

Intégration incrémentale de contraintes pour le clustering avec la programmation par contraintes

Aymeric Beauchamp, Thi-Bich-Hanh Dao, Samir Loudni, Christel Vrain

▶ **To cite this version:**

Aymeric Beauchamp, Thi-Bich-Hanh Dao, Samir Loudni, Christel Vrain. Intégration incrémentale de contraintes pour le clustering avec la programmation par contraintes. Journées Francophones de Programmation par Contraintes (Evènement affilié à PFIA 2022), Jun 2022, Saint-Étienne, France. hal-03687871

HAL Id: hal-03687871
https://hal.science/hal-03687871

Submitted on 28 Sep 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Intégration incrémentale de contraintes pour le clustering avec la programmation par contraintes

Aymeric Beauchamp[1], Thi-Bich-Hanh Dao[1], Samir Loudni[2], Christel Vrain[1]

[1]Université d'Orléans, LIFO
[2]IMT Atlantique Bretagne-Pays de la Loire

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

Résumé

L'implication active d'un expert dans un processus de clustering sous contraintes se traduit par une démarche incrémentale où les contraintes expertes sont ajoutées à la volée. Toutefois, les clusterings intermédiaires produits devront être relativement similaires pour ne pas dérouter l'expert. Nous proposons un modèle en PPC permettant d'obtenir une partition similaire à la partition existante tout en tenant compte des contraintes. Notre modèle peut traiter plusieurs types de contraintes, relâcher les contraintes et créer un nouveau cluster. Des expériences menées sur des jeux de données de référence ainsi que dans un cas d'usage réel en télémétrie montrent l'intérêt de notre modèle.

Mots-clés

Clustering sous contraintes, programmation par contraintes, optimisation combinatoire

Abstract

Actively involving an expert in a constrained clustering loop translates into an incremental process where expert constraints are added on the fly. However, intermediate clusterings must be somewhat similar to one another in order not to confuse the expert. We propose a CP model to build a partition that is similar to an existing partition while taking constraints into account. Our model can support several types of constraints and features constraint relaxation and cluster creation. Experiments performed on popular datasets as well as a use case in remote sensing denote the relevance of our model.

Keywords

Constrained clustering, constraint programming, combinatorial optimization

1 Introduction

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 même cluster soient similaires mais différents de ceux appartenant aux autres clusters. Le clustering sous contraintes [6] 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 imposant que des objets appartiennent (ou non) à un même cluster [14], ou que les clusters aient une taille minimale et/ou maximale [1]. 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 incorporer dès le début du processus toutes les connaissances nécessaires à l'élaboration d'une partition intéressante. L'idée est donc de lui proposer une partition qui lui permettra de suggérer des connaissances intéressantes à intégrer, conduisant ainsi à une nouvelle partition qui pourra être encore itérativement améliorée. Ceci constitue un processus interactif qui est bien connu en fouille de données.

Comme l'illustre la figure 1, l'implication active de l'expert dans le processus de clustering se traduit par l'intégration incrémentale de contraintes représentant des connaissances de l'expert à une partition existante. Ces contraintes peuvent prendre des formes diverses et doivent être satisfaites dans la nouvelle partition modifiée. Par exemple, l'expert peut ainsi indiquer que deux objets appartenant à un même cluster devraient être séparés car il les juge dissimilaires. Les nouvelles contraintes données par l'expert doivent être prises en compte pour construire une nouvelle partition. Toutefois, afin de ne pas trop dérouter l'expert, la partition résultante doit être relativement similaire à la partition initiale.

Une première approche naïve pour tenir compte des contraintes données par l'expert consiste à relancer un algorithme de clustering sous contraintes sur les données avec les nouvelles contraintes. Cependant, cette approche présente au moins deux inconvénients majeurs : (1) elle relance le clustering sur les données initiales à partir de zéro sans tenir compte des résultats intermédiaires ; (2) elle peut produire des résultats très différents de la partition qui a été présentée à l'expert. Ainsi, l'enjeu dans une approche incrémentale est de réduire le fossé entre les résultats produits par l'algorithme du clustering et les intuitions de l'expert durant la boucle d'interaction.

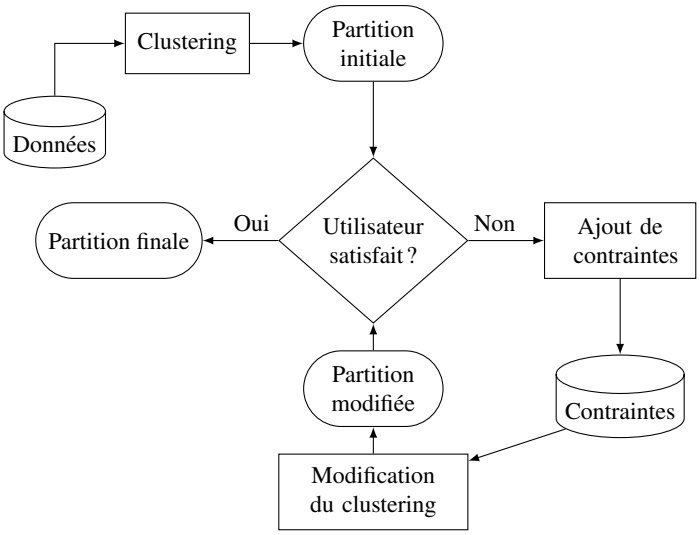


FIGURE 1 – Représentation du processus de clustering par contraintes incrémentales.

Dans cet article, nous proposons une nouvelle approche incrémentale basée sur la programmation par contraintes (PPC) permettant d'intégrer des contraintes utilisateur à une partition existante, et de produire une nouvelle partition qui soit proche de la partition existante. L'originalité de notre méthode est d'intégrer dans le critère d'optimisation, pour les points changeant de cluster, la distance au centre de leur nouveau cluster, pour limiter les modifications par rapport à une partition existante. À la différence des méthodes existantes à base de la programmation linéaire en nombres entiers (PLNE) [11], notre méthode permet de trouver une solution satisfaisant le plus grand nombre de contraintes dans le cas de conflits entre les contraintes, alors que les autres approches échouent. Des expériences menées sur des jeux de données de référence ainsi que sur un cas d'usage réel en télémétrie montrent l'intérêt de notre modèle. Nous discutons également des inconvénients d'une modification minimale et présentons une solution de compromis entre deux critères : l'amélioration du clustering par les contraintes et la nécessité de ne pas produire de résultats perturbants pour l'expert.

Dans la section 2, nous discutons des travaux antérieurs liés au clustering sous contraintes, ou à la modification minimale de clustering. Dans la section 3, nous décrivons notre modèle en programmation par contraintes pour la modification de clustering et détaillons des méthodes pour augmenter l'efficacité des contraintes. Dans la section 4, nous présentons les résultats du modèle sur des jeux de données de référence ainsi que sur des séries temporelles en télémétrie.

2 Travaux antérieurs

Les premiers travaux développés sur l'intégration de contraintes dans des tâches de clustering sont des extensions d'algorithmes de clustering classiques pour intégrer des contraintes *must-link/cannot-link* [4, 14, 16]. Ces algorithmes peuvent chercher à satisfaire toutes les contraintes comme COP-KMEANS [14] ou cherchent une solution qui est un compromis entre la satisfaction de contraintes et la qualité du clustering [2]. Par la suite, des approches fondées sur des formalismes déclaratifs ont été proposées, permettant d'intégrer des contraintes plus générales que des contraintes *must-link/cannot-link*, avec un solveur SAT [5], avec la PLNE [12] ou encore avec la PPC [3]. Toutes ces méthodes cependant cherchent à construire un clustering à partir des données en présence des contraintes. Elles ne sont par conséquent pas appropriées dans une démarche incrémentale, car l'apparition de nouvelles contraintes nécessite de relancer l'algorithme de nouveau à partir des données, sans garantie que la nouvelle partition soit similaire à l'ancienne.

Le problème de modification minimale d'une partition a été posé dans [9]. Le modèle proposé dans ce travail est basé sur la PLNE et intègre des contraintes sur les diamètres des clusters relativement à un descripteur. L'objectif recherché est d'enlever des propriétés indésirables d'une partition. Cependant dans ce travail, la différence avec la partition initiale est mesurée en nombre de points qui changent de cluster, sans tenir compte de l'homogénéité des clusters. Une autre approche utilisant la PLNE [11] exploite la partition initiale en mesurant le score d'appartenance d'un point à chaque cluster. Ce travail cherche ensuite à construire un clustering optimisant un critère fondé sur ce score ; toutes les données sont prises en compte, même celles qui ne sont pas liées aux contraintes. Notons enfin que ces approches ne permettent pas de gérer le cas de conflits entre contraintes.

Notre approche est basée sur la PPC, et vise à conserver l'homogénéité des clusters d'une partition existante tout en tenant compte de nouvelles contraintes. Elle permet d'intégrer différents types de contraintes et de relâcher des contraintes en cas de conflit.

3 Modèle de clustering par contraintes incrémentales

Nous présentons dans cette section un modèle en programmation par contraintes pour le problème de modification minimale de partition : nous utilisons un critère d'optimisation basé sur la distance qui tend à limiter la portée des changements apportés à une partition existante – produite par un algorithme de clustering ou issue d'une précédente itération du modèle – pour rester relativement similaire à cette dernière tout en améliorant sa qualité par l'intégration du retour expert sous forme de contraintes.

3.1 Problème d'optimisation sous contraintes pour la modification de clustering

Nous considérons un ensemble de N instances $\mathcal{X}$ et une partition existante $\mathcal{P}$ de $\mathcal{X}$ en K clusters, où $\mathcal{P}[i]$ indique le numéro du cluster contenant l'instance i. Nous proposons de représenter la structure de la partition $\mathcal{P}$ par une matrice de distances $\mathcal{D}$ de taille $N \times K$, où $\mathcal{D}[i, c]$ représente la distance d'une instance i au centre du cluster c.

Variables et fonction objectif. Pour représenter la partition modifiée, nous utilisons N variables $X_1, \ldots, X_N$ ayant pour domaines $\{1, \ldots, K\}$, où $X_i = c$ indique que l'instance i est affectée au cluster c. Comme la matrice $\mathcal{D}$ représente la structure de $\mathcal{P}$, afin d'obtenir une partition proche à $\mathcal{P}$, nous proposons une fonction objectif qui minimise, pour les instances réaffectées dans la partition modifiée, la distance au centroïde de leur nouveau cluster :

$$\arg\min \sum_{i=0}^{N} \mathbb{I}(\mathcal{P}[i] \neq X_i)\ \mathcal{D}[i, X_i]$$

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.

Contraintes expert. Différents types de contraintes d'expert peuvent être intégrés dans le modèle. Une contrainte *must-link* (resp. *cannot-link*) sur deux instances i, j indique que ces instances doivent être (resp. ne doivent pas être) dans le même cluster. Ces contraintes sont représentées par $X_i = X_j$ pour la contrainte *must-link* et $X_i \neq X_j$ pour la contrainte *cannot-link*.
Une contrainte triplet (a, p, n) signifie que l'instance a *est plus proche de* p *que de* n. Elle est traduite par :

$$X_a = X_n \implies X_a = X_p$$
$$X_a \neq X_p \implies X_a \neq X_n$$

Pour inclure des retours expert plus thématiques, des contraintes logiques peuvent être intégrées. Elles peuvent être de la forme $P \implies Q$, où P et Q sont des conjonctions de contraintes élémentaires de type $X_i = X_j$ ou $X_i \neq X_j$.

Création de nouveaux clusters. Notre modèle permet de spécifier un nombre de clusters plus grand que K, en définissant le domaine des X_i à $\{1, \ldots, K'\}$, avec $K' > K$. Il permet ainsi d'affecter une instance à un nouveau cluster en cas de non satisfaction de toutes les contraintes. Par exemple si $K = 2$ et que l'expert impose les trois contraintes suivantes : $X_i \neq X_j, X_j \neq X_l$ et $X_l \neq X_i$, alors il sera nécessaire de former un nouveau cluster pour trouver une solution. Par rapport à la partition existante $\mathcal{P}$, nous fixons la distance $\mathcal{D}[i, k']$ à une valeur plus grande que la distance au centre le plus éloigné de i pour $K < k' \leq K'$, et ce afin d'éviter que le modèle ne crée des clusters dont le seul intérêt est d'optimiser la fonction objectif.

Relâchement de contraintes expert. L'expert pourrait faire des erreurs, ce qui se traduirait par le fait que les contraintes ajoutées sont contradictoires. Il n'existe pas dans ce cas de solution satisfaisant toutes les contraintes. Nous autorisons alors à satisfaire un certain nombre de contraintes. Chaque contrainte expert c est associée à une variable booléenne S_c de sorte que $S_c = 1$ si et seulement si c est satisfaite. La somme des variables S_c donne le nombre de contraintes satisfaites par une solution donnée. On peut ainsi imposer une contrainte établissant un seuil de satisfaction des contraintes expert :

$$\sum_c \mathcal{S}_c \gtreqless \delta$$

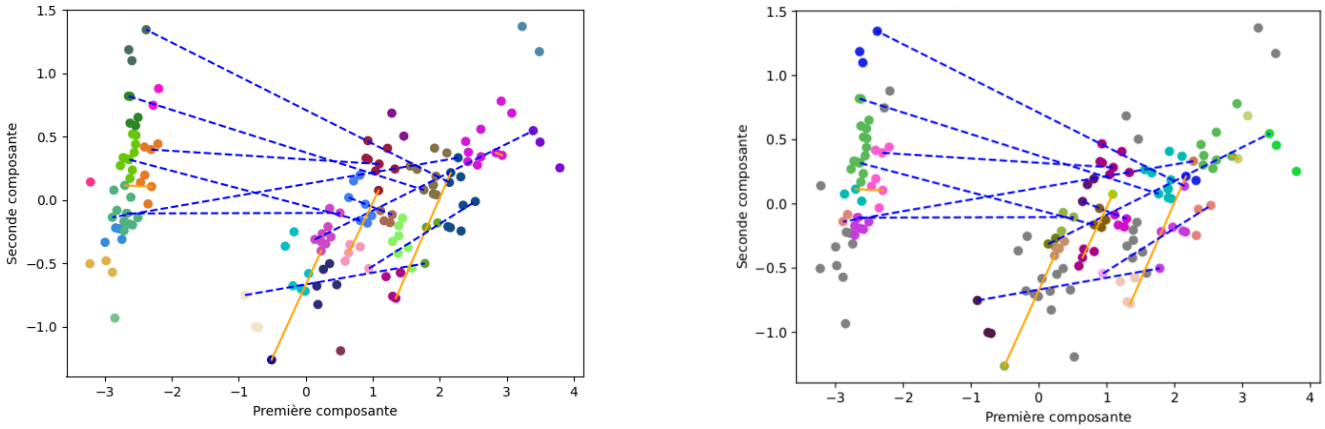
L'introduction de la relaxation de contraintes permet à notre modèle de résoudre des problèmes contenant des contraintes contradictoires et d'ignorer des contraintes nécessitant la modification d'instances éloignées de leur nouveau cluster. Dans un contexte incrémental, on peut envisager que l'expert fasse varier le taux de satisfaction des contraintes. Cette fonctionnalité engendre toutefois une combinatoire plus importante qui se répercute sur le temps de calcul (cf. section 4.1.2 pour plus de détails).

Restriction du modèle. Étant données de nouvelles contraintes, l'objectif est de modifier la partition existante en préservant le plus possible la structure de $\mathcal{P}$ tout en satisfaisant ces nouvelles contraintes. Ainsi, si une instance n'apparaît dans aucune contrainte, son affectation ne sera pas modifiée afin de conserver la structure de $\mathcal{P}$. Par conséquent il suffit de n'inclure dans le modèle de PPC que les instances concernées par au moins une contrainte.

3.2 Propagation des modifications

Dans une application réelle, nous supposons que l'expert ne va réagir que sur un petit nombre d'instances par itération. Il en découle que la modification du clustering est tellement minimale qu'elle en devient imperceptible. Il est donc important d'exploiter les retours pour propager les informations/modifications et ainsi éviter de poser à l'expert un nombre prohibitif de requêtes. Il faut dès lors trouver un moyen de permettre la propagation des modifications à des instances non contraintes, tout en la contrôlant pour ne pas dérouter l'expert. Notre intuition est que les contraintes sur des instances ne concernent pas uniquement les instances elles-mêmes, mais aussi leur voisinage. Dans cette optique, nous présentons deux façons de transmettre localement les modifications réalisées par notre modèle.

Propagation par super-instances. Notre première approche consiste à construire des super-instances [13] : des instances virtuelles regroupant plusieurs points de données réels. Ainsi, la modification des super-instances se traduit par des changements sur toutes les instances qui la composent, contraints ou non. La création des super-instances est réalisée en décomposant chaque cluster de la partition initiale en petits groupes, chaque groupe représentant une super-instance. On calcule ensuite une matrice de distance $\mathcal{D}'$ de taille $N_s \times K$ avec N_s le nombre de super-instances, où $\mathcal{D}'[i, c]$ représente la distance moyenne des instances de la super-instance i au centre du cluster c. Le nombre de super-instances par cluster est déterminé arbitrairement et a une utilité indirecte pour définir la granularité de la propagation des contraintes : quand N_s tend vers le nombre d'instances du cluster, le nombre moyen d'instances dans une super-instance tend vers 1.
La décomposition des clusters en super-instances est effectuée par un algorithme de clustering exécuté sur chacun des clusters de la partition existante. Conformément à notre idée que les contraintes de l'expert sur des instances symbolisent en fait un ensemble de modifications à effectuer sur ces instances et leur voisinage, nous recherchons des super-instances compactes, ce qui oriente notre choix vers des algorithmes comme le clustering hiérarchique *complete-link*,



(a) Résultat d'une génération de super-instances via un algorithme de clustering hiérarchique ascendant *complete-link* lancé sur chaque cluster. Les instances d'une même couleur sont représentées par la même super-instance.

(b) Super-instances obtenues après division des instances contraintes pour éviter l'émergence de contraintes indésirables. Les instances en gris font partie de super-instances non contraintes et ne sont donc pas pris en compte par le modèle.

FIGURE 2 – Exemple de prétraitement pour la génération de super-instances sur le jeu de données Iris.

FPF [7], ou KMEANS. Le processus est décrit dans la figure 2.

Les contraintes de l'utilisateur liant des instances doivent être transférées aux super-instances afin que notre modèle puisse les traiter. La transmission des contraintes peut avoir comme effet de bord l'émergence de contraintes absentes du jeu de contraintes fourni, et qui peuvent rendre le problème insoluble. La figure 3 présente une illustration de ce problème. Pour éviter cet écueil, nous avons décidé d'ajouter un pré-traitement supplémentaire consistant à diviser les super-instances contenant plusieurs instances contraints. Nous nous servons des instances contraintes comme centres des nouvelles super-instances et nous assignons les instances restantes au centre le plus proche. Ainsi, le problème de la figure 3 est résolu en divisant les super-instances en deux.

Propagation aux plus proches voisins. Notre seconde approche consiste à propager les modifications du modèle en post-traitement aux plus proches voisins des instances modifiées. Cette méthode permet de faire varier facilement la propagation des modifications à partir d'une solution optimale du modèle et est facilement applicable quel que soit le jeu de données. C'est également une méthode qui fait des modifications locales et qui n'est pas sujette aux problèmes de temps et/ou de mémoire qui peuvent arriver avec la méthode des super-instances. Cependant la portée de la propagation par cette voie est moins prévisible, alors que la génération de super-instances définit en amont les instances ciblées par la propagation de modifications ; l'expert peut ainsi être informé à l'avance de la portée des changements.

4 Expérimentations

Les expérimentations visent à :

— estimer l'impact des contraintes sur la qualité du clustering, tout en gardant la structure générale des clusters existants.

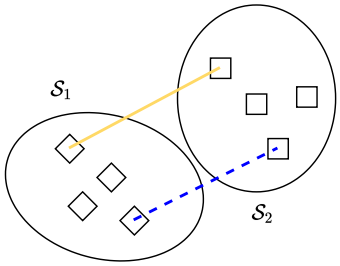
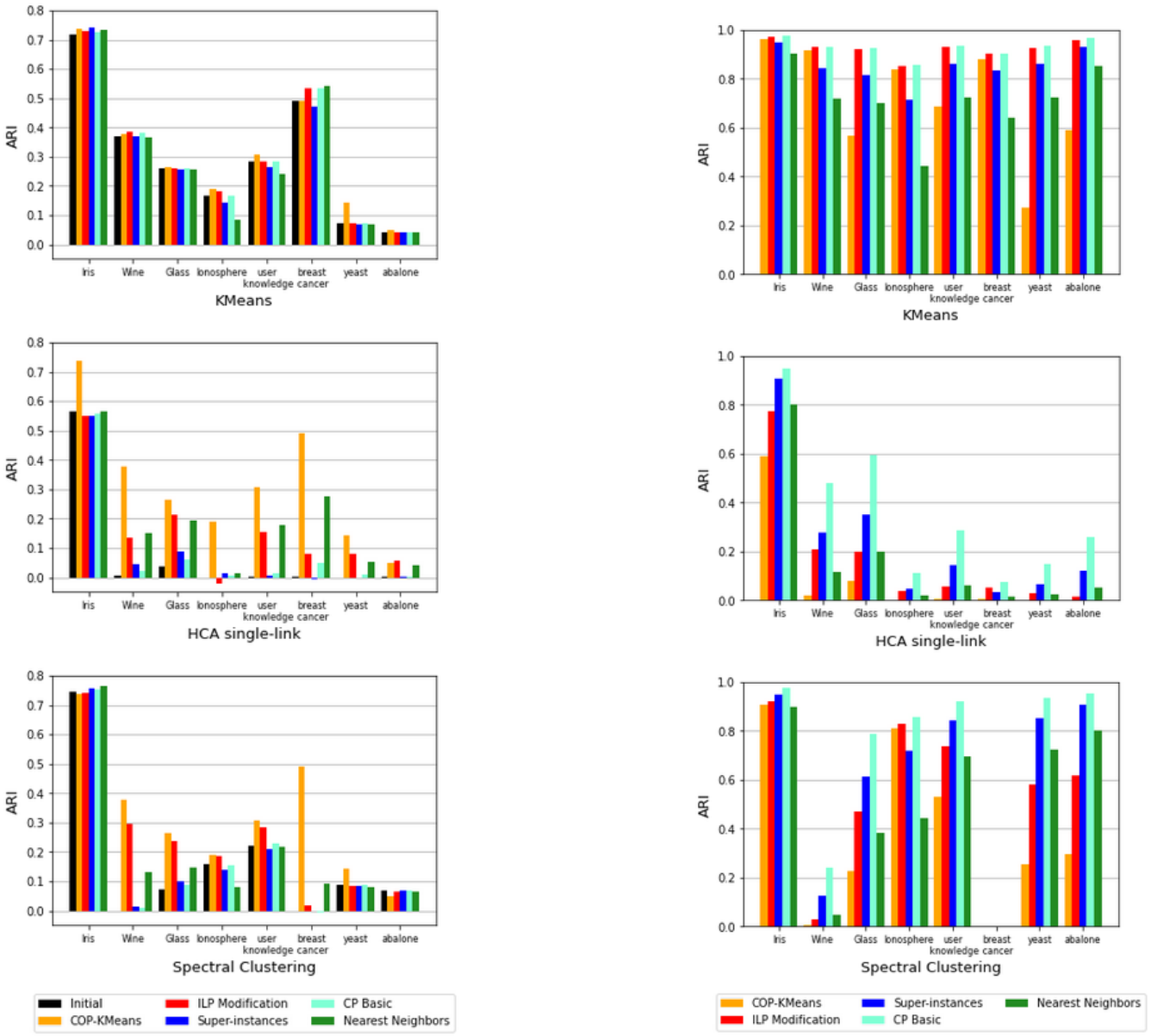


FIGURE 3 – Exemple de contraintes émergentes : une paire d'instances appartenant à deux super-instances $\mathcal{S}_1$ et $\mathcal{S}_2$ sont soumises à une contrainte *must-link* (en orange), tandis qu'une autre paire est soumise à une contrainte *cannot-link* (en bleu pointillés). Il se trouve alors que $\mathcal{S}_1$ et $\mathcal{S}_2$ sont en même temps soumis à une contrainte *must-link* et une contrainte *cannot-link*, ce qui rend le problème insoluble.

— comparer les performances de notre approche par rapport à d'autres méthodes similaires.
— étudier la capacité de relâcher des contraintes.

Implémentation. Le modèle a été implémenté dans un premier temps en C++ avec la librairie *open source* Gecode 6.2.0 [1], puis en Python avec CPMpy [2]. Cette nouvelle librairie fait l'interface avec différents solveurs connus, tels que CP-SAT d'`or-tools` [3], que nous utilisons. Nous avons constaté un *speedup* allant jusqu'à deux ordres de grandeur par rapport à Gecode. En termes de stratégie de recherche, nous choisissons en premier les variables X de plus haut degré puisque leur affectation peut permettre de satisfaire plusieurs contraintes rapidement et avec peu de modifications. Nous comparons notre approche avec une méthode de

1. https://www.gecode.org
2. https://github.com/CPMpy/cpmpy
3. https://developers.google.com/optimization



(a) Valeurs d'ARI entre la vérité terrain et les méthodes

(b) ARI entre la partition existante et les méthodes.

FIGURE 4 – Évolution de l'ARI des partitions calculées avec les différentes méthodes comparées à (a) la vérité des données et (b) les partitions existantes. **Initial** est la partition générée par l'algorithme indiqué en abscisse ; **ILP Modification** est la méthode de post-processing en PLNE présentée dans [11] ; **CP Basic** est le résultat de notre modèle sans propagation ; les meilleurs résultats des méthodes de propagation sont également indiqués par **Super-instances** et **Nearest Neighbors.**

post-processing en PLNE avec Gurobi [4] [11] ainsi qu'une implémentation Python de COP-KMEANS [5] [15].

4.1 Jeux de données de référence

Nous utilisons des jeux de données issus du dépôt pour l'apprentissage automatique de l'UC Irvine [6] décrits en table 1.

4.1.1 Comparaison de performances

Pour chaque jeu de données, nous générons 15 jeux de contraintes *must-link*/*cannot-link* par une procédure aléatoire : on tire au hasard deux points du jeu de données ; si les points sont dans le même cluster du point de vue de la vérité terrain, on crée une contrainte *must-link*, sinon on crée une contrainte *cannot-link*. Nous simulons ainsi un retour expert sur les données. Chaque jeu de contraintes ainsi créé contient un nombre de contraintes équivalent à 10% de la taille du jeu de données.

Nous avons généré une partition de départ avec trois méthodes : KMEANS, clustering hiérarchique ascendant *single-link* (HCA) et clustering spectral. Lors de l'intégration de contraintes, COP-KMEANS n'utilise pas la partition de départ et est relancé pour chaque jeu de contraintes.

4. https://github.com/dung321046/ConstrainedClusteringViaPostProcessing
5. https://github.com/Behrouz-Babaki/COP-Kmeans
6. https://archive-beta.ics.uci.edu/ml/datasets

Nom	Objets	Attributs	Classes
iris	150	4	3
wine	178	13	3
glass	214	9	7
ionosphere	351	34	2
user knowledge	403	5	4
breast cancer	569	30	2
yeast	1484	8	10
abalone	4177	7	28

TABLE 1 – Jeux de données utilisés, triés par ordre croissant du nombre d'objets. Le nombre d'attributs que possède chaque objet, ainsi que le nombre de classes (et donc de clusters à trouver) sont également indiqués.

Nous effectuons donc une seule itération du processus de clustering par contraintes incrémentales. Dans le cas de plusieurs itérations, la partition produite après une itération deviendrait le point de départ des modifications suivantes.
Nous comparons également les méthodes de propagation avec différents paramètres : pour la propagation par super-instances, nous faisons varier le nombre de super-instances créés selon une proportion de 30%, 40% et 50% du nombre de points total ; pour la seconde méthode, nous avons aussi fait les expériences pour 5, 10, 15 et 20 voisins.
Pour déterminer la performance du clustering, nous utilisons l'indice de Rand ajusté (ARI [8]) entre deux partitions P et P' qui s'exprime ainsi :

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

où a est le nombre de paires de points qui sont dans le même cluster dans les deux partitions, b est le nombre de paires de points dans des clusters différents dans les deux partitions, c est le nombre de paires de points dans le même cluster dans P et dans des clusters différents dans P' et d est le nombre de paires de points dans des clusters différents dans P et dans le même cluster dans P'. L'ARI est compris entre -1 et 1, une valeur de 1 dénotant des partitions identiques, et une valeur de 0 ou inférieure correspondant à des partitions indépendantes entre elles.

Qualité du clustering. Les résultats des comparaisons sont présentés en figure 4. La figure 4a présente les comparaisons des partitions trouvées par les différentes méthodes par rapport à la vérité des données. L'ARI entre la partition initiale et la vérité sert de référence. On peut constater que la partition initiale influe beaucoup sur les résultats car les méthodes ILP et CP se basent sur cette partition pour intégrer les contraintes. La partition trouvée par KMEANS possède en général un meilleur ARI que celles produites par HCA ou clustering spectral. On peut constater qu'en général, la modification produite par ILP ou CP améliore l'ARI quelle que soit la partition initiale. À noter que COP-KMEANS se comporte toujours de la même façon indépendamment de la partition initiale, puisque cet algorithme calcule une nouvelle partition à partir des données et des contraintes, sans tenir compte de la partition initiale.

La figure 4b présente les comparaisons des partitions produites par les méthodes par rapport à la partition initiale. On peut constater que COP-KMEANS produit une partition assez similaire à la partition initiale produite par KMEANS, car les deux méthodes visent la même fonction objectif. Cependant pour les partitions initiales produites par HCA ou le clustering spectral, COP-KMEANS ne peut pas garantir une partition similaire. On constate également que les méthodes ILP et CP produisent une partition très similaire à la partition initiale produite par KMEANS. Cependant, ILP s'écarte plus de la partition initiale dans les cas de HCA et du clustering spectral, alors que notre modèle est celui qui reste le plus proche de la partition de départ dans tous les cas sans propagation, et aussi dans bon nombre de cas où l'on effectue une propagation par les super-instances.

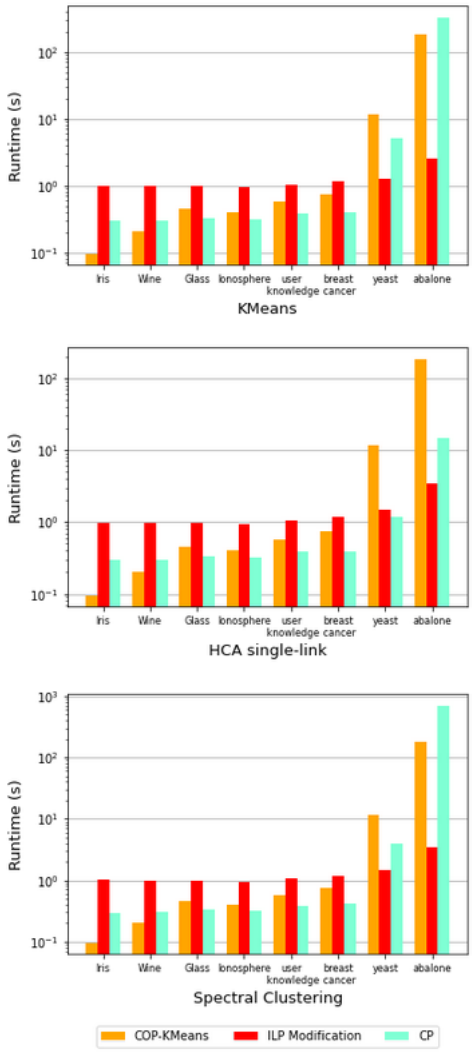


FIGURE 5 – Temps d'exécution en secondes des différentes méthodes (échelle logarithmique)

En effet le modèle ILP utilise le score d'allocation des points aux clusters pour réarranger tous les points, ce qui peut produire d'importantes modifications sur la structure des clusters, y compris sur les points non contraints. Notre modèle, en se restreignant sur les points ou super-instances

contraints, assure des modifications moins importantes tout en améliorant les résultats. De plus les propagations sont localisées au niveau des points contraints, ce qui les rend plus prévisibles pour l'expert.

Temps d'exécution. Nous avons mesuré l'efficacité des méthodes en termes de temps d'exécution. La figure 5 représente le temps d'exécution des méthodes en secondes. On peut constater que l'approche ILP est assez stable en temps alors que l'approche CP peut demander un temps d'exécution important lorsque le nombre de contraintes devient grand. Pour améliorer l'efficacité, une perspective est d'améliorer la stratégie de recherche pour atteindre rapidement des solutions proches de l'optimum.

4.1.2 Relâchement de contraintes

Le modèle ILP [11] ne permet pas de gérer les cas sur-contraints, par exemple lorsque les contraintes utilisateur sont conflictuelles. Notre modèle permet de gérer ces cas grâce à l'utilisation de la PPC. Afin de mettre en évidence l'intérêt de notre modèle dans le cas de problèmes sur-contraints, nous avons utilisé un problème de résolution de 40 contraintes sur Iris avec une partition de départ produite par KMEANS, dans lequel nous avons intentionnellement créé un réseau de trois contraintes rendant le problème insoluble ainsi que deux contraintes *must-link* contradictoires avec la vérité terrain entre des points très éloignés l'un de l'autre. Nous simulons ainsi le cas où des erreurs sont introduites dans le retour expert.

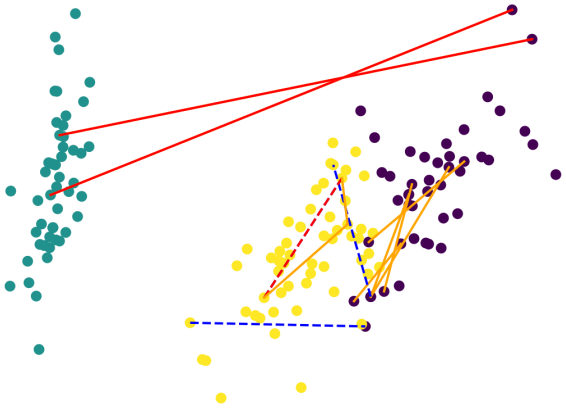


FIGURE 6 – Solution du problème sur-contraint ; les contraintes affichées ne sont pas satisfaites au lancement du modèle ; les contraintes en rouge sont relâchées par le modèle (*must-link* en trait plein, *cannot-link* en pointillés).

La solution obtenue en demandant la satisfaction de 37 contraintes sur les 40 est visible dans la figure 6. On constate que l'une des contraintes du réseau ainsi que les contraintes erronées sont relâchées. L'ARI entre la solution et la vérité terrain est de 0.802, alors qu'il est de 0.764 si l'on demande la satisfaction de 39 contraintes (toutes les contraintes ne pouvant pas être satisfaites à cause du réseau de contraintes contradictoires). Le choix par le modèle des contraintes à relâcher pour minimiser la fonction objectif permet donc dans une certaines mesure de corriger des erreurs dans le retour expert.

Par ailleurs, nous avons mesuré l'évolution du temps de calcul en fonction du taux de satisfaction pour estimer l'impact de la possibilité de relâcher les contraintes. Pour cette expérimentation, nous avons considéré le jeu de données *yeast* avec 10 jeux de 50 contraintes *must-link*/*cannot-link* en proportions égales. Les résultats sont présentés dans la figure 7. On constate que l'augmentation du temps de calcul est significative quand le taux de satisfaction est inférieur à 25% du nombre de contraintes. La variance du temps de calcul entre les jeux de contraintes est également très forte pour ces valeurs.

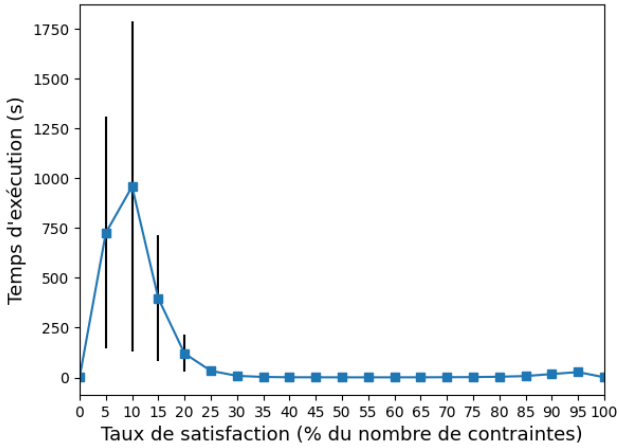


FIGURE 7 – Évolution du temps de calcul du modèle sur le jeu de données *yeast* en fonction de la proportion de contraintes à satisfaire. Les points sans barre d'erreur ont une variance trop faible pour être affichée.

4.2 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) [10]. 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.

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 pro-

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

gressif 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 8 présente la répartition en pixels des classes annotées et met en exergue la complexité de la tâche.



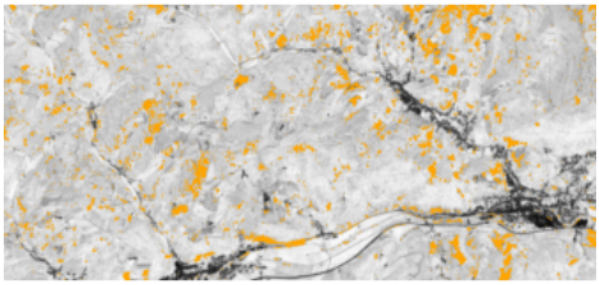
FIGURE 8 – Répartition des clusters labellisés sur la série d'images satellite du jeu de données *Coupes de bois*. La forêt est représentée en vert (156684 pixels), les zones d'activité humaine en bleu (86665 pixels) et les zones de coupes claires en jaune (639 pixels).

Présentation du problème. Étant donné que la seule classe pour laquelle nous avons des informations précises est celle des coupes de bois, nous nous donnons comme but de distinguer ce groupe du reste des données ; il s'agit ainsi d'un problème de clustering binaire.
Les données de départ décrivent l'évolution temporelle du NDVI des images satellites. Une donnée x_i est donc constituée de 11 valeurs NDVI $x_{i1}, \ldots, x_{i11}$. Pour chaque donnée, nous avons construit une série dérivée $x'_{i1}, \ldots, x'_{i10}$, en calculant la différence de NDVI entre chaque pas de temps $x'_{ij} = x_{ij+1} - x_{ij}$. Pour tenter de trouver un cluster proche des annotations d'experts, nous avons suivi la méthode décrite dans [10] consistant à effectuer un clustering avec $k = 15$ sur les données et à sélectionner le cluster avec la plus grande intersection avec la classe de coupes claires. Nous avons remarqué que le clustering sur les variations de NDVI permettait d'obtenir un cluster couvrant plus de points de coupes de bois que l'utilisation des données de base. Les données sélectionnées sont présentées en orange dans la figure 9a. Il s'agit d'un ensemble de 16183 pixels, dont 348 qui sont des coupes claires, soit plus de la moitié des points de coupe totaux.
On sépare une nouvelle fois ces données en deux en espérant distinguer ainsi les coupes de bois des autres évolutions similaires (fauchage, récoltes). Cette partition nous sert de point de départ. Elle contient un cluster de petite taille (561 points) contenant une majorité de points de coupe. C'est ce cluster qui est identifié comme les points de coupes. Nous représentons ce cluster dans la figure 9b, où les points de coupe réels (vrais positifs) sont indiqués en vert, et les points qui ne sont pas des coupes (faux positifs) sont indiqués en rouge. Le reste de la partition de départ reste affichée en orange. Les points de coupe non sélectionnés, qui sont donc absents de la partition de départ, sont indiqués en jaune. Ces zones sont présentées dans la figure 9c afin de mieux les positionner dans la zone géographique initiale.
Nous générons ensuite 25 contraintes sur des points de coupe de bois, sur la base des annotations d'experts. Ces contraintes servent de données d'entrée à notre modèle pour modifier la partition que nous avons construite. La figure 9b présente les contraintes générées, où les contraintes *must-link* sont en violet et *cannot-link* sont en bleu et pointillés.

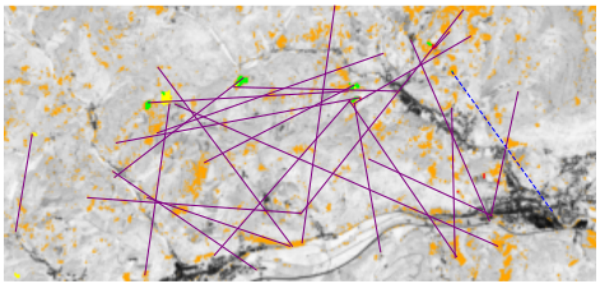
Résultats. Nous présentons dans la figure 9d le résultat obtenu avec la modification par notre modèle, sans utilisation de super-instances et avec la propagation aux 8 plus proches voisins, qui donne les meilleurs résultats d'après nos expériences. On peut constater que la modification en tenant compte de ces contraintes a permis de réduire le nombre de faux positifs. En particulier, la zone de faux positifs près de la zone de coupe au nord-est de la zone géographique est réduite après modification par notre modèle. La modification en elle-même est réalisée en 2.24 secondes en moyenne.
Nous avons analysé l'impact des paramètres sur le nombre de super-instances et le nombre de plus proches voisins, le résultat est présenté dans la figure 10. On peut constater qu'un très faible nombre de super-instances entraîne des modifications trop importantes, ce qui se traduit par une chute de l'ARI. Le meilleur résultat est obtenu pour un taux de super-instances équivalent à environ 50% du nombre de points de chaque cluster. La propagation par les plus proches voisins est cependant moins sensible à l'augmentation du nombre de voisins. Le choix d'un nombre de voisins entre 5 et 10 pour propager efficacement les modifications semble raisonnable. Enfin, le temps de génération des super-instances calculé en utilisant le clustering hiérarchique *complete-link* est de 8.73 secondes.

5 Conclusion

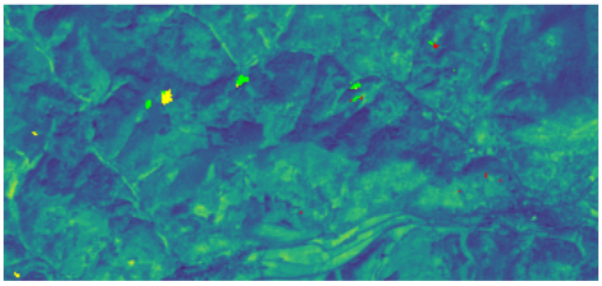
Nous avons développé un modèle en programmation par contraintes pour la modification d'un clustering, permettant d'une part l'intégration incrémentale de contraintes et d'autre part de préserver la structure des clusters. Il peut également traiter différents types de contraintes, dont des contraintes thématiques, et relâcher des contraintes pour ré-



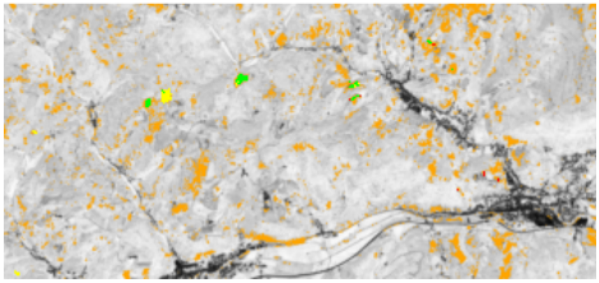
(a) En orange, le cluster couvrant le plus de coupes de bois.



(b) Mise en évidence des coupes de bois et des contraintes.

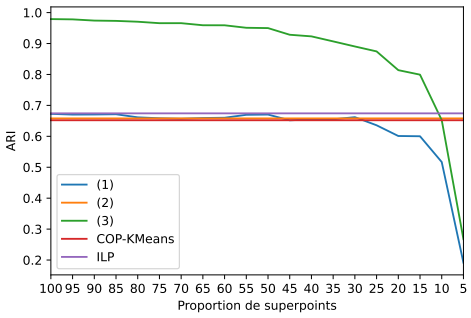


(c) Image d'origine.

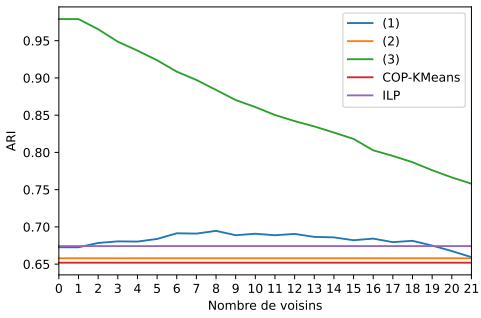


(d) Cluster modifié par notre modèle.

FIGURE 9 – Le cluster utilisé pour l'étude de cas est affiché en orange. La première image utilise les canaux d'origine, la seconde est en niveaux de gris selon la valeur NDVI. Les vrais positifs sont en vert, les faux positifs en rouge, les points de coupe absents des données considérées en jaune.



(a) Propagation par les super-instances



(b) Propagation par les proches voisins

FIGURE 10 – Évolution de l'ARI selon les méthodes de propagation sur la partition construite à partir des données de coupes de bois. On considère l'ARI entre la partition modifiée et la vérité terrain (1), entre la partition initiale et la vérité terrain (2), entre la partition initiale et la partition modifiée (3). En outre, l'ARI entre les résultats de COP-KMEANS (ou ILP) et la vérité terrain sont indiqués pour comparaison.

soudre des conflits ou corriger des erreurs dans le retour de l'expert. Ces atouts rendent ce modèle adapté à l'utilisation dans un contexte d'interaction avec un expert pour des tâches de clustering. L'efficacité des contraintes sur la qualité du clustering peut être améliorée par des processus d'apprentissage actif pour tirer au mieux avantage du retour expert. La contrainte du temps réel pour l'interaction signifie qu'il faut également étudier des stratégies de recherche adaptées pour obtenir rapidement des solutions proches de l'optimum.

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) [7].

Références

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

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

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

[3] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain. Constrained clustering by constraint programming. *Artificial Intelligence*, 244 :70–94, March 2017.

[4] Ian Davidson and S. S. Ravi. Agglomerative Hierarchical Clustering with Constraints : Theoretical and Empirical Results. In Alípio Mário Jorge, Luís Torgo, Pavel Brazdil, Rui Camacho, and João Gama, editors, *Knowledge Discovery in Databases : PKDD 2005*, Lecture Notes in Computer Science, pages 59–70, Berlin, Heidelberg, 2005. Springer.

[5] Ian Davidson, S. S. Ravi, and Leonid Shamis. A SAT-based Framework for Efficient Constrained Clustering. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 94–105, Columbus, OH, USA, April 2010. Society for Industrial and Applied Mathematics.

[6] 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.

[7] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38 :293–306, 1985.

[8] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1) :193–218, December 1985.

[9] Chia-Tung Kuo, S. S. Ravi, Thi-Bich-Hanh Dao, Christel Vrain, and Ian Davidson. A framework for minimal clustering modification via constraint programming. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, pages 1389–1395, San Francisco, CA, USA, February 2017. AAAI Press.

[10] 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.

[11] Nguyen-Viet-Dung Nghiem, Christel Vrain, Thi-Bich-Hanh Dao, and Ian Davidson. Constrained Clustering via Post-processing. In Annalisa Appice, Grigorios Tsoumakas, Yannis Manolopoulos, and Stan Matwin, editors, *Discovery Science*, Lecture Notes in Computer Science, pages 53–67, Thessaloniki, Greece, 2020. Springer International Publishing.

[12] Abdelkader Ouali, Samir Loudni, Yahia Lebbah, Patrice Boizumault, Albrecht Zimmermann, and Lakhdar Loukil. Efficiently finding conceptual clustering models with integer linear programming. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 647–654, New York, NY, USA, July 2016. AAAI Press.

[13] Toon Van Craenendonck, Sebastijan Dumančić, Elia Van Wolputte, and Hendrik Blockeel. COBRAS : Fast, Iterative, Active Clustering with Pairwise Constraints. *arXiv :1803.11060 [cs, stat]*, March 2018. arXiv : 1803.11060.

[14] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained k-means clustering with background knowledge. In Carla E. Brodley and Andrea Pohoreckyj Danyluk, editors, *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001*, pages 577–584. Morgan Kaufmann, 2001.

[15] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schroedl. Constrained K-means Clustering with Background Knowledge. page 8, 2001.

[16] Xiang Wang and Ian Davidson. Flexible constrained spectral clustering. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 563–572, New York, NY, USA, July 2010. Association for Computing Machinery.