Ensemble de correction minimal basé sur l'adéquation des contraintes en clustering

Aymeric Beauchamp[1], Thi-Bich-Hanh Dao[1], Samir Loudni[23], Christel Vrain[1]

[1]Université d'Orléans, INSA CVL, LIFO UR 4022
[2]IMT Atlantique Bretagne-Pays de la Loire, TASC (LS2N-CNRS)
[3]Université de Caen-Normandie, GREYC

[1]{aymeric.beauchamp, thi-bich-hanh.dao, christel.vrain}@univ-orleans.fr
[2] samir.loudni@imt-atlantique.fr

Résumé

Le clustering sous contraintes incrémentales voit un expert améliorer pas à pas une partition de départ par ajouts successifs de contraintes. Un conflit peut apparaître quand des contraintes ajoutées entrent en contradiction avec d'autres fournies aux itérations précédentes. Pour résoudre ce problème, nous proposons une approche hybride combinant apprentissage automatique et programmation par contraintes. Notre méthode consiste à entraîner un modèle sur les retours utilisateurs passés, pour calculer un score d'adéquation des contraintes à ce modèle. Cette mesure est ensuite employée comme pondération pour un problème d'optimisation sous contraintes afin d'identifier les contraintes divergeant le plus du retour utilisateur pour les enlever en priorité. Des tests sur des jeux de données et des cas pratiques d'utilisation montrent la pertinence de cette approche.

Mots-clés

Clustering sous contraintes incrémentales, programmation par contraintes, insatisfiabilité, ensemble de correction minimal

Abstract

Constraint-incremental clustering consists in letting an expert iteratively add constraints to improve an initial partition step-by-step. Conflicts can occur when a newly added constraint contradicts others given in previous iterations. We address this issue by introducing a hybrid approach combining machine learning and constraint programming (CP). Our method learns a model that scores the adequacy of each constraint with respect to past user feedback. These scores are then used in a CP model computing a minimal correction set, identifying the constraints whose removal would resolve the conflict while preserving those most aligned with user intent. Experiments on synthetic and real-world datasets demonstrate the efficiency and practical relevance of this guided correction mechanism.

Keywords

Constraint-incremental clustering, constraint programming, unsatisfiability, minimal correction sets

1 Introduction

La recherche en IA hybride - l'intégration de différents domaines de l'intelligence artificielle pour créer une synergie entre eux - suscite de plus en plus d'intérêt à l'heure où les demandes d'explicabilité des modèles et d'intégration de l'humain dans les systèmes utilisant l'IA sont des enjeux importants. Parmi les défis posés par ces approches, la gestion du bruit dans le retour utilisateur revêt une importance particulière. Qu'il provienne d'incohérences dans les spécifications de l'utilisateur ou d'interprétations ambiguës, l'expérience montre qu'elles sont inévitables en pratique. En programmation par contraintes (PPC), elles conduisent à des ensembles de contraintes insatisfiables nécessitant une relaxation pour arriver à une solution.

Le clustering est une tâche importante en fouille de données dont l'objectif est de partitionner un ensemble de données (objets) en groupes (clusters) d'une manière telle que les objets appartenant à un même cluster soient similaires mais différents de ceux appartenant aux autres clusters. Le clustering sous contraintes [8] a pour objectif de trouver des clusters plus pertinents en spécifiant, sous forme de contraintes, les propriétés requises. Cela permet de pallier dans une certaine mesure aux biais des algorithmes de clustering. Parmi les contraintes les plus communément utilisées figurent les contraintes *must-link/cannot-link* (ML/CL) imposant que des objets appartiennent (ou non) à un même cluster [21], ou que les clusters aient une taille minimale et/ou maximale [2]. Elles sont en général données par l'expert et modélisent ses connaissances a priori sur le résultat attendu.

Dans la pratique, l'expert peut éprouver des difficultés à spécifier dès le début du processus toutes les connaissances nécessaires à l'élaboration d'une partition intéressante. Une alternative est de lui proposer de réagir sur un résultat en cherchant à le corriger afin de suggérer des connaissances intéressantes à intégrer, conduisant ainsi à une nouvelle partition qui pourra être encore itérativement améliorée. Ceci constitue un processus interactif potentiellement source de

conflits : une nouvelle contrainte peut être incompatible avec une autre proposée plusieurs itérations auparavant, sans que l'utilisateur s'en rende compte.

Résoudre un conflit dans un ensemble de contraintes revient à identifier un sous-ensemble de contraintes à relâcher afin de trouver une solution. Afin de respecter au mieux les désirs de l'utilisateur, ce sous-ensemble doit préserver au mieux les informations déjà données tout en permettant la reprise du processus. Il doit aussi être possible d'expliquer à l'utilisateur comment le choix du sous-ensemble à relâcher a été effectué. Un système qui semble (du point de vue de l'utilisateur) ignorer arbitrairement une partie des informations données peut réduire la confiance dans le processus et décourager l'implication humaine.

Nous proposons une mesure d'adéquation des contraintes exploitant le contexte incrémental, décrivant l'accord entre une contrainte et un modèle d'apprentissage basé sur les retours utilisateurs antérieurs. Nous expliquons également comment utiliser cette mesure en PPC pour trouver un ensemble minimal de correction optimal d'après cette mesure. Nous donnons une implémentation de ces contributions dans un système de clustering sous contraintes incrémentales composé d'un modèle PPC de modification minimale de partition et d'un algorithme de clustering actif reposant sur un arbre de décision.

Dans la section 2, nous passons en revue des travaux antérieurs liés à l'identification de sous-ensembles de contraintes insatisfiables, et au clustering sous contraintes incrémentales. Dans la section 3, nous clarifions le contexte du clustering sous contraintes incrémentales dans lequel s'inscrivent nos contributions. Dans la section 4, nous décrivons la mesure d'adéquation et son utilisation pour le calcul de MCS. Dans la section 5, nous présentons les résultats expérimentaux sur les aspects critiques de cette approche.

2 Travaux antérieurs

2.1 Contraintes et insatisfiabilité

Le problème de satisfiabilité d'un ensemble de contraintes est bien établi en programmation par contraintes et plus largement dans les paradigmes déclaratifs. La recherche sur le sujet est allée au-delà du simple diagnostic d'insatisfiabilité pour développer des méthodes de calcul d'ensembles insatisfiables minimaux (MUS), qui aident à comprendre les causes des conflits en identifiant des combinaisons problématiques de contraintes. De plus, des ensembles de correction minimaux (MCS) peut aussi être trouvés ; ils représentent le plus petit ensemble de contraintes à enlever pour rendre le reste satisfiable. Parmi les approches notables, on peut citer CAMUS [14] qui effectue des résolutions successives de problèmes MaxSAT pour énumérer des MCS pour ensuite en déduire des MUS. L'algorithme MARCO [13] parcourt des formules booléennes représentant les différentes parties de l'ensemble de contraintes. Chaque nouvelle formule est testée pour savoir sa satisfiabilité, puis étendue ou rétrécie pour arriver à un MSS ou un MUS selon le résultat. QUICKXPLAIN [9] sélectionne un MUS en prenant en compte un ordre donné sur les contraintes pouvant refléter des préférences utilisateur. Il fonctionne par recherche dichotomique en respectant l'ordre établi sur les contraintes.

2.2 Clustering sous contraintes incrémentales

Il existe un effort de recherche grandissant dans ce que nous appelons le clusteing sous contraintes incrémentale. On peut donner comme point d'origine [6], qui introduit l'idée de récupérer des retours utilisateur de façon itérative à partir d'un résultat existant, au lieu de demander à définir l'ensemble des contraintes à utiliser en une seule fois. Par la suite, d'autres travaux reprendront et étendront ce concept. [19] décrit une méthode basée sur une matrice d'utilité qui représente la valeur de décisions prises selon des attributs catégoriels (par exemple, choisir une place de parking en fonction du bâtiment dans lequel on travaille). Une même décision a donc une valeur différente selon la catégorie à laquelle elle est appliquée. L'objectif est de faire coïncider du mieux possible les clusters avec les catégories qui maximisent une décision particulière. Pour cela, on prend une décision pour chaque cluster de la partition courante et on calcule la distribution des utilités des points selon la décision prise. Des contraintes ML sont générées entre les points qui ont les utilités les plus élevées dans le même cluster, et des contraintes CL sont ajoutées entre les points d'utilités très différentes. On calcule alors une nouvelle partition à l'aide de ces nouvelles contraintes avec COPKMEANS [21]. Dans [16], l'algorithme INCREMENT découpe une partition de départ en sous-clusters et en extrait un représentant de chaque. L'utilisateur étiquette chaque représentant, puis un encodeur apprend un espace sémantique en se basant sur les étiquettes généralisées à l'ensemble des sous-clusters. Enfin, INCREMENT partitionne le nouvel espace en fonction du nombre d'étiquettes distinctes avec KMEANS, ce qui permet de modifier le nombre de clusters de départ tout en propageant les étiquettes dans le jeu de données.

Du côté des approches déclaratives, [10] présente un modèle en programmation linéaire à nombre entiers (PLNE) qui effectue des modifications au niveau des clusters telles que la réduction du diamètre d'un cluster. Une autre approche en PLNE [18] intègre des contraintes en post-traitement à une matrice d'allocation probabiliste des points aux clusters. Le PLNE produit une partition en assignant à chaque point son cluster le plus proche en prenant en compte les contraintes utilisateur de divers types : ML/CL, triplet, taille des clusters, contrainte sur les voisins, les attributs... Ce modèle peut également produire des clusters chevauchants en permettant à un point d'appartenir à plusieurs clusters en même temps.

Plus récemment, [11] présente un système de clustering collaboratif sous contraintes appelé I-SAMARAH. Plusieurs agents calculent différentes partitions d'un jeu de données, puis on identifie les conflits entre les partitions pour les modifier. Quand tous les conflits ont été résolus, un résultat final est désigné par vote. Les contraintes ML/CL sont inté-

grées dans le critère global de qualité de I-SAMARAH, et de nouvelles contraintes peuvent être ajoutées après l'unification. Les contraintes sont dispersées entre les différents agents et servent à provoquer des différences par la création de nouveaux clusters, pour créer de nouveaux conflits et contrebalancer la tendance du système à l'homogénéisation des agents.

L'exploitation de contraintes déjà fournies a été explorée en clustering actif pour générer de nouvelles contraintes à faire valider par l'expert, qui ne sont pas redondantes avec les contraintes connues [22, 1]. Par ailleurs, [20] présente une approche bayésienne pour estimer le meilleur sous-ensemble de contraintes d'un ensemble insatisfiable de contraintes *must-link*/*cannot-link*. Enfin, [7] décrit deux mesures de qualité d'un ensemble de contraintes : l'informativité par rapport à un algorithme de clustering, c'est-à-dire le taux de contraintes qui ne sont pas immédiatement satisfaites par l'exécution de l'algorithme ; la cohérence des contraintes entre elles selon leur projection de l'une sur l'autre dans un espace métrique, qui estime la redondance de l'ensemble.

3 Clustering sous contraintes incrémentales

Soit $\mathcal{D}$ un jeu de données de n points décrits par m attributs. Le problème du clustering consiste à chercher une **partition** de $\mathcal{D}$ en k clusters de points similaires entre eux et différents des points des autres clusters. Dans le contexte de clustering sous contraintes incrémentales, des informations sur la partition voulue ou des relations entre les points sont exprimées sous forme de contraintes. Ces contraintes sont ajoutées itérativement ; on obtient à chaque itération un nouveau lot de contraintes à utiliser. L'ensemble $\mathcal{C}_t$ contient les contraintes employées à l'itération t pour modifier la partition courante. $\mathcal{C}_t$ peut contenir des contraintes issues des lots précédents. Le lot L_t contient les contraintes obtenues à l'itération t par un algorithme de clustering actif proposant des contraintes informatives et/ou par ajout direct de l'utilisateur. De façon générale, le processus peut être résumé par les étapes suivantes :

1. calculer une partition de départ $\mathcal{P}$;
2. demander à l'expert un lot de contraintes d'après $\mathcal{D}$ et $\mathcal{P}$;
3. modifier $\mathcal{P}$ à l'aide de l'ensemble de contraintes $\mathcal{C}_t$ comprenant le lot de contraintes obtenu L_t et tout ou partie des contraintes antérieures ;
4. répéter les étapes 2 et 3 jusqu'à satisfaction de l'expert ou épuisement des contraintes possibles.

De nombreuses contraintes pour le clustering ont été développées, portant sur les relations entre certains points ou sur des propriétés des clusters eux-mêmes. En l'occurrence, nous nous concentrons sur des contraintes portant sur des paires, des triplets ou des groupes de points.

Définition 3.1 (Contrainte *must-link* [21])**.** *Soit $\mathcal{P}$ une partition d'un jeu de données $\mathcal{X}$. Une contrainte must-link entre deux points x_i et x_j de $\mathcal{X}$ indique qu'ils doivent appartenir au même cluster de $\mathcal{P}$, formellement :*

$$ML(x_i, x_j) \stackrel{def}{=} \forall\, C_p \in \mathcal{P},\ x_i \in C_p \implies x_j \in C_p, \quad (1)$$

Définition 3.2 (Contrainte *cannot-link*)**.** *Soit $\mathcal{P}$ une partition d'un jeu de données $\mathcal{X}$. Une contrainte cannot-link entre deux points x_i et x_j de $\mathcal{X}$ indique qu'ils doivent appartenir à des clusters différents de $\mathcal{P}$, formellement :*

$$CL(x_i, x_j) \stackrel{def}{=} \forall\, C_p \in \mathcal{P},\ x_i \in C_p \implies x_j \notin C_p, \quad (2)$$

La fermeture transitive sur un réseau de contraintes ML/CL permet d'obtenir des contraintes supplémentaires selon trois règles [15] :

1. $ML(x_a, x_b) \wedge ML(x_b, x_c) \implies ML(x_a, x_c)$;
2. $ML(x_a, x_b) \wedge CL(x_b, x_c) \implies CL(x_a, x_c)$;
3. pour un clustering binaire ($k = 2$), $CL(x_a, x_b) \wedge CL(x_b, x_c) \implies ML(x_a, x_c)$.

Définition 3.3 (Contrainte triplet)**.** *Soit $\mathcal{P}$ une partition d'un jeu de données $\mathcal{X}$. Une contrainte triplet entre trois points x_a, x_p et x_n de $\mathcal{X}$ indique que le point x_a est plus similaire à x_p qu'à x_n. Cela implique les relations suivantes en fonction des affectations de $\mathcal{P}$:*

$$Triplet(x_a, x_p, x_n) \stackrel{def}{=} x_p \notin C_a \implies x_n \notin C_a \quad (3)$$

avec C_a le cluster auquel est assigné x_a. Autrement dit, si x_a et x_n sont dans le même cluster, x_p doit également faire partie du cluster puisqu'un point moins similaire à x_a en fait partie.

Définition 3.4 (Plage de clusters d'un ensemble de points)**.** *Soit S un ensemble de points d'un jeu de données $\mathcal{X}$. La **plage** (span) de clusters de S pour une partition $\mathcal{P}$ est :*

$$Sp(S, \mathcal{P}) = \{C_p \mid x \in C_p \cap S\}\ \forall\, C_p \in \mathcal{P} \quad (4)$$

Exemple 3.1. *La plage de clusters de $\{x_1, x_2, x_3\}$ pour la partition $\{1, 1, 2\}$ est $Sp(\{x_1, x_2, x_3\}, \{1, 1, 2\}) = \{1, 2\}$.*

Définition 3.5 (Contrainte sur la plage de clusters)**.** *Soit $\mathcal{P}$ une partition d'un jeu de données $\mathcal{X}$ en k clusters. Une contrainte sur la plage de clusters [17] sur $S \subseteq \mathcal{X}$ restreint la plage de clusters $Sp(S, \mathcal{P})$. On distingue deux types de contrainte sur la plage de clusters. Une contrainte spécifique contraint la plage de S par rapport à un ensemble $C \subseteq [1, K]$ donné :*

$$SSpan(S, C) \stackrel{def}{=} Sp(S, \mathcal{P}) \subseteq C \quad (5)$$

Une contrainte sur la plage de clusters générique ne donne qu'une borne supérieure γ sur le nombre de clusters distincts dans S :

$$GSpan(S, \gamma) \stackrel{def}{=} |Sp(S, \mathcal{P})| \leq \gamma \quad (6)$$

Au fil des itérations, il peut arriver que l'expert donne des contraintes contradictoires avec d'autres reçues à une itération antérieure. Cela peut venir d'une simple erreur, ou d'un changement des attentes au cours du processus pouvant être provoqué par exemple par un déclic en observant les partitions successives. Dans tous les cas, il est nécessaire de résoudre le conflit afin de procéder à la modification et de ne pas bloquer le processus.

3.1 Cadre applicatif

Outre une description théorique, nous proposons une implémentation de notre modèle dans un cadre de clustering sous contraintes incrémentales incorporant un modèle PPC de modification minimale de clustering [3] et un algorithme de clustering actif basé sur les retours utilisateurs passés.

Algorithm 1 Clustering actif sous contraintes incrémentales

Input : Dataset $\mathcal{D}$, budget B
Output : partition $\mathcal{P}$

$t \leftarrow 0$
$\mathcal{C} \leftarrow \emptyset$
$\mathcal{P} \leftarrow initialClustering(\mathcal{D})$
$\mathcal{U}, sat \leftarrow userFeedback(\mathcal{P})$ ▷ Retour initial
while $sat == false$ **do**
  if $\mathcal{U} \neq \emptyset$ **then**
    $L_t \leftarrow \mathcal{U}$ ▷ Ajout de contraintes
  else
    $L_t \leftarrow activeSelection(\mathcal{D}, \mathcal{P}, B)$ ▷ [4]
  $\mathcal{C} \leftarrow \mathcal{C} \cup B_t$
  $\mathcal{C}_t \leftarrow constraintSet(\mathcal{C})$
  $\mathcal{P} \leftarrow modifyClustering(\mathcal{D}, \mathcal{P}, \mathcal{C}_t)$ ▷ [3]
  if $\mathcal{P} == \emptyset$ **then**
    $\mathcal{P} \leftarrow solveConflict(\mathcal{C}_t)$ ▷ Section 4
  $\mathcal{U}, sat \leftarrow userFeedback(\mathcal{P})$
  $t \leftarrow t + 1$
return $\mathcal{P}$

Ce cadre décrit par l'algorithme 1 repose sur une boucle reposant sur la réaction de l'expert à la partition qui lui est présentée. Il ou elle peut donner ses propres contraintes ou utiliser le clustering actif pour générer des contraintes potentielles à valider. Le lot de contraintes produit ainsi que les contraintes antérieures sont utilisées pour modifier la partition. Notre contribution se place dans le cas où la modification n'est pas réalisable en raison d'un conflit entre les contraintes choisies.

La modification minimale de clustering est faite via un modèle en PPC. La partition modifiée est représentée par des variables entières G_i de domaine $\{1, \ldots, k\}$ telle que $G_i = c$ signifie que le point i appartient au cluster c dans la partition modifiée.

$$\min \quad \sum_{i=1}^{N} \mathbb{I}(\mathcal{P}[i] \neq G_i)\, d(i, G_i) \tag{7}$$

s.t.

$$Z_j = 1 \iff c_j \tag{8}$$

$$\sum_{c \in \mathcal{C}} Z_c \geq \delta \cdot |\mathcal{C}| \tag{9}$$

avec c_j une contrainte quelconque parmi les types décrits auparavant (ML/CL, triplet ou sur la plage de clusters) et $d(i, G\_i)$ la distance du point i à un représentant du cluster G_i auquel il est affecté.

La fonction indicatrice $\mathbb{I}(\mathcal{P}[i] \neq X_i)$ retourne 1 si le numéro de cluster du $i^{\text{ème}}$ élément de la partition modifiée est différent du numéro de cluster à l'entrée du modèle, et 0 sinon.

Les contraintes décrites précédemment sont exprimées de la forme suivante :

— Les contraintes ML/CL sont exprimées par $G_i = G_j$ et $G_i \neq G_j$ respectivement.

— Une contrainte triplet peut s'exprimer avec une implication logique :

$$G_a = G_n \implies G_a = G_p \tag{10}$$

— Les contraintes sur la plage de clusters sont exprimés avec des contraintes globales. Concernant les contraintes spécifiques portant sur un groupe de points S et un ensemble C de clusters autorisés, des contraintes de domaine peuvent être employées :

$$G_i \in C,\ \forall\, i \in S \tag{11}$$

— Dans le cas des contraintes sur la plage de clusters génériques, on utilise la contrainte globale *atmost_nvalue* [1] [5] :

$$\textit{atmost\_nvalue}(\gamma, \{G_i \mid i \in S\}) \tag{12}$$

4 Résolution de conflit entre des contraintes de clustering

Pour rappel, un problème de satisfaction de contraintes est posé sur un ensemble de variables $\mathcal{V}$, chacune ayant un domaine de valeurs autorisées dans un ensemble $\mathcal{D}$, et un ensemble de contraintes $\mathcal{C}$ à satisfaire. Un solveur en programmation par contraintes alterne entre la recherche (choisir une variable et y assigner une valeur de son domaine) et la propagation de contraintes (réduire les domaines des autres variables en filtrant les valeurs qui ne permettent plus de satisfaire le problème) jusqu'à trouver une assignation de toutes les variables, solution du problème. Soit un problème de satisfaction de contraintes $\mathcal{T} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ tel que $\mathcal{C}$ est insatisfiable, c'est-à-dire qu'au moins un sous-ensemble $S \subseteq \mathcal{C}$ est tel que ses contraintes ne peuvent pas être toutes satisfaites en même temps. Nous cherchons à résoudre le conflit en identifiant des contraintes à enlever pour permettre la satisfaction de toutes les contraintes restantes. Différents sous-ensembles pertinents peuvent être calculés à partir de ce problème. Leurs définitions sont rappelées ci-dessous [14].

Définition 4.1 (Sous-ensemble insatisfiable minimal (MUS [2]))**.** *Soit S un sous-ensemble de $\mathcal{C}$. S est un sous-ensemble minimal non satisfiable si et seulement si :*

1. https://sofdem.github.io/gccat/gccat/Catmost_nvalue.html
2. De l'anglais *Minimal Unsatisfiable Subset*.

— *S est insatisfiable ;*
— $\forall\, c \in S, \{S \setminus c\}$ *est satisfaisable.*

Définition 4.2 (Sous-ensemble satisfaisable maximal (MSS [3]))**.** *Soit S un sous-ensemble de $\mathcal{C}$. S est un sous-ensemble satisfiable maximal si et seulement si :*
— *S est satisfiable ;*
— $\forall\, c \notin S, \{S \cup c\}$ *est insatisfiable.*

Définition 4.3 (Sous-ensemble de correction minimal (MCS [4]))**.** *Soit S un sous-ensemble de $\mathcal{C}$. S est un sous-ensemble de correction minimal **e** si et seulement si :*
— $\mathcal{C} \setminus S$ *est satisfiable ;*
— $\forall\, c \in S, \{\mathcal{C} \setminus \{S \setminus c\}\}$ *est insatisfiable.*
De façon équivalente, S est un MCS si et seulement si $\mathcal{C} \setminus S$ est un MSS.

Il y a donc trois façons d'obtenir un MCS : soit par construction depuis l'ensemble vide, soit par énumération des MUS du problème pour trouver un MCS dual, soit par calcul d'un MSS pour obtenir le complémentaire.

4.1 Adéquation d'une contrainte à un modèle

Il existe souvent de multiples façons de résoudre le conflit, de sorte que la recherche d'un MCS est un problème combinatoire. Si l'on recherche un seul ensemble répondant à des exigences pratiques, il faut définir une fonction à optimiser représentant les propriétés désirées. La cardinalité de l'ensemble est l'objectif par défaut dans ce genre de problèmes, mais il est également possible de pondérer les contraintes et chercher le MCS minimisant le poids des contraintes qui le constituent.
C'est dans ce but que nous présentons une mesure destinée à servir de fonction de pondération dans un problème d'optimisation sous contraintes. Cette mesure, que nous appelons adéquation d'une contrainte à un modèle, estime à quel point l'affectation des points d'une contrainte dans un modèle d'apprentissage supervisé entraîné sur des retours utilisateurs passés (étiquetage partiel du jeu de données, contraintes déjà fournies...) est en accord avec la définition de la contrainte. La raison d'être de cette mesure est liée à l'interaction intrinsèque au clustering sous contraintes incrémentales : quand un conflit est détecté dans l'ensemble des contraintes fournies, il faut expliquer à l'utilisateur ce qui pose problème et proposer une solution qui permette de continuer le processus. Dans cet esprit, on considère que - quitte à proposer une résolution du conflit - un MCS pertinent contient des contraintes qui semblent incongrues au vu des informations déjà données par l'utilisateur. On peut alors renforcer l'efficacité de la correction en laissant l'utilisateur confirmer la validité de tout ou partie des contraintes proposées à la relaxation, ou le rejeter pour trouver un autre MCS en écartant ces contraintes, le tout dans un fonctionnement transparent et itératif.
On peut faire le parallèle avec la mesure d'informativité présentée dans [7] : un contrainte informative selon un algorithme de clustering $\mathcal{A}$ contient surtout des contraintes qui ne sont pas immédiatement satisfaites par la simple exécution de $\mathcal{A}$ sur le jeu de données. De façon analogue, l'adéquation d'une contrainte à un modèle peut être vue comme une mesure de la capacité avec laquelle un modèle d'apprentissage supervisé $\mathcal{M}$ peut satisfaire une contrainte grâce à son seul entraînement.
Le modèle appris est un algorithme d'apprentissage supervisé qui utilise comme étiquettes l'affectation de la partition courante pour l'apprentissage. Son ensemble d'entraînement est un sous-ensemble du jeu de données sur lesquels l'utilisateur a déjà réagi. Ce modèle tente ensuite de généraliser aux points qui ne sont liés à aucune contrainte, représentant ainsi une approximation de la partition désirée par l'utilisateur, d'après les informations connues [4]. En fonction du modèle utilisé (basé sur des arbres, des vecteurs de support, probabiliste...), les paramètres définissant le modèle varient et la façon de calculer l'adéquation change également. Mais la signification pratique est claire : nous voulons éviter d'enlever les contraintes qui répondent parfaitement à cette vision de la partition désirée.

Définition 4.4 (Adéquation d'une contrainte à un modèle)**.** *Soit un jeu de données $\mathcal{X}$, un ensemble de contraintes $\mathcal{C}$, $\mathcal{M}$ un modèle d'apprentissage supervisé appris sur un sous-ensemble $\mathcal{X}_\mathcal{C}$ de $\mathcal{X}$, contenant les points impliqués dans des contraintes de $\mathcal{C}$, et c une contrainte d'arité n. Quand $\mathcal{M}$ est utilisé pour prédire les assignation du reste du jeu de données, il produit une partition prédit $\mathcal{P}_\mathcal{M}$. L'adéquation de c à $\mathcal{M}$ est une fonction $ADQ_\mathcal{M}(c) : \mathcal{M} \to [0,1]$ telle qu'une valeur de 1 correspond à un accord total avec $\mathcal{M}$ et à l'inverse, une valeur de 0 représente la plus forte dissonance entre $\mathcal{M}$ et la contrainte.*

Pour prendre l'exemple des contraintes de clustering définies en Section 3 : une contrainte *must-link*, qui indique que deux points doivent être groupés ensemble, aura une adéquation de 1 si le modèle utilisé les considère comme similaires, et une adéquation de 0 s'ils sont très différents (la notion de similarité dépendant du fonctionnement du modèle utilisé). De même, une contrainte triplet sera d'adéquation maximale si le modèle classe le point de référence et le point positif ensemble tout en classant le point négatif dans un cluster éloigné ou différent ; dans le cas des contraintes sur la plage de clusters, on considère que l'adéquation est nulle si le modèle disperse les points concernés dans plus de clusters que spécifié, sinon on évalue l'adéquation moyenne de chaque sous-ensemble placé dans le même cluster.
On définit les formules suivantes pour chaque type de contrainte afin de calculer leur adéquation à un modèle $\mathcal{M}$ doté d'une fonction de similarité entre deux points SIM(x,y) : $\mathcal{X} \to [0,1]$ dépendant du fonctionnement de $\mathcal{M}$:

$$ADQ_\mathcal{M}(ML(x,y)) = \text{SIM}(x,y) \tag{13}$$

$$ADQ_\mathcal{M}(CL(x,y)) = 1 - ADQ_\mathcal{M}(ML(x,y)) \tag{14}$$

3. De l'anglais *Maximal Satisfiable Subset*.
4. De l'anglais *Minimal Correction Subset*.

$$ADQ_{\mathcal{M}}(Triplet(a,p,n)) = 0.5 * (1 + \frac{ap - an}{\max(ap, an)}) \quad (15)$$

où $ap = ADQ_{\mathcal{M}}(ML(a,p))$ et $an = ADQ_{\mathcal{M}}(ML(a,n))$. Cette mesure donne 0.5 si le point de référence a est à égale distance de p et de n. Elle augmente à mesure que p se rapproche et/ou n s'éloigne, et diminue s'ils ont le comportement inverse. Dans le cas où les trois points sont confondus, l'adéquation vaut également 0.5, ce qui correspond au but de la fonction : il est étrange de donner une contrainte posant des distances différentes entre trois points s'ils sont confondus.

$$ADQ_{\mathcal{M}}(Span(S,\gamma)) = \begin{cases} 0 & \text{si } Sp(S) > \gamma \\ \sum_{x,y} ADQ_{\mathcal{M}}(ML(x,y)) & \text{sinon} \end{cases} \quad (16)$$

où γ est le nombre de clusters maximal de la contrainte (dans le cas générique) ou la cardinalité de l'ensemble de clusters fourni (dans le cas spécifique).

Les contraintes considérées dans le cadre de cette publication peuvent toutes être retranscrites en un ensemble de contraintes ML, ce qui permet de simplifier les formulations des mesures d'adéquation.

Pour expliciter le principe, prenons un cas concret où le modèle utilisé est un arbre de décision. La fonction SIM peut reposer sur différentes métriques liés aux arbres. Nous choisissons d'utiliser la mesure du dernier ancêtre commun entre deux points (Eq. 17). Ainsi, l'adéquation d'une contrainte ML par rapport à un arbre de décision sera de 1 si les deux points se trouvent dans la même feuille, et va diminuer à mesure que le dernier ancêtre commun est proche de la racine, jusqu'à atteindre 0 s'il s'agit de la racine elle-même. D'autres choix de métriques ou de modèles sont possibles à condition de pouvoir normaliser la mesure. En effet, l'adéquation pouvant porter sur des contraintes de portées très différentes, il est nécessaire de pouvoir exprimer l'adéquation de toute contrainte sur le même intervalle de valeurs afin d'éviter que des contraintes aient un score plus élevée simplement parce qu'elles portent sur un grand nombre de points. Concrètement, cela introduirait un biais favorisant la sélection de contraintes binaires dans la constitution de MCS.

$$LCA(x,y) = \frac{\text{profondeur}(x \cap y)}{\max(\text{profondeur}(x), \text{profondeur}(y))} \quad (17)$$

4.2 Modèle PPC pour choisir le MCS d'adéquation minimale

Afin de calculer un MCS contenant les contraintes d'adéquation minimale à un modèle donné, nous exprimons un problème d'optimisation sous contraintes (COP) en programmation par contraintes en reprenant les variables et contraintes décrites en Section 4. Plus particulièrement, la construction du MCS revient à donner une affectation aux variables binaires Z réifiant les contraintes utilisateur. Dans une solution du COP, on obtient donc un MCS $\{c_i | Z_i = 0\}$ et un MSS $\{c_i | Z_i = 1\}$. Une fonction objectif pondérant les contraintes avec leur adéquation au modèle appris sur les retours utilisateurs $\mathcal{M}$ oriente le solveur vers le MCS contenant les contraintes les moins adéquates :

$$\arg\min \sum_{i=0}^{|\mathcal{C}|} \max(ADQ_{\mathcal{M}}(c), \epsilon) \times \mathbb{I}(Z_i == 0) \quad (18)$$

La fonction indicatrice $\mathbb{I}(Z_i == 0)$ retourne 1 si la contrainte i doit être relâchée (et donc faire partie du MCS) et 0 si elle peut être conservée pour la modification.

Si le poids d'une contrainte est nul, alors l'assignation de la variable binaire correspondante dans le COP n'aura pas d'impact sur l'objectif. C'est pourquoi une valeur ϵ arbitraire non nulle mais proche de 0 est utilisée le cas échéant afin que toutes les contraintes de E soient prises en compte dans le calcul du MCS optimal. Cela est d'autant plus important qu'il paraît très pertinent d'inclure une contrainte avec une adéquation nulle au modèle dans une solution.

Dans le cadre du clustering sous contraintes incrémentales, l'insatisfiabilité d'un CSP à une itération t vient d'au moins une contrainte appartenant au lot L_t qui vient d'être obtenu (dans le cas contraire, la modification au temps $t-1$ n'aurait pas pu être calculée). Le problème peut donc être restreint à l'ensemble des contraintes contenant au moins un des points impliqués dans les contraintes de L_t :

$$E = \{c \in \mathcal{C}_t \mid \exists\, l \in L_t \; s.t. \; \text{portée}(c) \cap \text{portée}(l) \neq \emptyset\} \quad (19)$$

La formalisation du COP s'écrit donc sous la forme suivante :

$$\arg\min \quad \sum_{i=1}^{|E|} w_i \, (1 - Z_i) \quad (20)$$

$$\textbf{s.t.} \quad \sum_{j=1}^{|L_t|} (1 - Z_j) \geq 1 \quad (21)$$

$$Z_i \leftrightarrow c_i \quad (22)$$

La contrainte 21 indique qu'au moins une des contraintes du MCS doit provenir du dernier lot de contraintes. Elle sert à orienter la recherche sur un ensemble restreint des contraintes considérées.

5 Expérimentations

Les expérimentations visent à :

- estimer l'impact de la mesure d'adéquation sur les contraintes choisies pour correction.
- déterminer le nombre de contraintes qu'il est possible de traiter en temps raisonnable pour un contexte d'interaction avec l'humain.
- évaluer l'efficacité de la méthode dans un cas d'application réel, sur des séries temporelles d'images satellite.

5.1 Protocole expérimental

Implémentation. Le modèle PPC de calcul de MCS a été implémenté en Python avec CPMpy [5]. Cela permet d'utiliser en même temps le solveur CP-SAT d'`or-tools` [6] et les librairies d'apprentissage automatique de Python telles que *scikit-learn*. Nous avons choisi 10^{-4} comme valeur de ϵ [7]. La prédiction de la partition désirée par l'expert est réalisée avec un arbre de décision de profondeur maximale 3 utilisant l'indice de Gini pour les décisions de branchement.

Méthodologie. Le modèle PPC de calcul de MCS est intégré dans le cadre de clustering actif incrémental sous contraintes décrit en Section 3. L'arbre de décision utilisé pour calculer l'adéquation des contraintes est appris en début d'itération, avant d'obtenir un nouveau lot de contraintes. Le modèle PPC de calcul du MCS optimal est utilisé en *post-hoc*, si la modification de partition échoue. La première itération ne produit pas de conflit : les contraintes sont générées en se basant sur la vérité terrain des jeux de données. Le deuxième lot de contraintes est bruité de façon à introduire un conflit avec le premier lot. Nous avons répété les expériences 10 fois pour prendre en compte la variance des résultats.

Jeux de données. Les expériences ont été réalisées sur trois jeux de données du dépôt de l'UC Irvine [8] : MNIST (60000 points répartis en 10 classes), Letters (20000 points répartis en 26 classes) et Adult (48842 points répartis en 2 classes).

5.2 Impact de la mesure d'adéquation

Pour évaluer si les MCS calculés en utilisant la mesure d'adéquation sont pertinents, nous comparons les MCS construits à partir d'ensembles de contraintes insatisfiables **sans pondération** (c.à.d le MCS optimal est celui qui contient le moins de contraintes possible), et avec **pondération par l'adéquation** à un arbre de décision appris sur une partition des points soumis aux contraintes.

Nous avons testé différents cas représentant les différentes combinaisons de contraintes possibles parmi celles définies en Section 4 :

- un ensemble de contraintes ML/CL contenant un ou plusieurs cycles contenant une unique contrainte CL [20] ;
- un ou plusieurs triplets en conflit avec des contraintes CL posées sur les mêmes points concernés par le triplet ;
- une contrainte sur la plage de clusters sur un groupe de points impliqués dans un cycle de contraintes CL tel qu'il faut plus de clusters pour satisfaire le cycle de CL que ne le permet la contrainte sur la plage de clusters.

On constate d'après la table 1 que les contraintes constituant les MCS calculés par notre modèle PPC ont une adéquation plus faible à l'arbre de décision utilisé que celles d'un MCS obtenu sans pondération des contraintes. Quand les MCS calculés avec et sans pondération ont la même cardinalité, le choix optimal fait avec la pondération ne varie pas. A l'inverse, le seul critère de cardinalité introduit de la variance dans les résultats à partir du moment où plusieurs MCS de même taille existent.

Cas	ML/CL	Triplet	Plage de clusters
MNIST			
Non pondéré	0.67	0.78	0.89
Pondéré	0.45	0.12	0.34
Letters			
Non pondéré	0.6	0.85	0.59
Pondéré	0.34	0.4	0.23
Adult			
Non pondéré	0.89	0.77	0.87
Pondéré	0.24	0.18	0.23

TABLE 1 – Table des valeurs d'adéquation moyennes des contraintes constituant les MCS calculées avec ou sans pondération.

5.3 Passage à l'échelle du modèle

Étant donné que le modèle est destiné à une application dans un cadre interactif, il est primordial que la solution optimal doit trouvée en temps réel. Nous avons mesuré le temps de calcul du MCS pour différentes tailles d'ensembles de contraintes ML/CL (de 10 à 10000 contraintes) sur le jeu de données MNIST. La figure 1 montre que le temps de calcul n'excède pas 5 secondes en moyenne pour la plus grande valeur testée. Pour les tailles d'ensembles de contraintes attendues dans le contexte du clustering incrémental sous contraintes, le temps d'exécution est de l'ordre de la seconde.

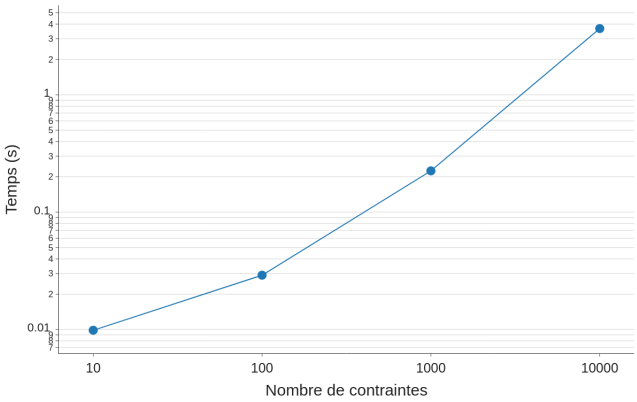


FIGURE 1 – Évolution du temps de calcul de MCS pour différentes tailles d'ensembles de contraintes (échelle logarithmique).

Ces résultats montrent non seulement que ce modèle est

5. https://github.com/CPMpy/cpmpy
6. https://developers.google.com/optimization
7. Depuis la version 9.24 de CPMpy, des coefficients réels peuvent être utilisés dans la fonction objectif, ce qui évite les transformations en valeurs entières habituellement nécessaires. Nous avons cependant constaté que le solveur cesse de donner des solutions correctes pour des valeurs en-dessous de 10^{-4}.
8. https://archive.ics.uci.edu/ml/index.php

suffisamment rapide pour être intégré dans un cadre interactif, mais aussi qu'il peut être employé *avant* de tenter la modification de partition, pour vérifier la satisfiabilité de l'ensemble de contraintes C_t et produire directement un moyen de résoudre le conflit le cas échéant.

5.4 Application sur un cas réel : coupes de bois

Présentation des données. Le jeu de données utilisé est une série temporelle de 11 images satellite de dimensions 724×337 d'une zone des Vosges, prises sur une période de 3 ans (détail en table 2) [12]. Au lieu des canaux habituels, chaque pixel est associé à une série de valeurs NDVI (*Normalized Difference Vegetation Index*) indiquant le niveau de végétation à chaque date.

Période	Date	Coupes
t_1	08/05/2016	-
t_2	23/08/2016	-
t_3	26/08/2016	-
t_4	10/05/2017	8
t_5	19/06/2017	-
t_6	23/08/2017	-
t_7	25/09/2017	-
t_8	08/05/2018	2
t_9	02/07/2018	-
t_{10}	24/07/2018	-
t_{11}	16/08/2018	-

TABLE 2 – Dates des prises de vue de la série temporelle et nombre d'apparitions de zones de coupes claires à chaque période

On dispose d'étiquettes qui décomposent la série temporelle en 3 classes : végétation, artificiel, coupe claire. Cette dernière classe a plusieurs particularités : elle a été construite avec précision par des experts, alors que les deux autres ont été délimitées de façon plus approximative. Elle est aussi très petite en proportion des données totales (moins de 0.3% des données, ce qui correspond à 639 pixels sur un total de 243.988 pixels), ce qui la rend difficile à détecter par des méthodes non supervisées. Elle est composée de 10 zones identifiées par les experts comme des endroits où des arbres ont été coupés. La période d'apparition des zones de coupes claires est indiquée dans la table 2. Enfin, l'évolution temporelle des zones de coupes claires (une baisse forte et ponctuelle du NDVI suivie d'un retour progressif à la normale) est similaire à celle des zones de prairies fauchées ou de récoltes de cultures, ce qui complique encore le problème. Dans ces conditions, l'intervention de l'expert pour dissiper la confusion est primordiale. La figure 2a présente la répartition en pixels des classes annotées et met en exergue la complexité de la tâche.
On dispose de 179 contraintes ML/CL issues de l'expertise humaine (Figure 2b). Pour tester notre méthode, nous introduisons du bruit dans ces contraintes pour simuler des erreurs dans les annotations humaines. Cela se traduit par l'inversion du statut des contraintes bruitées (un ML devient un CL ou inversement). Plus la proportion de contraintes bruitées est grande, plus l'algorithme d'apprentissage supervisé duquel découle l'adéquation est impacté par le bruit. Nous avons mesuré que notre modèle PPC de calcul de MCS identifie toutes les contraintes bruitées tant que moins 10% de l'ensemble de contraintes est bruité. Au-delà de 25% de contraintes bruitées, les MCS ont autant de chance de comporter de vraies annotations que les contraintes bruitées.

6 Conclusion

Nous avons développé un modèle en programmation par contraintes pour calculer un ensemble de correction minimal sur des ensembles de contraintes en clustering. Ce modèle repose sur une mesure d'adéquation des contraintes à un algorithme d'apprentissage supervisé appris sur les retours utilisateurs passés. Le modèle trouve un MCS optimal efficacement et choisit des contraintes qui semblent contradictoires avec le reste du retour utilisateur. Une piste d'amélioration possible est l'étude du fonctionnement de la mesure d'adéquation avec d'autres modèles (SVM, classificateur bayésien). Des expériences supplémentaires simulant des comportements d'utilisateurs humains dans la génération d'erreurs pourraient confirmer la pertinence de notre méthode.

Remerciements

Ce travail est soutenu par le projet ANR HERELLES (Hétérogénéité des données - Hétérogénéité des méthodes : Un cadre collaboratif unifié pour l'analyse interactive de données temporelles) [9].

Références

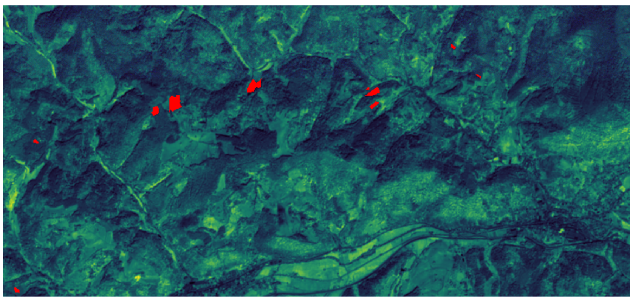
[1] Ahmad Ali Abin and Hamid Beigy. Active selection of clustering constraints : A sequential approach. *Pattern Recognition*, 47(3) :1443–1458, March 2014.

[2] Arindam Banerjee and Joydeep Ghosh. Scalable clustering algorithms with balancing constraints. *Data Min. Knowl. Discov.*, 13(3) :365–395, 2006.

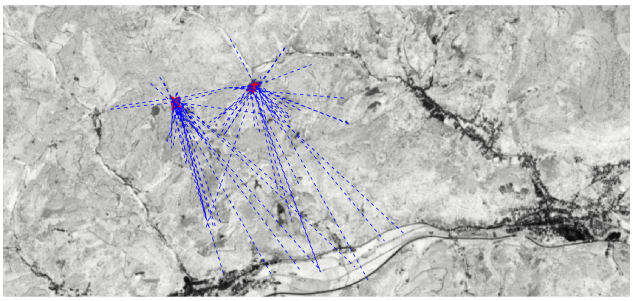
[3] Aymeric Beauchamp, Thi-Bich-Hanh Dao, Samir Loudni, and Christel Vrain. Incremental Constrained Clustering by Minimal Weighted Modification. In Roland H. C. Yap, editor, *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10 :1–10 :22, Toronto, Canada, September 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[4] Aymeric Beauchamp, Thi-Bich-Hanh Dao, Samir Loudni, and Christel Vrain. Rule-Based Constraint Elicitation For Active Constraint-Incremental Clustering. In *2024 IEEE 36th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 798–805, October 2024.

9. https://anr.fr/Projet-ANR-20-CE23-0022



(a) Zones de coupes de bois sur les images (en rouge).



(b) Mise en évidence des contraintes ML/CL des annotateurs.

FIGURE 2 – Présentation du jeu de données (contraintes ML en rouge, CL en bleu pointillé).

[5] Christian Bessière, Emmanuel Hébrard, Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. Filtering Algorithms for the NValue Constraint. *Constraints*, 11(4) :271–293, December 2006.

[6] David Cohn, Rich Caruana, and Andrew Mccallum. Semi-Supervised Clustering with User Feedback. 4, October 2001.

[7] Ian Davidson, Kiri L. Wagstaff, and Sugato Basu. Measuring Constraint-Set Utility for Partitional Clustering Algorithms. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Knowledge Discovery in Databases : PKDD 2006*, Lecture Notes in Computer Science, pages 115–126, Berlin, Heidelberg, Germany, 2006. Springer.

[8] Pierre Gançarski, Thi-Bich-Hanh Dao, Bruno Crémilleux, Germain Forestier, and Thomas Lampert. Constrained Clustering : Current and New Trends. In Pierre Marquis, Odile Papini, and Henri Prade, editors, *A Guided Tour of Artificial Intelligence Research*, pages 447–484. Springer International Publishing, Cham, 2020.

[9] Ulrich Junker. *QUICKXPLAIN : Preferred Explanations and Relaxations for over-Constrained Problems*. January 2004.

[10] C.-T. Kuo, S. S. Ravi, T.-B.-H. Dao, C. Vrain, and I. Davidson. A framework for minimal clustering modification via constraint programming. AAAI, pages 1389–1395, 2017.

[11] Baptiste Lafabregue, Pierre Gançarski, Jonathan Weber, and Germain Forestier. Incremental constrained clustering with application to remote sensing images time series. In *2022 IEEE International Conference on Data Mining Workshops (ICDMW)*, 2022.

[12] Thomas Lampert, Baptiste Lafabregue, Thi-Bich-Hanh Dao, Nicolas Serrette, Christel Vrain, and Pierre Gancarski. Constrained Distance-Based Clustering for Satellite Image Time-Series. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(11) :4606–4621, November 2019.

[13] Mark H. Liffiton and Ammar Malik. Enumerating Infeasibility : Finding Multiple MUSes Quickly. In Carla Gomes and Meinolf Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 7874, pages 160–175. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[14] Mark H. Liffiton and Karem A. Sakallah. Algorithms for Computing Minimal Unsatisfiable Subsets of Constraints. *Journal of Automated Reasoning*, 40(1) :1–33, January 2008.

[15] Pavan Kumar Mallapragada, Rong Jin, and Anil K. Jain. Active query selection for semi-supervised clustering. In *2008 19th International Conference on Pattern Recognition*, pages 1–4, Tampa, FL, USA, December 2008. IEEE.

[16] Logan Adam Mitchell. *INCREMENT - Interactive Cluster Refinement*. PhD thesis, 2016.

[17] Nguyen-Viet-Dung Nghiem, Christel Vrain, and Thi-Bich-Hanh Dao. Knowledge integration in deep clustering. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2022, Proceedings, Part I*, volume 13713 of *Lecture Notes in Computer Science*, pages 174–190. Springer, 2022.

[18] Nguyen-Viet-Dung Nghiem, Christel Vrain, Thi-Bich-Hanh Dao, and Ian Davidson. Constrained Clustering via Post-processing. In *Discovery Science*, pages 53–67, 2020.

[19] Swapna Raj Prabakara Raj and Balaraman Ravindran. Incremental Constrained Clustering : A Decision Theoretic Approach. In *Trends and Applications in Knowledge Discovery and Data Mining*, volume 7867, pages 475–486. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[20] Jonas Soenen, Sebastijan Dumančić, Toon Van Craenendonck, and Hendrik Blockeel. Tackling Noise in Active Semi-supervised Clustering. In Frank Hutter, Kristian Kersting, Jefrey Lijffijt, and Isabel Valera, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 121–136, Cham, 2021. Springer International Publishing.

[21] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained K-means Clustering with

Background Knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 577–584, San Francisco, CA, USA, June 2001. Morgan Kaufmann Publishers Inc.

[22] Aras Yurtman, Wannes Meert, and Hendrik Blockeel. *COBRAS+ : Reusing Previously Obtained Constraints in Active Semi-Supervised Clustering*. November 2021.