l'espace de recherche avec pour conséquence la nécessité de définir des heuristiques conduisant souvent à un optimum local. Poser des contraintes sur la solution recherchée permet d'une part, de modéliser plus finement les applications réelles et d'autre part de restreindre la taille de l'espace de recherche. Néanmoins, la plupart des algorithmes classiques n'ont pas été développés pour la classification non supervisée sous contraintes et doivent être adaptés, si possible, pour prendre en compte les contraintes posées par l'utilisateur. Développer des solveurs généraux applicables à une grande variété de problèmes pose de nouveaux défis.

D'autre part, des avancées récentes en Programmation par Contraintes (PPC) ont rendu ce paradigme beaucoup plus puissant. Plusieurs travaux (De Raedt et al. (2008, 2010)) (Boizumault et al. (2011)) ont étudié l'intérêt de la PPC pour modéliser des problèmes de fouille de données et ont montré l'apport de la déclarativité inhérente à la PPC.

Dans ce papier, nous proposons un cadre pour modéliser la classification non supervisée sous contraintes en Programmation par Contraintes. L'intérêt de notre approche est de fournir un modèle déclaratif permettant de spécifier le problème de classification non supervisée et d'intégrer facilement des contraintes. Dans notre modèle, nous faisons l'hypothèse que nous

disposons d'une mesure de dissimilarité entre les paires d'objets et que le nombre k de classes est fixé (en théorie aucune limite n'est donnée sur la valeur de k, mais plus k est grand, plus la complexité est élevée). Nous considérons, dans ce papier, le problème de trouver une partition minimisant le diamètre maximal des classes et nous montrons que notre cadre intègre naturellement des contraintes sur les instances ou sur les classes.

Il est reconnu en PPC que le choix du modèle est fondamental, mais nous insistons aussi sur le fait que la stratégie de recherche est tout aussi importante pour améliorer l'efficacité. Plusieurs stratégies sont étudiées, différant sur la manière d'ordonner les points ou reposant sur des résultats théoriques. Des expérimentations sur des bases de données classiques montrent l'intérêt de notre approche. Contrairement à la plupart des travaux existants, notre modèle permet de trouver un optimum global, et nous ne pouvons espérer qu'il soit compétitif en terme de temps de calcul avec des méthodes heuristiques. Nous comparons la qualité des solutions obtenues par notre méthode avec celles obtenues par la méthode FPF (Gonzalez (1985)), méthode très efficace pour la classification non supervisée optimisant le diamètre maximum des classes, mais conduisant à une solution approchée.

Des travaux récents (Guns et al. (2011)), (Métivier et al. (2012)) ont déjà proposé d'utiliser la PPC pour la classification non supervisée conceptuelle. Le problème est formalisé comme la recherche d'un ensemble de k-motifs fréquents, deux à deux non recouvrants, dont l'ensemble couvre toutes les données : les transactions sont vues comme des objets et les motifs 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. Notons que ces approches sont donc adaptées à des 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. Davidson et al. (2010) propose un cadre SAT pour la classification non supervisée sous contraintes, mais uniquement pour un problème à 2 classes ($k = 2$) : il traite les contraintes sur les instances ("must-link" et "cannot-link") et des contraintes sur les classes (diamètre des clusters, séparation entre les clusters). Son algorithme converge vers un optimum global. Notre approche est plus générale dans la mesure où le nombre de classes n'est pas limité à 2.

Le papier est organisé comme suit. Dans la section 2, nous rappelons des notions sur la classification non supervisée à base de contraintes et sur la Programmation par Contraintes. La section 3 est dédiée à la présentation de notre modèle et la section 4 aux expérimentations. La conclusion et une discussion sur les travaux futurs sont données dans la section 5.

2 Préliminaires

2.1 Classification non supervisée

La classification non supervisée (ou clustering) consiste à regrouper les données dans des classes (ou clusters), de manière à regrouper dans un même cluster les données similaires et à séparer les données distantes dans des clusters différents. Etant donnés une base de données de n objets $\mathcal{O} = \{o_1, \ldots, o_n\}$ d'un espace $\mathcal{X}$ et une mesure de dissimilarité $d(o_i, o_j)$ entre deux objets o_i et o_j, l'objectif est de regrouper les objets en différentes classes de telle manière que la partition obtenue satisfasse un critère donné. Le problème de clustering peut être donc formulé comme un problème d'optimisation.

Les problèmes de clustering sont variés, dépendant de différents critères, comme la structure souhaitée (une partition, une hiérarchie, des classes recouvrantes, ...), ou encore le critère à optimiser. Dans ce papier, nous nous intéressons à la recherche d'une partition des objets en k classes $C_1, \ldots, C_k$ telle que : (1) pour tout i, $C_i \neq \emptyset$, (2) $\cup_i C_i = \mathcal{O}$, (3) pour tout i, j tel que $i \neq j$, $C_i \cap C_j = \emptyset$, et (4) un critère E est optimisé. Le critère optimisé peut être, entre autres :

- Critère des moindres carrés : $E = \sum_{c=1}^{k} \sum_{o_i \in C_c} d(m_c, o_i)^2$, où m_c est le centre de chaque cluster C_c.
- Critère de variance intra-classe : $E = \sum_{c=1}^{k} \sum_{o_i, o_j \in C_c} d(o_i, o_j)^2$. Ce critère est équivalent au critère des moindres carrés dans le cas de la distance euclidienne.
- Critère d'erreur absolue : $E = \sum_{c=1}^{k} \sum_{o_i \in C_c} d(o_i, r_c)$, où r_c est le représentant du cluster C_c.
- Critère de diamètre maximal : $E = max_{c \in [1,k], o_i, o_j \in C_c}(d(o_i, o_j))$. E est le diamètre maximal des clusters, où le diamètre d'un cluster est la distance maximale entre chaque paire de ces objets.

L'algorithme k-means représente chaque cluster par la moyenne des points du cluster et tend à minimiser le critère des moindres carrés. L'algorithme k-médoïdes choisit des objets pour représenter des clusters et cherche à minimiser le critère d'erreur absolue. A chaque itération, les deux méthodes réduisent la valeur du critère jusqu'à un optimum, en général local.

L'algorithme FPF (Furthest Point First) (Gonzalez (1985)) minimise le diamètre maximal des clusters. L'algorithme commence par choisir un point comme le représentant du premier cluster et affecte tous les points à ce cluster. A l'itération suivante, le point le plus loin du premier représentant est choisi comme le représentant du second cluster. Les points qui sont plus proches du second représentant que du premier sont réaffectés au second cluster. L'algorithme réitère de cette façon : choisir comme nouveau représentant le point le plus loin des représentants existants et réaffecter les points. Il s'arrête après k itérations, ayant ainsi formé k clusters. La complexité en temps est $O(nk)$. Si d_{opt} est l'optimum global alors l'algorithme garantit de trouver un clustering avec le diamètre maximal des clusters d tel que $d_{opt} \leq d \leq 2d_{opt}$, si la mesure utilisée satisfait l'inégalité triangulaire. De plus, Gonzalez a prouvé que trouver d tel que $d \leq (2 - \epsilon)d_{opt}$ est NP-Difficile pour tout $\epsilon > 0$, lorsque les points sont dans un espace à 3 dimensions. L'algorithme modifié (M-FPF) proposé dans (Geraci et al. (2006)) est plus rapid en temps mais la solution trouvée est identique que celle de FPF.

La classification non supervisée sous contraintes tient compte des contraintes définies par l'utilisateur pour identifier les clusters. Ces contraintes peuvent être posées sur les clusters ou sur des instances (points). Les contraintes sur les clusters imposent des conditions sur la forme, la taille ou d'autres caractéristiques. Par exemple, la contrainte sur la taille exprime que chaque cluster doit avoir un nombre minimum α de points (contrainte de capacité minimale) : $\forall c \in [1, k],\ |C_c| \geq \alpha$, ou encore que chaque cluster doit avoir un nombre maximum β de points (contrainte de capacité maximale) : $\forall c \in [1, k],\ |C_c| \leq \beta$. L'utilisateur peut aussi limiter le diamètre des clusters avec la contrainte de diamètre maximal : $\forall c \in [1, k], \forall o_i, o_j \in C_c,\ d(o_i, o_j) \leq \gamma$, avec γ un paramètre. Un autre exemple est la contrainte de séparation minimale, qui impose que la distance entre chaque paire d'objets de différents clusters soit supérieure à un seuil θ : $\forall c \in [1, k], \forall c' \neq c, \forall o_i \in C_c, o_j \in C_{c'}, d(o_i, o_j) \geq \theta$.

Les contraintes sur des instances sont en général posées sur des paires d'objets individuels. Deux types de contraintes sont souvent utilisés : "must-link" et "cannot-link". Une contrainte

must-link indique que deux objets o_i et o_j doivent être dans le même cluster : $\forall c \in [1,k],\ o_i \in C_c \Leftrightarrow o_j \in C_c$. Une contrainte cannot-link indique que deux objets ne peuvent pas appartenir au même cluster : $\forall c \in [1,k],\ \neg(o_i \in C_c \wedge o_j \in C_c)$.

Les contraintes définies par l'utilisateur sont nécessaires dans les applications réelles. Cependant, peu d'algorithmes sont adaptables pour gérer de telles contraintes. Il n'existe pas de solution générale pour étendre un algorithme traditionnel (k-means, k-médoïdes, etc.) avec des contraintes. Notre approche se basant sur la Programmation par Contraintes permet d'ajouter directement des contraintes définies par l'utilisateur, sans modifier le modèle.

2.2 Programmation par contraintes

La Programmation par Contraintes (PPC) est un paradigme puissant pour résoudre des problèmes combinatoires, se basant sur des techniques issues de l'intelligence artificielle ou de la recherche opérationnelle. La PPC se base sur le principe suivant : (1) le programmeur spécifie le problème d'une façon déclarative comme un problème de satisfaction de contraintes ; (2) le solveur cherche des solutions en intégrant la propagation de contraintes à la recherche. Un *problème de satisfaction de contraintes* (Constraint Satisfaction Problem – CSP) est un triplet $\langle X, D, C\rangle$ où $X = \{x_1, x_2, ..., x_n\}$ est un ensemble de variables, $D = \{D_1, D_2, ..., D_n\}$ est un ensemble de domaines ($x_i \in D_i$), $C = \{C_1, C_2, ..., C_t\}$ est un ensemble de contraintes où chaque contrainte C_i est une condition sur un sous-ensemble de X.

Une solution d'un CSP est une affectation complète de valeur $a_i \in D_i$ à chaque variable x_i satisfaisant toutes les contraintes de C. Un *problème d'optimisation sous contraintes* (Constraint Optimization Problem – COP) est un CSP auquel est associée une fonction objectif. Une solution optimale d'un COP est une solution du CSP qui optimise la fonction objectif.

En général, les CSP sont NP-Difficiles. Cependant, les techniques utilisées par les solveurs permettent de résoudre un grand nombre d'applications réelles de façon efficace. Les techniques les plus connues reposent sur la propagation de contraintes et des stratégies de recherche.

La propagation de contraintes consiste, pour une contrainte c, à supprimer du domaine des variables de c des valeurs pour lesquelles on peut déterminer qu'elles ne peuvent participer à une solution de c. A chaque contrainte est associé un ensemble de propagateurs, dépendant du choix de consistance pour cette contrainte. Par exemple, si la contrainte c est posée avec la consistance d'arc, les propagateurs sont implantés de manière à supprimer toutes les valeurs inconsistantes avec c du domaine des variables. Si c est posée avec la consistance de borne, les propagateurs modifient seulement les bornes inférieures et supérieures des domaines de variables. Le choix de la consistance est indiqué par le programmeur lorsque les contraintes sont posées.

Le fait que chaque contrainte soit réalisée par un ensemble de propagateurs implique que toute formule ou relation mathématique ne peut être une contrainte, seules celles pour lesquelles on peut construire un ensemble de propagateurs sont disponibles. Nous disposons des relations arithmétiques, logiques et de relations plus complexes représentées sous forme des contraintes globales.

Le solveur recherche des (les) solutions en itérant deux étapes : propagation des contraintes et branchement. Le solveur propage toutes les contraintes jusqu'à un état stable, dans lequel soit le domaine d'une variable est réduit à l'ensemble vide, soit aucun domaine ne peut se réduire davantage. Dans le premier cas, il n'existe pas de solution et le solveur effectue un

retour en arrière. Dans l'autre cas, si tous les domaines sont singletons, une solution est trouvée, sinon le solveur choisit une variable dont le domaine est non singleton et découpe le domaine en deux parties, ce qui crée deux nouvelles branches dans l'arbre de recherche. Le solveur explore ensuite chaque branche, la propagation de contraintes peut devenir de nouveau active suite aux modifications du domaine d'une variable.

La stratégie de l'exploration de l'arbre de recherche peut être déterminée par le programmeur. Avec la stratégie de recherche en profondeur d'abord, le solveur ordonne les branches par l'ordre spécifié par le programmeur et explore en profondeur chaque branche. Pour un problème d'optimisation, la stratégie de recherche en profondeur devient la stratégie *branch-and-bound* : chaque fois qu'une solution du CSP est trouvée, la valeur de la fonction objectif sur cette solution est calculée et une nouvelle contrainte est ajoutée, imposant qu'une nouvelle solution soit meilleure que celle-ci. Le solveur effectue une recherche exhaustive, la solution retournée est donc garantie d'être optimale. Les choix de variables et de valeurs à chaque branchement sont extrêmement importants, car ils peuvent aider à réduire de façon drastique l'arbre de recherche. Pour plus de détails sur la programmation par contraintes, le lecteur est invité à consulter l'ouvrage de Rossi et al. (2006).

Afin d'illustrer un problème d'optimisation sous contraintes et les stratégies de recherche, considérons l'exemple suivant.

Exemple 1 *Trouver une affectation de lettres aux chiffres telle que*

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & S & T \\ \hline = & M & O & N & E & Y \end{array}$$

et maximisant la valeur de MONEY. Ce problème peut se modéliser comme un COP, avec huit variables S, E, N, D, M, O, T, Y*, dont le domaine est l'ensemble des chiffres* $[0, 9]$*. Les contraintes spécifiant ce problème sont :*

- *les lettres* S *et* M *ne doivent pas être 0 :* $S \neq 0,\ M \neq 0$
- *les lettres sont deux à deux différentes :* $\texttt{alldifferent}(S, E, N, D, M, O, T, Y)$ *(Notons qu'ici au lieu de poser les contraintes* $\neq$ *pour chaque paire de variables, on utilise cette seule contrainte, qui porte sur l'ensemble des variables. Cette contrainte est appelée "contrainte globale" dans la PPC, comme la contrainte linéaire suivante.)*
- $(1000S + 100E + 10N + D) + (1000M + 100O + 10S + T) = 10000M + 1000O + 100N + 10E + Y$
- *maximise*$(1000M + 1000O + 100N + 10E + Y)$

La solution optimale est l'affectation $S = 9, E = 7, N = 8, D = 2, M = 1, O = 0, T = 4, Y = 6$*, avec* $MONEY = 10876$*. La propagation initiale de ces contraintes mène à un état stable avec les domaines* $D_S = \{9\}$*,* $D_E = \{2, 3, 4, 5, 6, 7\}$*,* $D_M = \{1\}$*,* $D_O = \{0\}$*,* $D_N = \{3, 4, 5, 6, 7, 8\}$ *et* $D_D = D_T = D_Y = \{2, 3, 4, 5, 6, 7, 8\}$*. Les stratégies précisent le choix des variables et pour chaque variable choisie, le choix des valeurs. Si les variables sont choisies dans l'ordre* S, E, N, D, M, O, T, Y *et pour chaque variable, les valeurs du domaine restant sont choisies dans l'ordre croissant, l'arbre de recherche se compose de 29 états et 7 solutions intermédiaires (solutions meilleures que la précédente mais non optimales). Si les variables sont choisies dans l'ordre* S, T, Y, N, D, E, M, O*, l'arbre de recherche a seulement 13 états et 2 solutions intermédiaires. Les deux arbres de recherche*[1] *sont donnés dans Figure 1. Ici,*

1. Ces arbres de recherche sont générés par l'environnement Gist du solveur Gecode.

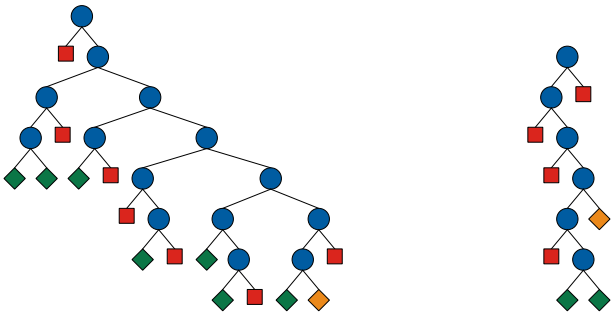

FIG. 1 – *Arbre de recherche*

un cercle bleu est un état stable mais non encore une solution, un carré rouge est un état échec (pas de solution), un losange vert est une solution intermédiaire et le losange orange est la solution optimale. Pour chaque état stable, la branche à gauche est le cas où la variable choisie reçoit la valeur choisie, la branche à droite est l'autre cas, où la valeur choisie est supprimée du domaine de la variable.

3 Modélisation des problèmes de clustering sous contraintes

Nous présentons dans cette section un modèle en PPC pour le clustering sous contraintes. Nous disposons d'une collection de n points (objets) et d'une mesure de dissimilarité entre ces objets. Sans perte de généralité nous supposons que les points sont indexés et nommés par leur indice. La distance entre deux points i, j est notée $d(i, j)$. Nous considérons le cas où le nombre k de clusters est connu à l'avance. Le modèle a pour objectif de trouver une partition des points en k clusters minimisant le diamètre maximal des clusters.

3.1 Modèle

Variables Nous fixons pour chaque cluster un représentant, qui est le point de plus petit indice. Nous introduisons donc une variable entière $I[c]$ pour chaque $c \in [1, k]$, qui donne le représentant du cluster c. Le domaine de chaque $I[c]$ est l'intervalle $[1, n]$, car chaque point peut être représentant.

Affecter un point à un cluster devient associer ce point au représentant du cluster. Nous introduisons pour chaque point $i \in [1, n]$ une variable entière $G[i]$ qui donne la valeur du représentant associé.

Nous introduisons également une variable D qui représente le diamètre maximal. C'est une variable entière, car nous utilisons des solveurs sur des variables entières. Le domaine de D est l'intervalle formé par la distance minimale et la distance maximale entre chaque paire de points. Dans nos expérimentations, lorsque la distance n'est pas entière, elle est multipliée par 100 et seule la partie entière est conservée.

Notre modèle permet de trouver la solution optimale, c'est-à-dire une affectation complète des variables I, G et D qui satisfait les contraintes suivantes.

Modélisation d'une partition La relation entre les points et leur cluster est exprimée par ces contraintes :

– Le représentant d'un représentant est lui-même :

$$\forall c \in [1,k],\ G[I[c]] = I[c].$$

– Le représentant d'un point doit être parmi les représentants : la valeur de chaque $G[i]$ doit être présente une fois dans I

$$\forall i \in [1,n],\ \ \#\{c \mid I[c]{=}G[i]\} = 1.$$

– Le représentant doit être d'indice minimal : $\forall i \in [1,n],\ \ G[i] \leq i$.

Un même ensemble de clusters pourrait être représenté de différentes manières, selon l'ordre des clusters. Les contraintes suivantes permettent d'éviter cette symétrie :

– Les représentants sont en ordre croissant : $\forall c < c' \in [1,k],\ I[c] < I[c']$.

– Le représentant du premier cluster est le premier point : $I[1] = 1$.

Contraintes portant sur le diamètre maximal :

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

$$d(i,j) > D \rightarrow (G[j] \neq G[i] \wedge\ G[j] \neq i) \tag{1}$$

– Le diamètre maximal doit être minimisé : minimise D.

Modélisation des contraintes définies par l'utilisateur L'avantage d'un modèle en PPC est d'être extensible à d'autres contraintes. Plusieurs contraintes définies par l'utilisateur, sur des clusters ou sur des points, peuvent s'y ajouter directement.

Pour les contraintes portant sur les clusters :

– La capacité minimale α des clusters : $\forall c \in [1,k],\ \ \#\{i \mid G[i]{=}I[c]\} \geq \alpha$.

– La capacité maximale β des clusters : $\forall c \in [1,k],\ \ \#\{i \mid G[i]{=}I[c]\} \leq \beta$.

– La contrainte de séparation minimale indique que la séparation entre deux clusters doit être au moins θ, ou que deux points à une distance inférieure à θ doivent être dans le même cluster. Pour chaque $i < j \in [1,n]$ tel que $d(i,j) < \theta$, nous posons la contrainte : $G[i] = G[j]$.

– La contrainte de diamètre maximum indique que le diamètre de chaque cluster doit être au plus γ, autrement dit que deux points à une distance supérieure à γ doivent être dans des clusters différents. Pour chaque $i < j \in [1,n]$ tel que $d(i,j) > \gamma$, nous posons les contraintes : $G[j] \neq G[i]$ et $G[j] \neq i$.

Pour les contraintes sur les couples de points :

– Une contrainte must-link sur i, j se traduit par le fait que les points ont le même représentant : $G[i] = G[j]$.

– Une contrainte cannot-link sur $i < j$ est exprimée par : $G[i] \neq G[j]$ et $G[j] \neq i$.

Stratégie de recherche Les stratégies de choix des variables et des valeurs doivent être données au solveur. Les variables sont choisies dans l'ordre I, D puis G. Cet ordre indique que les représentants de cluster doivent être identifiés en premier, puis pour chaque borne sur le diamètre maximal le solveur essaye de déterminer l'affectation de points aux clusters.

Les variables de I sont choisies de $I[1]$ à $I[k]$, c'est-à-dire du premier au dernier représentant. Puisqu'un représentant doit avoir l'indice minimal dans le cluster, les valeurs pour chaque $I[c]$ sont choisies dans l'ordre croissant.

Pour la variable D, nous utilisons l'idée de la recherche dichotomique. A chaque point de choix pour D, le domaine restant $[b_{min}, b_{max}]$ est découpé en deux parties $[b_{min}, \lceil (b_{min} + b_{max})/2 \rceil]$ et $[\lceil (b_{min} + b_{max})/2 \rceil + 1, b_{max}]$. Puisque nous cherchons à minimiser D, la partie inférieure est considérée en premier.

Les variables de G sont choisies dans l'ordre croissant sur la taille du domaine restant. Pour le choix de valeur pour chaque $G[i]$, l'indice du représentant le plus proche est choisie en premier.

3.2 Améliorations du modèle

Les solveurs de PPC réalisant une recherche exhaustive, ce modèle permet de trouver la solution optimale. Afin d'améliorer l'efficacité du modèle, différents aspects sont considérés.

Amélioration de la stratégie de recherche en réordonnant les points Afin d'identifier les représentants, les variables de I sont choisies dans l'ordre $I[1], \ldots, I[k]$. Puisque le représentant doit avoir l'indice minimal, pour chaque variable, les valeurs (indices possibles) sont choisies dans l'ordre croissant. L'indice des points a donc une grande importance. Par conséquent, les points sont réordonnés de façon à ce que ceux qui sont plus probables d'être représentant aient un indice plus faible. Nous présentons deux versions utilisant des heuristiques pour réordonner les points : l'heuristique de poids (version 1.1) et l'heuristique FPF (version 1.2).

La version 1.1 repose sur un ordonnancement préalable de points où : le premier point est loin des autres points, le deuxième point est loin du premier mais aussi assez loin des autres points, le troisième point est loin des deux premiers et aussi assez loin des restes, ...Pour réaliser cette heuristique, supposons que les p premiers points soient identifiés, le $(p+1)$-ème point est le $\arg\max_{i \in [p+1,n]} f(i)$, où

$$f(i) = \frac{\sum_{j \in [1,p]} d(i,j)}{p} \frac{\min_{j \in [1,p]} d(i,j)}{\max_{j \in [1,p]} d(i,j)} + \frac{\sum_{j \in [p+1,n]} d(i,j)}{n-p}$$

La version 1.2 ordonne les points avec l'algorithme FPF, en prenant $k = n$ (autant de classes que de points). Ainsi, chaque point est choisi par l'algorithme FPF et l'ordre de choix donne l'ordre des points.

Amélioration des contraintes Avec un k fixé et sans connaissances de contraintes utilisateur, 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 pour la variable D, à savoir $[d_{FPF}/2, d_{FPF}]$. De plus, pour chaque couple de points i, j :

- si $d(i,j) < d_{FPF}$, nous ne posons pas la contrainte réifiée (1) sur i, j,
- si $d_{FPF}/2 \leq d(i,j) \leq d_{FPF}$, nous posons la contrainte (1) sur i, j,
- si $d(i,j) > d_{FPF}$, la contrainte (1) sur i, j est remplacée par une contrainte cannot-link : $G[j] \neq G[i]$ et $G[j] \neq i$.

Base de données	#objets	#attributs	#classes
iris	150	4	3
ionosphere	351	34	2
kdd_synthetic_control	600	60	6
vehicle	846	18	4
yeast	1484	8	inconnu

TAB. 1 – *Les bases de données*

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. Les versions 2.1 et 2.2 sont développées avec cette méthode à partir des versions 1.1 et 1.2, respectivement.

4 Expérimentations

A cause de la complexité de la classification non supervisée, la plupart des algorithmes classiques se contentent d'un optimum local. De plus, la performance de ces méthodes diminue souvent lors de l'ajout de contraintes définies par l'utilisateur. Notre modèle garantit de trouver un optimum global et est directement extensible à des contraintes utilisateur. Dans cette section, nous présentons des expérimentations sur différentes bases de données, une comparaison entre l'optimum global et l'optimum local trouvé par l'algorithme FPF et des expérimentations dans le cas où des contraintes utilisateur sont ajoutées.

Nous avons réalisé notre modèle avec le solveur Gecode version 3.7.3 [2]. Les expérimentations sont réalisées sur un processeur 2.4GHz Core i5 Intel sous Ubuntu 12.04. Nous considérons les 5 bases de données dont les caractéristiques sont présentés dans Tableau 1.

Clustering Les quatre versions 1.1, 1.2, 2.1 et 2.2 sont implantées. Les variables sont choisies dans l'ordre I, D, G. Figure 2 représente le temps en secondes pris par les différentes versions pour chaque base. L'absence de temps signifie que la limite de temps de 10 minutes est dépassée. L'heuristique FPF utilisée dans les versions 1.2 et 2.2 donne en général une meilleure performance sauf pour les bases *ionosphere* et *yeast*. L'utilisation des bornes sur le diamètre dans les contraintes rend les versions 2.1 et 2.2 plus efficaces. On peut voir que les versions 1.x ne peuvent pas traiter *kdd_synthetic_control* et *vehicle* avec $k > 6$ ou *yeast* avec $k > 1$, tandis que les versions 2.x peuvent traiter des valeurs de k plus grandes.

Qualité de solution L'algorithme FPF est très rapide (complexité $O(nk)$) mais il retourne une solution d telle que $d_{opt} \leq d \leq 2d_{opt}$, avec d_{opt} la solution optimale. Cette borne est stricte (Gonzalez (1985)) : satisfaire $d \leq (2 - \epsilon)d_{opt}$ pour tout $\epsilon > 0$ est NP-Difficile dans l'espace à 3 dimensions. Notre modèle qui trouve l'optimum global nécessite évidemment beaucoup plus de temps. Cependant, pour voir l'intérêt de trouver une solution optimale, nous comparons la qualité des solutions dans Figure 3. On observe qu'avec un nombre assez faible de clusters, la solution de l'algorithme FPF est assez comparable. Par contre, lorsque ce nombre augmente, la différence entre les solutions devient plus significative.

2. http ://www.gecode.org

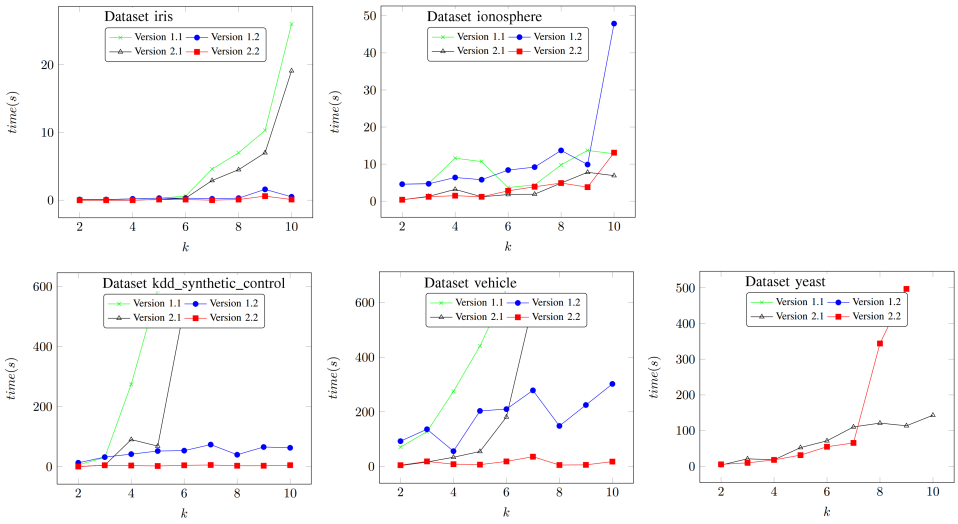

FIG. 2 – *Clustering à base de distance*

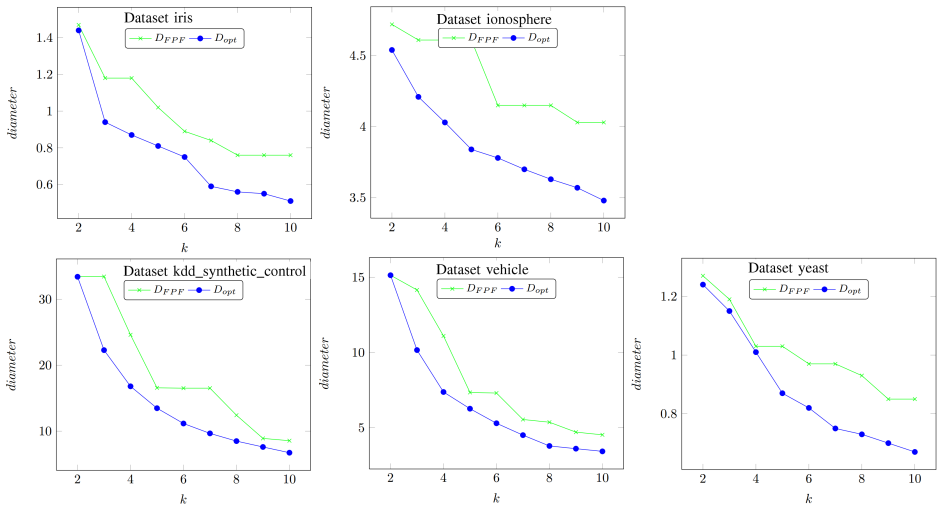

FIG. 3 – *Qualité de solutions*

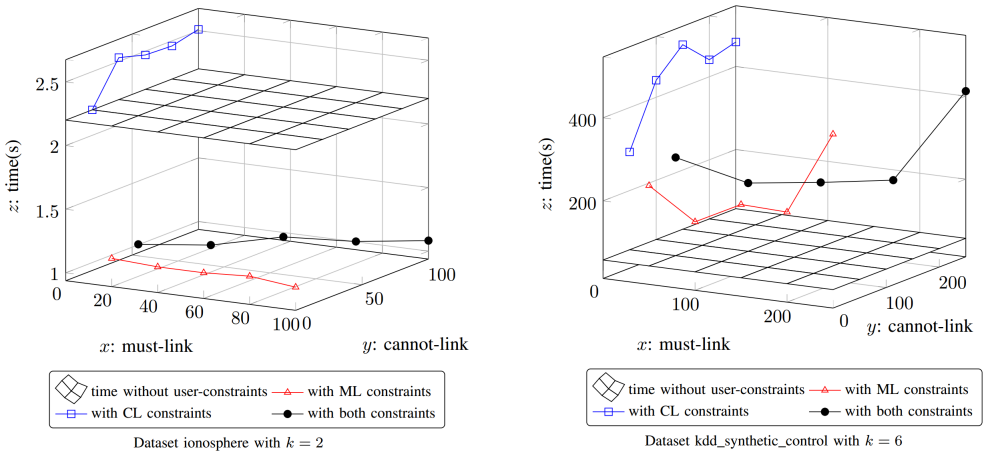

FIG. 4 – *Clustering avec contraintes d'utilisateur*

Clustering sous contraintes Un autre avantage de notre modèle est qu'il est extensible directement avec des contraintes utilisateur. Des expérimentations avec des contraintes d'instances ont été réalisées. Dans des bases où l'on connaît à l'avance la classe des objets, des contraintes d'instance peuvent être générées de cette façon : nous choisissons aléatoirement plusieurs paires d'objets et posons des contraintes must-link (ML) ou cannot-link (CL) en fonction de leurs classes. Des expérimentations avec la version 1.2 sont faites pour différentes combinaisons de contraintes ML et CL. Pour chaque combinaison, les tests sont effectués 10 fois. Le temps moyen dans le cas avec contraintes utilisateur et le temps effectif sans ces contraintes sont présentés dans Figure 4.

Les résultats montrent que pour la base *ionosphere*, les contraintes ML aident en général à réduire l'espace et le temps de recherche, ce qui n'est pas le cas avec les contraintes CL. Cependant pour la base *kdd_synthetic_control*, les contraintes utilisateur diminuent la performance, car la différence entre les diamètres optimaux dans les cas avec ou sans contraintes est souvent significative. Le diamètre optimal dans le cas avec contraintes est souvent deux fois plus grand que celui sans contraintes. Nous pensons que le critère de diamètre maximal pourrait ne pas être le plus approprié pour la structure réelle de cette base, ce qui pourrait expliquer les résultats.

5 Conclusion

Nous présentons dans ce papier un modèle en programmation par contraintes pour la classification non supervisée. Le modèle trouve un optimum global qui minimise le diamètre maximal des clusters. Des données qualitatives ou quantitatives peuvent être considérées. Le modèle est extensible directement aux contraintes sur des clusters ou sur des instances. Nous présentons également des stratégies de recherche pour améliorer l'efficacité du modèle. Des expérimentations sur des jeux de données classiques montrent l'intérêt de notre approche.

Le modèle actuel considère le critère du diamètre maximal. Nous souhaitons renforcer le côté générique du modèle, le rendre extensible à d'autres critères comme la variance intra classe ou l'erreur absolue. Outre la généricité, l'amélioration de l'efficacité constitue un autre objectif des travaux futurs afin de passer à l'échelle. Un des aspects possibles sera d'étudier une propagation de contraintes efficace pour exploiter des contraintes utilisateur.

Enfin, nous souhaitons également étendre le cadre pour adresser d'autres tâches de clustering, comme par exemple la classification recouvrante.

Références

Boizumault, P., B. Crémilleux, M. Khiari, S. Loudni, et J.-P. Métivier (2011). Discovering Knowledge using a Constraint-based Language. *CoRR abs/1107.3407*.

Davidson, I., S. S. Ravi, et L. Shamis (2010). A SAT-based Framework for Efficient Constrained Clustering. In *SDM*, pp. 94–105.

De Raedt, L., T. Guns, et S. Nijssen (2008). Constraint programming for itemset mining. In Y. Li, L. Bing, et S. Sunita (Eds.), *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008,*, pp. 204–212. ACM.

De Raedt, L., T. Guns, et S. Nijssen (2010). Constraint Programming for Data Mining and Machine Learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Geraci, F., M. Pellegrini, M. Maggini, et F. Sebastiani (2006). Cluster Generation and Cluster Labelling for Web Snippets : a Fast and Accurate Hierarchical Solution. In *Proceedings of the 13th Int. Conf. on String Processing and Information Retrieval*, pp. 25–36.

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

Guns, T., S. Nijssen, et L. De Raedt (2011). k-Pattern set mining under constraints. *IEEE Transactions on Knowledge and Data Engineering*. Accepted.

Métivier, J.-P., P.Boizumault, B. Crémilleux, M. Khiari, et S. Loudni (2012). Clustering sous constraintes utilisant SAT. *JFPC*.

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

Summary

In this paper, we address the problem of constraint-based clustering based on Constraint Programming (CP). We consider the problem of optimizing the maximum diameter of the clusters. We propose a model for this task in CP and we also show the importance of search strategies for improving the efficiency. Our model is based on the distance between objects, thus allowing to consider qualitative and quantitative data. Cluster-based and instance-based constraints can easily be added. Experiments on classical datasets show the interest of our approach.