

UNIVERSITÉ D'ORLÉANS

*École doctorale Mathématiques, Informatique, Physique
Théorique et Ingénierie des Systèmes*

Laboratoire d'Informatique Fondamentale d'Orléans

THÈSE présentée par :

Aymeric BEAUCHAMP

soutenue le : **29 septembre 2025**

pour obtenir le grade de : **Docteur de l'Université d'Orléans**

Discipline / Spécialité : **Informatique**

Clustering sous contraintes : intégration incrémentale et active des retours utilisateur

Thèse dirigée et encadrée par :

Mme Thi-Bich-Hanh DAO
M. Samir LOUDNI
Mme Christel VRAIN

Professeure des universités, Université d'Orléans
Professeur des universités, IMT Atlantique
Professeure émérite, Université d'Orléans

RAPPORTEURS :

M. Saïd JABBOUR
M. Nadjib LAZAAR

Professeur des universités, Université d'Artois
Professeur des universités, Université de Paris-Saclay

EXAMINATEURS :

M. Pierre GANÇARSKI
(Président du jury)
M. Bruno CRÉMILLEUX

Professeur des universités, Université de Strasbourg

Professeur des universités, Université de Caen-Normandie

Remerciements

Tout d’abord, il me faut évidemment remercier Bich, Samir et Christel pour avoir choisi de m’accompagner et pour m’avoir conseillé tant au niveau scientifique que professionnel pendant toutes ces années. Vous m’avez appris à naviguer dans le monde de la recherche, et j’espère que nous pourrons continuer à travailler ensemble. Je remercie aussi Abdelkader Ouali avec qui j’ai commencé la recherche en master et qui m’a amené à cette opportunité. La décision d’un algorithme d’affectation des projets tutorés est devenue un stage de recherche et maintenant une thèse.

Tout cela n’aurait pas été possible sans le projet ANR HERELLES et ses porteurs (en plus de Bich) : Pierre Gançarski, Antoine Cornuéjols, Bruno Crémilleux et Maguelonne Teisseire. Merci d’avoir animé le projet et organisé ces rencontres qui ont été l’occasion de discussions enrichissantes. J’en profite pour remercier Antoine, Baptiste, Hussein et Chenglin pour ne citer qu’eux.

Je remercie également Saïd Jabbour et Nadjib Lazaar, qui ont accepté d’être les rapporteurs de cette thèse. J’en suis honoré. Vos retours clairs et précis ont permis d’améliorer la rigueur et la qualité de ce manuscrit. J’en profite pour remercier l’association ParenThèse Île-de-France dont les retraits de rédaction ont été précieuses pour structurer la phase d’écriture.

Je dois aussi remercier tous les compagnons que j’ai rencontrés pendant mon parcours. D’abord, Étienne et Triss, en stage avec moi à Caen, qui ont de même tenté l’aventure de la thèse et que j’ai recroisés à diverses occasions depuis. Triss a notamment réalisé le prétraitement des données temporelles utilisées dans mes travaux. Ensuite, mes collègues de l’équipe CA avec qui j’ai débuté en 2021 : Badreddine, Frank-Anaël et particulièrement Mathieu que je vois en face de moi depuis mon arrivée et avec qui j’ai eu d’innombrables discussions sur tant de sujets. Enfin, l’ensemble des doctorants du LIFO : Antoine, Dao, Darine, Florian, Hai Yen, Joanne, Jordan, Lingchen, Maël, Nicolas, Samuel, Sharhzad, Sofiane, Thibaut, Tom, Trung-Anh, Yani. La solidarité et l’entraide entre doctorants est cruciale.

Merci au LIFO et à l’équipe CA de m’avoir accueilli, à l’intérieur et en dehors : Anaïs et Jules (sur les blocs), Anthony et Florent (dans les bars), Jean-Michel (vers les étoiles). Merci surtout à Isabelle pour son aide constante dans la traversée des méandres de l’administration.

Merci à mes amis et notamment Baptiste et Vincent, que j’ai pas vus aussi souvent que j’aurais dû pendant toutes ces années. Je compte bien remédier à cela.

Pour terminer, je tiens à remercier du fond du coeur ma famille pour son soutien continu depuis presque trois décennies. Je me suis éloigné, mais je pense toujours à vous.

Sommaire

Liste des figures	v
Liste des tables	ix
Liste des algorithmes	xii
1 Introduction	1
1.1 Organisation	2
I Préliminaires et état de l’art	3
2 Préliminaires	7
2.1 Notations	7
2.2 Partitionnement de données (<i>clustering</i>)	9
2.2.1 Critères de validation externe de clustering	11
2.2.2 Aire sous la courbe du budget de requêtes	13
2.3 Clustering sous contraintes	13
2.3.1 Contraintes utilisées	16
2.4 Programmation par contraintes	18
2.4.1 Problème de satisfaction de contraintes	18
2.4.2 Séparation et évaluation	20
2.4.3 Backtracking	21
2.4.4 Heuristiques de choix	22
2.4.5 Propagation de contraintes	23
2.4.6 Contraintes globales	25
2.4.7 Contraintes et insatisfiabilité	26
2.4.8 Recherche d’ensembles insatisfiables	27
2.5 Méthodologie et données	31
2.5.1 Comparaison par analyse bayésienne	31
2.5.2 Jeux de données utilisés dans les expériences	32
2.5.3 Cas d’étude : série temporelles d’images satellite	33
3 Clustering sous contraintes	37
3.1 Topologie succincte du clustering sous contraintes	37
3.2 Clustering sous contraintes incrémentales	40

3.2.1	Contraintes utilisateur et contraintes de l'utilisateur	42
3.2.2	Algorithmes de clustering sous contraintes incrémentales	42
3.2.2.1	Construction incrémentale de partition hiérarchique	43
3.2.2.2	Approches déclaratives	46
3.2.2.3	Clustering collaboratif	51
3.2.3	Discussion	53
4	Clustering actif	57
4.1	Apprentissage actif	58
4.1.1	Scénarios d'apprentissage	58
4.1.2	Stratégies de requête	59
4.2	Clustering actif	61
4.3	Voisinages	62
4.4	Travaux antérieurs en clustering actif	64
4.4.1	Algorithmes basés sur les voisinages	64
4.4.2	Apprentissage de contraintes triplet	70
4.4.3	Estimer le changement apporté à la partition par une nouvelle contrainte	71
4.4.4	Propager les contraintes avec les voisins	72
4.4.5	Ordonner les voisins de points étiquetés	74
4.4.6	Sélection séquentielle d'un ensemble de contraintes	74
4.4.7	Discussion	77
II	Modification interactive de clustering	81
5	Un cadre de clustering sous contraintes adapté à l'utilisateur	85
6	Un modèle en PPC pour la modification minimale pondérée de partition	91
6.1	Modification minimale pondérée de partition (CP-MWCM)	91
6.1.1	Fonction objectif et ancres	92
6.1.2	Généraliser les modifications	94
6.1.3	Formulation du modèle en PPC	95
6.1.4	Construction de l'ensemble de contraintes	96
6.1.5	Relâchement des contraintes	98
6.2	Expérimentations	99
6.2.1	Protocole expérimental	99
6.2.2	Paramétrage de CP-MWCM	100
6.2.3	Impact du nombre et du type de contraintes utilisateur sur le temps de calcul	101
6.2.4	Relâchement des contraintes	103
6.2.5	Comparaison avec l'état de l'art	104
6.2.6	Gestion des contraintes	106
6.2.7	Cas d'étude : coupes de bois	106

7	Rendre le clustering actif utilisable en temps réel	109
7.1	Élicitation basée sur les règles pour le clustering sous contraintes incrémentales (REACT)	110
7.1.1	Extraction des règles	110
7.1.2	Génération de requêtes	111
7.1.2.1	Aire de confiance	111
7.1.2.2	Heuristiques de requête	112
7.2	Expérimentations	113
7.2.1	Protocole expérimental	114
7.2.2	Pertinence des aires de confiance	115
7.2.3	Efficacité individuelle des heuristiques	115
7.2.4	Comparaison avec d'autres approches basées sur les voisinages . . .	117
7.2.4.1	Qualité	118
7.2.4.2	Temps de calcul	118
8	Expliquer les conflits et proposer la meilleure solution	121
8.1	Adéquation des contraintes à un modèle	122
8.2	Modèle PPC pour choisir le MCS d'adéquation minimale	124
8.3	Expérimentations	125
8.3.1	Protocole expérimental	126
8.3.2	Impact du modèle sur l'adéquation	127
8.3.3	Impact du retrait des contraintes sur le clustering	128
8.3.4	Passage à l'échelle du modèle	129
8.3.5	Cas d'étude : coupes de bois	130
9	Conclusion	133
9.1	Contributions	133
9.2	Perspectives d'amélioration	135
A	Implémentation de CoActive	137
B	NPU incrémental	139
	Bibliographie	140

Liste des figures

2.1	Exemples de partitions obtenues avec différents critères d'optimisation. Sur les mêmes données, les clusters résultants peuvent être très différents (2.1a, 2.1b et 2.1c). A l'inverse, deux critères différents peuvent converger vers la même partition (2.1b et 2.1d). Images extraites de Dao et Vrain (2024) . . .	12
2.2	Exemples illustratifs d'aires sous la courbe de budget pour l'ARI. Le Q-AUBC (ARI) est en bleu , le S-AUBC (ARI) en vert . Le nombre de requêtes est en abscisse. Dans cet exemple, chaque itération pose 10 requêtes.	14
2.3	Taxonomie de contraintes pour le clustering extraite de González-Almagro et al. (2025)	15
2.4	Exemple de contraintes ML/CL , triplet et d'étendue limitée . La partie verte (resp. orange) de la contrainte triplet représente la relation (x_a, x_p) (resp. (x_a, x_n)) entre le point de référence et le point posé comme similaire (resp. différent). La contrainte d'étendue limitée est représentée visuellement par un cercle englobant les points concernés ainsi que la restriction associée. En l'occurrence, les points rose soumis à la contrainte doivent être réaffectés soit au cluster violet, soit au cluster noir.	17
2.5	Méthodes de base pour trouver un MSS (<i>à gauche</i>) et un MUS (<i>à droite</i>) .	28
2.6	Simplexe obtenu suite à l'échantillonnage d'un modèle bayésien hiérarchique. Les points sont des échantillons, placés dans le simplexe selon les probabilités $P(A > B)$, $P(B > A)$ et $P_r(H_0)$ observées pour l'échantillon. Les probabilités obtenues au final correspondent au nombre d'échantillons dans chaque zone du simplexe par rapport au nombre total d'échantillons. Dans cet exemple, on peut tirer l'observation suivante : "la méthode CoActiVe avec une sélection aléatoire de contraintes a 89.5% de chances d'être meilleure que PCKMEANS".	32
2.7	Répartition des clusters sur la série d'images satellite du jeu de données <i>Coupes de bois</i> . 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).	34
2.8	La première image utilise les canaux d'origine, la seconde est en niveaux de gris selon la valeur NDVI. Le cluster utilisé pour l'étude de cas est affiché en orange dans la figure 2.8b. Les vrais positifs sont en vert, les faux positifs en rouge, les points de coupe absents des données considérées en jaune. . .	35
3.1	Exemple illustratif du fonctionnement de COBRAS, tiré de Craenendonck et al. (2018) . Les contraintes ML sont en vert et les CL en rouge	46
3.2	Schéma de fonctionnement de I-SAMARAH tiré de Lafabregue (2021) . . .	52

4.1	Problème illustratif de classification binaire avec apprentissage actif. Chaque rectangle est borné par les exemples étiquetés par l'utilisateur, soit positifs (les 1 à l'intérieur), soit négatifs (les 0 à l'extérieur) et délimite une partie de l'espace possible pour la classe positive. La zone d'incertitude (en gris hachuré) se trouve entre l'enveloppe convexe des exemples positifs et l'espace peuplé par les exemples négatifs. Image extraite de Cohn et al. (1994)	60
4.2	Exemple de voisinages obtenus avec la méthode MinMax après 15 requêtes sur le jeu de données <i>iris</i> . On distingue trois groupes de points distincts reliés par des contraintes ML (en rouge), et des contraintes CL entre toutes les paires de points de groupes différents (en bleu pointillé).	63
4.3	Exemple de chemin fort (en rouge) permettant de propager la contrainte CL entre t et l aux points u et v . Illustration tirée de Vu et al. (2012)	73
4.4	Exemple de requête proposée par ALNO sur des images de galaxie. L'utilisateur doit donner un ordre sur les voisins par rapport à leur ressemblance à l'image de référence. Illustration tirée de Qian et al. (2013)	74
4.5	$v_{ij}^{out} = \frac{d_{ij}^{out}}{d_{ij}}$ est la proportion de la distance entre i et j hors de toute région détectée. Les points utilisés comme vecteurs de support sont en vert. Illustration tirée de Abin et Beigy (2014)	76
4.6	Chronologie des algorithmes de clustering actif présentés.	79
5.1	Schéma de CoActiVe.	86
5.2	Illustration de CoActiVe sur 10 itérations sur le jeu de données <i>1sun</i> , à partir d'une partition de KMEANS (Fig. 5.2a). Les figures successives montrent l'évolution de la partition après t itérations. L'ARI (cf. Définition 2.3) de la partition par rapport à la vérité terrain est reporté sur chaque figure.	89
6.1	Position des ancres pour différentes valeurs de α calculées à partir d'une partition de KMEANS sur le jeu de données <i>1sun</i> . Les ancres sont représentées avec une taille plus grande que les autres points.	94
6.2	Exemple de contraintes émergentes : une paire de points appartenant à deux super-instances $\mathcal{S}_1$ et $\mathcal{S}_2$ sont soumises à une contrainte <i>must-link</i> (en orange), tandis qu'une autre paire est soumise à une contrainte <i>cannot-link</i> (en bleu pointillés). Il se trouve alors que $\mathcal{S}_1$ et $\mathcal{S}_2$ sont en même temps soumis à une contrainte <i>must-link</i> et une contrainte <i>cannot-link</i> , ce qui rend le problème insoluble.	95
6.3	Comparaison du nombre de victoires pour chaque configuration (α, β) avec un modèle hiérarchique bayésien, selon le Q-AUBC (horizontal) ou le S-AUBC (vertical). Les meilleures valeurs sont proches du haut à droite, avec les configurations sur le front de Pareto en orange	100
6.4	Vue en simplexe de la comparaison bayésienne de deux configurations selon le Q-AUBC (<i>ARI</i>).	101
6.5	Évolution du temps de calcul du modèle PPC pour <i>letters</i> et <i>MNIST</i> en fonction du nombre et du type de contraintes, avec les intervalles de confiance à 95% (échelle logarithmique).	102

6.6	ARI de la partition existante à chaque itération, comparée avec la vérité terrain (à gauche) ou la partition précédente partition (à droite), pour le jeu de données <code>glass</code> . Les valeurs sont les moyennes sur 90 essais avec intervalle de confiance à 95%. Les valeurs les plus hautes sont les meilleures.	104
6.7	Comparaison de CoActiVe avec CP-MWCM et NPU avec les alternatives sur le Q-AUBC (<i>ARI</i>).	105
6.8	Comparaison bayésienne de CoActiVe avec ou sans NPU pour le S-AUBC pour les trois métriques.	105
6.9	Comparaison du temps de calcul des méthodes considérées (échelle logarithmique). Les méthodes utilisant NPU sont hachurées.	106
6.10	Temps de calcul pour les trois modalités de gestion des contraintes : once pour la prise en compte de toutes les contraintes en une seule fois, incr pour le fonctionnement incrémental par défaut où seul le lot de contraintes courant est utilisé et cumul pour la stratégie de cumul des lots de contraintes au cours du temps (échelle logarithmique).	107
6.11	Comparaison bayésienne des deux modalités incrémentales : cumul des contraintes au cours du temps ou intégration des seules contraintes de l'itération courante	107
6.12	Vues de la modification sur la zone spatiale ; les images montrent le cluster des "positifs", avant (6.12a) et après (6.12b) la modification. Les vrais positifs sont marqués en vert , les faux négatifs en jaune et les faux positifs en violet . Les pixels en noir et blanc ne font pas partie du cluster des positifs. Visionnage en couleurs conseillé.	108
7.1	Exemple illustratif du concept d'aires de confiance sur le jeu de données <code>1sun</code> .	113
7.2	ARI des heuristiques isolées sur le jeu de données <code>glass</code> pour chaque AC. Les résultats obtenus avec NPU sont pris comme référence.	116
7.3	ARI pour les jeux de données testés.	118
7.4	Temps de calcul des méthodes comparées sur les jeux de données testés en secondes (échelle logarithmique).	119
8.1	Évolution de $ADQ_{\mathcal{M}}(Triplet(x_{ref}, x_+, x_-))$ en fonction de la différence $ap - an$ pour 10000 paires différentes de valeurs (ap , an).	124
8.2	Temps nécessaire pour mesurer l'adéquation de 100 contraintes de chaque type en utilisant divers modèles sur le jeu de données <code>yeast</code> (échelle logarithmique). La présence de <i>mod</i> dans le nom d'un modèle signifie que le calcul a été effectué après modification de la partition avec les contraintes.	128
8.3	Évolution du temps de calcul de MCS pour différentes tailles d'ensembles de contraintes (échelle logarithmique).	129

Liste des tableaux

2.1	Table des symboles utilisés dans ce document.	8
2.2	Exemple de jeu de données.	10
2.3	Caractéristiques des jeux des données considérés, avec N le nombre de points, A le nombre d'attributs et K le nombre de clusters.	33
2.4	Dates des prises de vue de la série temporelle et nombre d'apparitions de zones de coupes claires à chaque période	34
3.1	Taille de l'espace de recherche pour chaque type de contrainte considérée en Section 2.3.1).	41
3.2	Complexité du modèle de Kuo et al. (2017) en termes de variables et de contraintes. Les domaines des variables indiquent les valeurs pouvant leur être affectées. La portée d'une contrainte est le nombre de variables qui y sont impliquées. E_t est l'étendue d'un attribut quelconque t (cf. Définition 2.1) parmi les m attributs de $\mathcal{X}$	49
3.3	Complexité du modèle de (Nghiem et al., 2020) en termes de variables et de contraintes.	51
3.4	Comparaison des approches présentées dans la Section 3.2.2. La colonne "Sélection" indique la méthode utilisée pour obtenir les contraintes. La colonne "Modification" indique la méthode employée pour recalculer ou modifier la partition existante. On entend par "manuelle" une étape qui ne propose aucune aide à l'utilisateur.	55
4.1	Complexité temporelle en moyenne des méthodes basées sur les voisinages.	69
4.2	Comparaison des approches présentées dans la Section 4.4.	78
6.1	Nombre de victoires pour chaque configuration (α, β) avec un modèle hiérarchique bayésien. Les valeurs entre parenthèses indiquent le nombre de cas où la probabilité de la configuration gagnante est plus grande que 95%. Les valeurs de α et β sont exprimées en pourcentages.	101
6.2	Proportion moyenne de contraintes CL, groupée par jeu de données, configuration et taille.	103
6.3	Étude comparative pour le clustering avec contraintes ML/CL avec relâchement sur le jeu de données mk2. Les métriques sont l'ARI avec la vérité terrain (Qualité), l'ARI avec KMEANS (Similarité), le temps de calcul et le nombre de contraintes relâchées dans la solution (n_r).	104
7.1	Stratégies employées pour trouver des points candidats pour des requêtes.	112

7.2	Tableau des heuristiques selon le statut du point et sa présence dans l'AC.	113
7.3	Taux de ML entre un exemple d'une règle et un point certifié qui sont en accord avec la vérité terrain, dans l'aire de confiance ($rate_{in}$) et en dehors ($rate_{out}$), pour la première itération de CoActiVe avec REACT. Le meilleur score pour une AC donnée est en <i>italique</i> , le meilleur score global est en gras .	115
7.4	Taux de ML entre un exemple d'une règle et un point certifié qui sont en accord avec la vérité terrain, dans l'aire de confiance ($rate_{in}$) et en dehors ($rate_{out}$), pour la dixième itération de CoActiVe avec REACT. Le meilleur score pour une AC donnée est en <i>italique</i> , le meilleur score global est en gras .	116
7.5	Nombre de contraintes <i>must-link</i> et <i>cannot-link</i> générées en utilisant uniquement l'heuristique H1, H2 ou H3 sur les jeux de données testés. Le ratio $\frac{ML}{CL}$ montre combien de contraintes ML ont été générées en plus par rapport aux contraintes CL.	117
8.1	Modèles supervisés testés, leurs paramètres, et la fonction utilisée pour SIM. Les paramètres non mentionnés utilisent la valeur par défaut de <i>scikit-learn</i> .	127
8.2	Adéquation et informativité pour différents modèles et types de contraintes, avec intervalle de confiance à 95%.	127
8.3	Évolution de l'ARI d'une partition initiale (ARI_0) après retrait d'un MCS minimisant (1) l'adéquation (ARI_{ADQ}), (2) la cardinalité (ARI_{min}) ou (3) l'informativité ARI_{inf}	129
8.4	Pourcentage de bruit corrigé (Ajout) ou différence entre l'adéquation moyenne des annotations et l'adéquation moyenne du bruit en points de pourcentage (Remplacement).	130

Liste des algorithmes

1	KMEANS (MacQueen, 1967)	10
2	Séparation et Évaluation (Branch and Bound)	21
3	Backtrack simple	22
4	Fonction REVISE	24
5	Algorithme AC3	24
6	BT avec <i>forward checking</i>	25
7	grow	28
8	shrink	28
9	DAA (<i>Dualize And Advance</i>)	29
10	MARCO (Mapping Regions of Constraint sets)	29
11	VIOLATE-CONSTRAINTS	38
12	Clustering sous contraintes incrémentales	41
13	COBRA (<i>CO</i> nstraint-Based Repeated Aggregation) (Van Craenendonck et al., 2017)	44
14	COBRAS (<i>CO</i> nstraint-Based Repeated Aggregation and Splitting) (Craenendonck et al., 2018)	45
15	analyze_relation	47
16	I-SAMARAH - Perturbation des agents avec les contraintes	52
17	EXPLORE	65
18	CONSOLIDATE	66
19	MINMAX-CONSOLIDATE	66
20	NPU	68
21	MOSTINFORMATIVE	68
22	Active Learning from Relative Comparisons	71
23	Active-HACC : Active Image Clustering with Pairwise Constraints	72
24	ACTIVE CONSTRAINT SELECTION (ACS)	73
25	CoActiVe	87
26	CP-MWCM	93
27	Rule-based Elicitation for Active Constraint-incremental clusTering (REACT)	110
28	EXTRACTIONRÈGLES	111
29	KNOWLEDGEEXPLOITATION	114
30	NPU incrémental	139

Chapitre 1

Introduction

Le problème du *clustering* est une tâche courante en fouille de données visant à organiser automatiquement les points d'un jeu de données en groupes cohérents. Cette méthode d'apprentissage non supervisé trouve des applications dans de nombreux domaines, tels que la découverte de communautés dans les réseaux sociaux, l'identification de groupes de patients à partir de données médicales ou la segmentation d'images satellite. Contrairement à la classification supervisée, le clustering ne s'appuie pas sur des étiquettes prédéfinies, ce qui le rend particulièrement utile pour l'exploration de données. Cependant, la partition optimale calculée par un algorithme peut ne pas correspondre à la structure attendue par l'utilisateur, et l'hypothèse sous-jacente d'une unique partition pertinente est souvent restrictive.

Pour résoudre cette problématique, le clustering sous contraintes introduit une approche semi-supervisée, intégrant des informations partielles – sous forme de contraintes ou d'étiquetage limité – pour guider la construction des groupes en fonction des attentes de l'utilisateur. Toutefois, les méthodes existantes supposent généralement que l'ensemble des contraintes nécessaires est disponible en amont du clustering. En pratique, l'utilisateur lui-même sait rarement quelles informations donner pour représenter efficacement ses connaissances.

C'est dans ce contexte que s'inscrit cette thèse, qui aborde le clustering sous contraintes dans un cadre incrémental et interactif. L'objectif est d'améliorer itérativement une partition en collaboration avec un ou une expert(e) (*human-in-the-loop*). Ce cadre pose plusieurs défis majeurs : il faut à la fois limiter le nombre d'interactions avec l'expert(e) pour préserver son engagement, exploiter au mieux les retours utilisateur déjà collectés, et assurer une continuité entre les partitions successives afin de ne pas dérouter l'utilisateur. La possibilité de conflits entre les retours utilisateur doit également être prise en compte.

Le projet ANR HERELLES dans lequel s'inscrit ce travail vise à intégrer de façon incrémentale des connaissances utilisateur dans un système de clustering collaboratif, avec une application à la détection d'évolutions temporelles dans des séries temporelles d'images satellite (STIS). Pour répondre aux enjeux de ce cadre interactif, nous proposons d'abord une méthode permettant de modifier une partition de façon minimale tout en respectant les contraintes, afin de maintenir la continuité entre les itérations. Nous présentons ensuite une méthode de clustering actif pour proposer des contraintes pertinentes à partir des retours utilisateur passés. Enfin, nous définissons une nouvelle mesure pour identifier les contraintes à enlever en cas de conflit à l'aide des retours utilisateur antérieurs, offrant

ainsi une capacité d'explication des décisions du système.

1.1 Organisation

Dans la première partie I, nous passons en revue les notions abordées dans le cadre de cette thèse, en clustering et en programmation par contraintes, dans le Chapitre 2. Nous faisons ensuite l'état de l'art de deux domaines : le clustering sous contraintes incrémentales (Chapitre 3) et le clustering actif (Chapitre 4). Nous positionnons nos contributions par rapport à cet état de l'art. Dans la deuxième partie II, nous présentons les contributions de cette thèse : un système de clustering sous contraintes incrémentales (CoActiVe, Chapitre 5), un modèle en programmation par contraintes pour la modification minimale pondérée de clustering (CP-MWCM, Chapitre 6), un algorithme de clustering actif adapté au cadre incrémental (REACT, Chapitre 7) et une mesure d'adéquation de contrainte à un modèle pour le calcul d'ensembles de correction minimaux (Chapitre 8). Le dernier chapitre 9 fait la synthèse des travaux produits et pointe les pistes de recherche ouvertes.

Les travaux réalisés pendant la thèse ont donné lieu à plusieurs publications nationales comme internationales dont voici la liste :

- CoActiVe et CP-MWCM ont d'abord été présentés aux Journées francophones de programmation par contraintes en 2022 dans l'article « *Intégration incrémentale de contraintes pour le clustering avec la programmation par contraintes* » (Beauchamp et al., 2022) ;
- ces travaux ont été renforcés sur la partie expérimentale ainsi qu'avec l'ajout des contraintes d'étendue limitée et publiés à la *International Conference on Principles and Practice of Constraint Programming* (CP'23) dans l'article « *Incremental Constrained Clustering by Minimal Weighted Modification* » (Beauchamp et al., 2023) ;
- Les travaux sur REACT ont été présentés dans l'article « *Rule-Based Constraint Elicitation For Active Constraint-Incremental Clustering* » *International Conference on Tools with Artificial Intelligence* (ICTAI'24) (Beauchamp et al., 2024) ;
- les travaux sur la mesure d'adéquation ont été présentés aux JFPC 2025 (Beauchamp et al., 2025) dans l'article « *Ensemble de correction minimal basé sur l'adéquation des contraintes en clustering* » ;
- enfin, nous avons soumis un article pour une édition spéciale du *Journal of Classification* portant sur le clustering dans laquelle nous compilons les travaux sur CoActiVe, CP-MWCM et REACT tout en approfondissant les expérimentations et en présentant des travaux préliminaires sur la gestion de conflits dans le système.

Première partie

Préliminaires et état de l'art

Dans cette partie, nous présentons les notions fondamentales utilisées dans nos travaux et nous positionnons ceux-ci dans le contexte des domaines étudiés.

Nous introduisons d'abord les notions importantes liées au clustering et à la programmation par contraintes (Chapitre 2). Nous présentons aussi les aspects méthodologiques liés aux expérimentations réalisées afin d'assurer la bonne compréhension des résultats présentés ultérieurement.

Nous nous focalisons ensuite sur le clustering sous contraintes et en particulier le clustering sous contraintes incrémentales (Chapitre 3). Nous décrivons les enjeux liés à ce contexte spécifique et faisons un état de l'art des méthodes qui s'y rattachent.

Nous introduisons enfin le domaine de l'apprentissage actif et du clustering actif (Chapitre 4) avant de décrire le fonctionnement d'algorithmes importants dans le domaine que nous utilisons comme référence expérimentale.

Chapitre 2

Préliminaires

Table des matières

2.1	Notations	7
2.2	Partitionnement de données (<i>clustering</i>)	9
2.2.1	Critères de validation externe de clustering	11
2.2.2	Aire sous la courbe du budget de requêtes	13
2.3	Clustering sous contraintes	13
2.3.1	Contraintes utilisées	16
2.4	Programmation par contraintes	18
2.4.1	Problème de satisfaction de contraintes	18
2.4.2	Séparation et évaluation	20
2.4.3	Backtracking	21
2.4.4	Heuristiques de choix	22
2.4.5	Propagation de contraintes	23
2.4.6	Contraintes globales	25
2.4.7	Contraintes et insatisfiabilité	26
2.4.8	Recherche d'ensembles insatisfiables	27
2.5	Méthodologie et données	31
2.5.1	Comparaison par analyse bayésienne	31
2.5.2	Jeux de données utilisés dans les expériences	32
2.5.3	Cas d'étude : série temporelles d'images satellite	33

Dans ce chapitre introductif, nous présentons le contexte général dans lequel s'inscrit la thèse en rappelant les notions et enjeux fondamentaux liés au clustering (Sections 2.1 et 2.2). Nous nous intéressons ensuite au clustering sous contraintes (Section 2.3) pour introduire les contraintes utilisées dans la thèse. Nous présentons ensuite le paradigme de la programmation par contraintes que nous utilisons dans nos contributions (Section 2.4).

Enfin, nous faisons un état de l'art du domaine spécifique de la thèse : le clustering sous contraintes incrémentales (Section 3.2).

2.1 Notations

Dans le cadre de cette thèse en clustering sous contraintes, il est essentiel de définir les notations et conventions utilisées afin d'assurer une lecture cohérente des analyses et

$\mathcal{X}$	Jeu de données ou espace de clustering
n	Nombre de points du jeu de données
m	Nombre d'attributs du jeu de données
$\mathcal{K}$	Espace des connaissances (attributs interprétables)
$\mathcal{P}$	Partition
$\mathcal{P}_t$	Partition calculée à l'itération t
k	Nombre de clusters
C_i	$i^{\text{ème}}$ cluster d'une partition
p_x	Affectation du point x dans une partition
$\mathcal{C}$	Ensemble de contraintes
$\mathcal{C}_t$	Ensemble de contraintes utilisé à l'itération t
L_t	Lot de contraintes obtenu à l'itération t

TABLE 2.1 – Table des symboles utilisés dans ce document.

résultats présentés. Cette section regroupe les symboles et la terminologie employés pour décrire tant les algorithmes de l'état de l'art que les nôtres.

Jeu de données Un jeu de données contenant n points¹ et m attributs² sera noté $\mathcal{X} = \{x_1, \dots, x_n\}$ avec $x_i = (x_{i,1}, \dots, x_{i,m})$ un point i et $x_{i,j}$ la valeur de l'attribut j pour le point i . La notation $\mathcal{X}$ est utilisée pour représenter un jeu de données quelconque utilisé pour le clustering. La notation $\mathcal{K}$ est utilisée dans le chapitre 7 pour noter un jeu de données dont les attributs sont facilement compréhensibles par l'utilisateur(e). $\mathcal{K}$ partage les mêmes n points de $\mathcal{X}$ et peut contenir tout ou partie de ses attributs (ou n'en contenir aucun et être uniquement composé d'attributs distincts de $\mathcal{X}$).

Partition On note $\mathcal{P} = \{C_1, \dots, C_k\}$ la partition de $\mathcal{X}$ en k clusters. $C_i \subseteq \mathcal{X}$ est le $i^{\text{ème}}$ cluster de $\mathcal{P}$. On peut aussi représenter $\mathcal{P}$ sous la forme $[p_1, \dots, p_n]$ avec p_i le numéro du cluster associé au point i de $\mathcal{X}$.

Contraintes On note $\mathcal{C} = \{c_1, \dots, c_h\}$ l'ensemble des contraintes utilisateur. On note L le lot de contraintes obtenues à une itération donnée (directement de l'utilisateur(e) ou par clustering actif).

Nos contributions sont focalisées sur un contexte incrémental, où chaque itération produit sa propre partition, ses propres contraintes. Par conséquent, on note $\mathcal{P}_t$, $\mathcal{C}_t$, L_t les partitions, contraintes et lots associés à l'itération t .

Les notations utilisées tout au long du manuscrit sont résumées en Table 2.1. Elles sont notamment utilisées dans les algorithmes de l'état de l'art sans toujours être rappelées auparavant.

1. Les termes *instance*, *objet*, *observation*, *transaction* sont aussi employés en fonction du contexte et des auteur(e)s. Par souci de clarté, nous n'emploierons le terme *point* par la suite.

2. De manière analogue, nous n'utiliserons que le terme *attribut* mais les termes *dimension*, *variable*, *propriété*, *descripteur* se retrouvent également dans la littérature.

2.2 Partitionnement de données (*clustering*)

Dans un processus de fouille de données, le clustering (Gan et al., 2020) est une étape exploratoire visant à dégager une structure générale des données³. Elle va guider les étapes suivantes : effectuer un traitement sur le jeu de données (réduire le bruit par exemple), choisir un algorithme adapté pour de l'apprentissage supervisé, chercher une description symbolique des données. . .

Pour résoudre le problème, les algorithmes de clustering peuvent se baser sur un critère d'optimisation et chercher la meilleure partition pour ce critère. Parmi les critères couramment utilisés, on trouve la somme des distances intra-cluster (*within cluster sum of distances* (WCSD)), la somme des carrés intra-cluster (*within cluster sum of squares* (WCSS)) utilisée par KMEANS (MacQueen, 1967, cf. Algorithme 1), la variance intra-cluster, le diamètre (distance maximale entre deux points du même cluster), la séparation (distance minimale entre deux points de clusters différents) ou les fonctions de vraisemblance pour les algorithmes basés sur des modèles statistiques. D'autres approches n'utilisent pas de critère d'optimisation. C'est le cas de DBSCAN (Ester et al., 1996) qui construit les clusters en fonction de leur densité via une distance de voisinage en-dessous de laquelle deux points sont considérés comme proches et groupés ensemble.

En outre, certains algorithmes ne produisent pas directement des partitions mais des dendrogrammes hiérarchisant les points sur plusieurs niveaux, du cluster unique contenant le jeu de données entier à une partition groupant chaque point dans son propre cluster. Chaque niveau de la hiérarchie correspond alors à une partition des données avec une granularité variable. D'autres ne regroupent pas l'ensemble des points et considèrent les points non groupés comme des anomalies que l'on peut choisir d'ignorer, de traiter comme un cluster supplémentaire ou avec des méthodes spécialisées. En outre, certains algorithmes ne calculent pas une affectation claire de chaque point à un cluster mais plutôt des scores d'appartenance à chacun des clusters : ce sont des partitions souples. Enfin, les variations dans le fonctionnement des algorithmes produisent des clusters de forme très différente. KMEANS ne peut former que des clusters sphériques autour des centroïdes, DBSCAN des clusters denses de forme et de taille arbitraire. La figure 2.1 illustre cela en montrant les différentes partitions des mêmes points obtenues en utilisant quatre critères d'optimisation distincts.

Dans nos travaux, nous traitons de partitions classiques, où chaque point est affecté à un seul cluster, et nous réalisons des modifications de partition à l'aide d'un critère d'optimisation. Nous nous situons par conséquent du côté des algorithmes de clustering à base de distance comme KMEANS. Il convient de définir ces termes avant de donner une définition formelle du problème.

Définition 2.1 (Jeu de données). Un **jeu de données** $\mathcal{X} = \{x_1, \dots, x_n\}$ est un ensemble de n **points** $x_i = (x_{i,1}, \dots, x_{i,m})$ définis par leurs valeurs pour un ensemble de m **attributs**. L'**étendue** E_f d'un attribut f correspond à l'ensemble des valeurs prises pour cet attribut

3. Comme l'explique MacQueen (1967) : « *The point of view taken in this application is not to find some unique, definitive grouping, but rather to simply aid the investigator in obtaining qualitative and quantitative understanding of large amounts of N -dimensional data by providing him with reasonably good similarity groups.* »

Algorithme 1 KMEANS (MacQueen, 1967)

Entrée: jeu de données $\mathcal{X}$, nombre de clusters k , nombre maximum d'itérations $\text{iter}_{\max}$

Sortie: centroïdes finaux $M = \{\mu_1, \mu_2, \dots, \mu_k\}$, partition $\mathcal{P}$ de $\mathcal{X}$

```

1: Choisir aléatoirement  $k$  centroïdes initiaux  $M = \{\mu_1, \mu_2, \dots, \mu_k\}$ 
2: Initialiser  $\mathcal{P} = \{p_1, \dots, p_n\}$ 
3:  $\text{iter} \leftarrow 0$ 
4:  $\text{convergence} \leftarrow \text{Faux}$ 
5: tant que  $\text{iter} < \text{iter}_{\max}$  et  $\text{convergence} = \text{Faux}$  faire
6:   pour chaque point  $x_i \in \mathcal{X}$  faire ▷ Affectation des points de  $\mathcal{X}$ 
7:     Calculer la distance entre  $x_i$  et chaque centroïde  $\mu_j$ 
8:      $p_i \leftarrow \arg \min_{j \in \{1, \dots, k\}} \|x_i - \mu_j\|^2$  ▷ Affecter au cluster le plus proche
9:    $M_{\text{old}} \leftarrow M$ 
10:  pour chaque cluster  $C_j \in \{C_1, \dots, C_k\}$  faire ▷ Mise à jour des centroïdes
11:     $\mu_j \leftarrow \frac{1}{|C_j|} \sum_{x \in C_j} x$  ▷ Mettre à jour le centroïde comme la moyenne du cluster
12:  si  $M = M_{\text{old}}$  alors ▷ Arrêter quand les centroïdes ne changent plus
13:     $\text{convergence} \leftarrow \text{Vrai}$ 
14:   $\text{iter} \leftarrow \text{iter} + 1$ 
15: retourner  $M, \mathcal{P}$ 

```

par l'ensemble des points. On peut qualifier un jeu de données selon le type d'attribut qu'il possède :

- **binaire** si ses attributs ont pour étendue $\{0 ; 1\}$.
- **numérique** si ses attributs ont pour étendue un ensemble inclus dans $\mathbb{N}, \mathbb{Z}$ (on parle alors de jeu de données à valeurs entières) ou $\mathbb{R}$ (valeurs réelles).
- **catégoriel** si ses attributs ont pour étendue un ensemble de modalités qualitatives.
- **mixte** si ses attributs n'ont pas tous la même étendue (par exemple, des attributs binaires et des attributs numériques).

Exemple 2.1. La table 2.2 est un jeu de données mixte composé de 5 points et 6 attributs (3 numériques, 2 binaires, 1 catégoriel).

Point	Taille	Poids	Âge	Fumeur	Sportif	Sexe
x_1	182	74	74	0	0	f
x_2	176	99	74	0	0	m
x_3	167	73	28	0	1	f
x_4	190	74	76	1	1	f
x_5	153	76	52	1	0	m

TABLE 2.2 – Exemple de jeu de données.

Définition 2.2 (Partition). Soit $\mathcal{X}$ un jeu de données. On appelle **partition** $\mathcal{P}$ de $\mathcal{X}$ tout ensemble de parties (appelées **clusters**) de $\mathcal{X}$ tel que :

1. aucun cluster n'est vide;

$$\forall C \in \mathcal{P}, C \neq \emptyset \quad (2.1)$$

2. l'union des clusters est égale à l'ensemble du jeu de données ;

$$\bigcup_{C \in \mathcal{P}} C = \mathcal{X} \quad (2.2)$$

3. les clusters sont deux à deux disjoints.

$$\forall C_1, C_2 \in \mathcal{P}, C_1 \neq C_2 \implies C_1 \cap C_2 = \emptyset \quad (2.3)$$

Cette définition est identique à celle d'une partition en théorie des ensembles.

Exemple 2.2. *L'ensemble $\{\{x_1, x_3, x_4\}, \{x_2\}, \{x_5\}\}$ est une partition en 3 clusters (ou 3-partition) du jeu de données de l'exemple 2.1. On note les affectations des points par $[1, 2, 1, 1, 3]$.*

Problème (Clustering à base de distance). Soit $\mathcal{X}$ un jeu de données et d une mesure de similarité. La tâche de **clustering** consiste à trouver une partition $\mathcal{P}$ de $\mathcal{X}$ telle que :

1. les points dans le même cluster minimisent d (maximiser l'homogénéité des clusters) ;
2. les points de clusters distincts maximisent d (maximiser l'hétérogénéité entre les clusters).

Derrière une définition relativement simple du clustering se cache un « problème mal défini »⁴ (Ackerman et al., 2010) : la notion de similarité est facile à appréhender, mais elle a beaucoup de traductions mathématiques qui ne convergent pas toutes entre elles, de sorte que le choix de la métrique utilisée pour résoudre un problème de clustering introduit nécessairement un biais dans le résultat obtenu, comme le montre la Figure 2.1. Ce n'est pas le seul : les algorithmes dépendent souvent d'hyperparamètres ayant une forte influence sur le résultat — notamment le nombre de clusters recherchés — et vont aussi favoriser certains types de structures par rapport à d'autres. Cornuéjols et al. (2018) expliquent même que le problème est « mal posé »⁵ étant donné que les hypothèses sur la partition idéale (des clusters denses clairement séparés entre eux) ne sont pas forcément souhaitables. Enfin, le concept même d'une partition idéale émergeant de la structure des données est depuis longtemps remis en cause : la qualité d'une partition dépend avant tout de l'utilisation qui peut en être faite, et donc du contexte d'application dans lequel s'inclut la procédure de clustering (Dasgupta et Ng, 2009 ; Von Luxburg et al., 2011).

2.2.1 Critères de validation externe de clustering

Estimer la qualité d'une partition produite par un algorithme de clustering est une tâche difficile, puisqu'il n'existe a priori pas d'étiquetage des données auquel se comparer. Néanmoins, des mesures ont été développées pour évaluer certaines propriétés d'une partition, ou la comparer à une autre partition. Dans le premier cas, on parle de critères de validation **interne** : l'évaluation repose uniquement sur les données groupées, pour estimer leur qualité par rapport à une certaine définition d'un cluster "idéal" (souvent un cluster compact et bien séparé des autres). Dans le second cas, ce sont des critères de validation **externe** qui mesurent la proximité d'une partition à une référence (étiquetage connu des données, partition obtenue par un autre algorithme...). Dans nos expériences, nous utilisons des critères externes dont nous donnons ici les définitions.

4. « *an ill-defined problem* » dans l'article d'origine.

5. « *an ill-posed problem* » dans l'article d'origine.

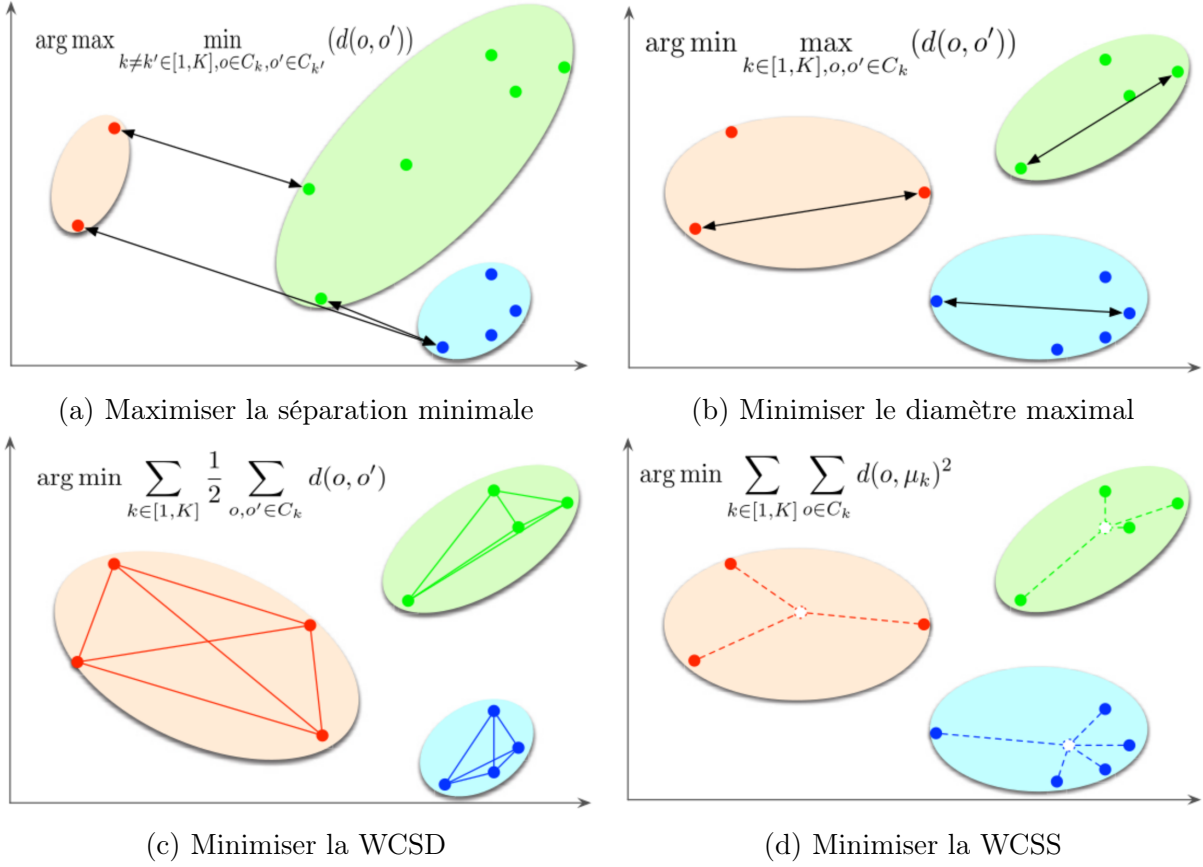


FIGURE 2.1 – Exemples de partitions obtenues avec différents critères d’optimisation. Sur les mêmes données, les clusters résultants peuvent être très différents (2.1a, 2.1b et 2.1c). A l’inverse, deux critères différents peuvent converger vers la même partition (2.1b et 2.1d). Images extraites de [Dao et Vrain \(2024\)](#).

Définition 2.3 (Indice de Rand ajusté (ARI⁶)). Soient deux partitions $\mathcal{P}$ et $\mathcal{P}'$. L’ARI ([Hubert et Arabie, 1985](#)) entre $\mathcal{P}$ et $\mathcal{P}'$ se formule :

$$ARI(\mathcal{P}, \mathcal{P}') = \frac{2(ab - cd)}{(a + d)(d + b) + (a + c)(c + b)} \quad (2.4)$$

où a (resp. b) est le nombre de paires de points groupés ensemble (resp. séparés) dans $\mathcal{P}$ et dans $\mathcal{P}'$, et c (resp. d) est le nombre de paires de points groupés ensemble (resp. séparés) dans $\mathcal{P}$ et séparés (resp. ensemble) dans $\mathcal{P}'$.

Définition 2.4 (Information mutuelle ajustée (AMI⁷)). Soient deux partitions $\mathcal{P}$ et $\mathcal{P}'$. L’AMI ([Nguyen et al., 2009](#)) entre $\mathcal{P}$ et $\mathcal{P}'$ se formule :

$$AMI(\mathcal{P}, \mathcal{P}') = \frac{MI(\mathcal{P}, \mathcal{P}') - E(MI(\mathcal{P}, \mathcal{P}'))}{\max(H(\mathcal{P}), H(\mathcal{P}')) - E(MI(\mathcal{P}, \mathcal{P}'))} \quad (2.5)$$

où MI est la mesure d’information mutuelle entre les partitions, H l’entropie d’une partition et E l’information mutuelle attendue si les partitions sont générées aléatoirement.

6. De l’anglais *Adjusted Rand Index*.

7. De l’anglais *Adjusted Mutual Information*.

Définition 2.5 (Indice de Fowlkes-Mallows (FMI⁸)). Soient deux partitions $\mathcal{P}$ et $\mathcal{P}'$. Le FMI (Fowlkes et Mallows, 1983) est la moyenne géométrique de la précision et du rappel en prenant une partition comme référence pour l'autre :

$$FMI(\mathcal{P}, \mathcal{P}') = \sqrt{\frac{a}{a+d} * \frac{a}{a+c}} \quad (2.6)$$

où a , b et c sont les mêmes termes que pour l'ARI. Ici, ils sont appelés les *vrais positifs* (a), *faux positifs* (b) et *faux négatifs* (c).

2.2.2 Aire sous la courbe du budget de requêtes

Les mesures présentées dans la section précédentes donnent l'état d'une partition à un instant donné par rapport à une autre. Or notre travail se place dans un cadre interactif décrit en Section 3.2, où des requêtes sont faites à un utilisateur pour obtenir des informations sur le résultat voulu. On obtient plusieurs résultats successifs ; mesurer la performance de ce processus itératif revient à mesurer l'évolution d'une partition de départ au fur et à mesure des requêtes effectuées et des modifications qu'elles induisent. Lors de nos expériences, nous obtenons des mesures à chaque itération : qualité de la partition par rapport à une vérité terrain, similarité entre deux partitions successives... Afin de pouvoir mesurer le processus incrémental dans son entièreté, nous calculons l'AUBC (*Area Under the Budget Curve*, Zhan et al., 2021), c'est-à-dire l'aire sous la courbe formée par les mesures au fil des requêtes. Cette mesure traduit de façon intuitive la performance d'une méthode. Moins il faut d'itérations pour converger vers 1, plus l'aire sous la courbe sera proche de la valeur maximale. Elle permet aussi de distinguer deux parcours qui amènent à la même valeur finale : si l'un des parcours passe par une période de dégradation des performances, cela sera répercuté sur la valeur de l'AUBC. Nous calculons le Q-AUBC quand on compare une partition à chaque itération avec la vérité terrain, et le S-AUBC quand on compare une partition à celle de l'itération précédente. La figure 2.2 donne un exemple graphique de ces mesures pour l'évolution de l'ARI en fonction du nombre de requêtes. Étant donné que le S-AUBC est basée sur $t - 1$ mesures pour t itérations (on peut comparer la partition de départ à la vérité terrain, mais il n'existe pas de partition antérieure à celle-ci), nous normalisons le S-AUBC afin que toutes les métriques se trouvent dans le même intervalle $[0, 1]$.

2.3 Clustering sous contraintes

Le clustering sous contraintes consiste à intégrer des informations extérieures au jeu de données dans un algorithme de clustering (Basu et al., 2008). Envisagée à la base comme un moyen d'améliorer les performances des algorithmes classiques en réduisant l'espace de recherche et en guidant la recherche de la partition optimale, l'intégration de contraintes permet de représenter le contexte d'application de façon opérationnelle : l'utilisateur peut préciser ses attentes en les spécifiant sous forme de contraintes.

Problème (Clustering sous contraintes). Soit $\mathcal{X}$ un jeu de données et $\mathcal{C}$ un ensemble de contraintes. La tâche de **clustering sous contraintes** consiste à trouver une partition $\mathcal{P}$ de $\mathcal{X}$ telle que :

8. De l'anglais *Fowlkes-Mallows Index*.

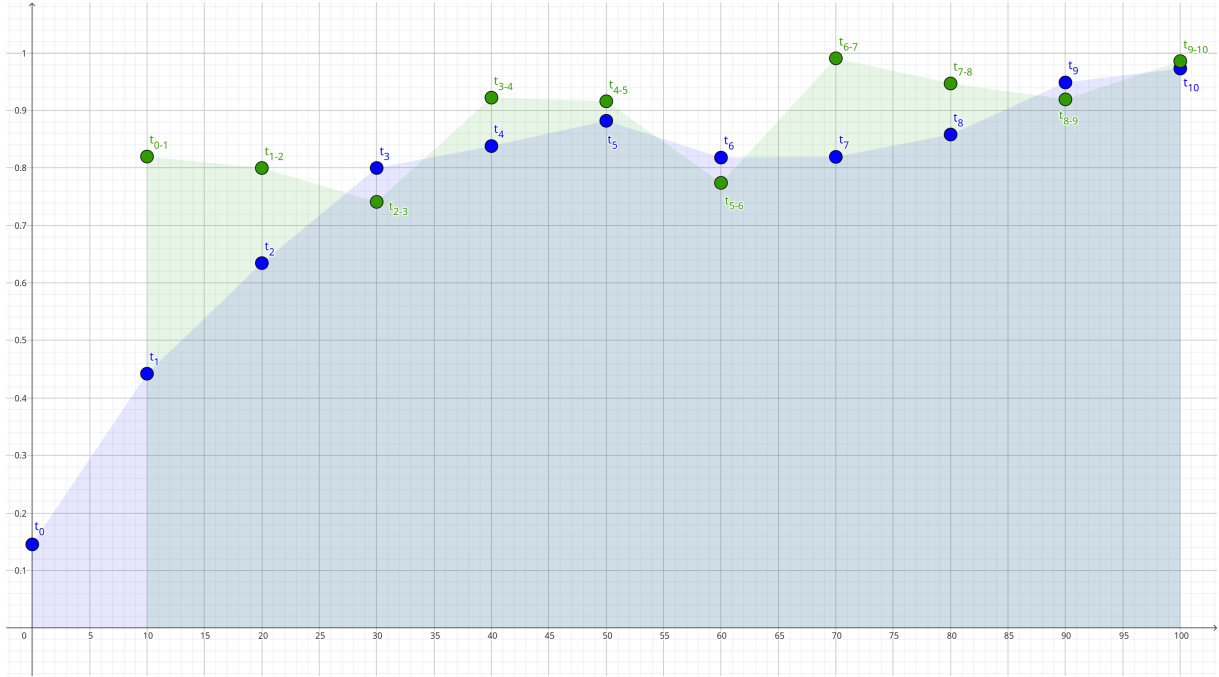


FIGURE 2.2 – Exemples illustratifs d’aires sous la courbe de budget pour l’ARI. Le Q-AUBC (ARI) est en **bleu**, le S-AUBC (ARI) en **vert**. Le nombre de requêtes est en abscisse. Dans cet exemple, chaque itération pose 10 requêtes.

1. les points dans le même cluster sont aussi similaires entre eux que possible (maximiser l’homogénéité des clusters) ;
2. les points de clusters distincts sont aussi différents que possible (maximiser l’hétérogénéité entre les clusters) ;
3. un maximum de contraintes de $\mathcal{C}$ sont satisfaites.

En clustering, les contraintes sont des restrictions sur la partition à obtenir. Elles peuvent prendre des formes très variées et s’appliquent à différentes échelles ainsi que le montre la taxonomie en figure 2.3 :

Contraintes au niveau du jeu de données : des contraintes sur la structure globale de la partition, comme par exemple le nombre de clusters (éventuellement avec des bornes), une homogénéité dans la taille des clusters, ou une limite générale à la taille des clusters.

Contraintes au niveau des clusters : des contraintes sur des propriétés de certains clusters telles que leur taille, leur diamètre (la plus grande distance entre deux points d’un même cluster), la distance aux autres clusters. Elles peuvent aussi forcer la scission d’un cluster ou la fusion de plusieurs clusters.

Contraintes au niveau des points : des contraintes impliquant des paires, des triplets ou des groupes plus larges de points indépendamment de leur appartenance à un cluster. Le type de contraintes le plus courant sont les contraintes *must-link* et *cannot-link* qui poussent deux points à être rassemblés ou séparés (Wagstaff et Cardie, 2000).

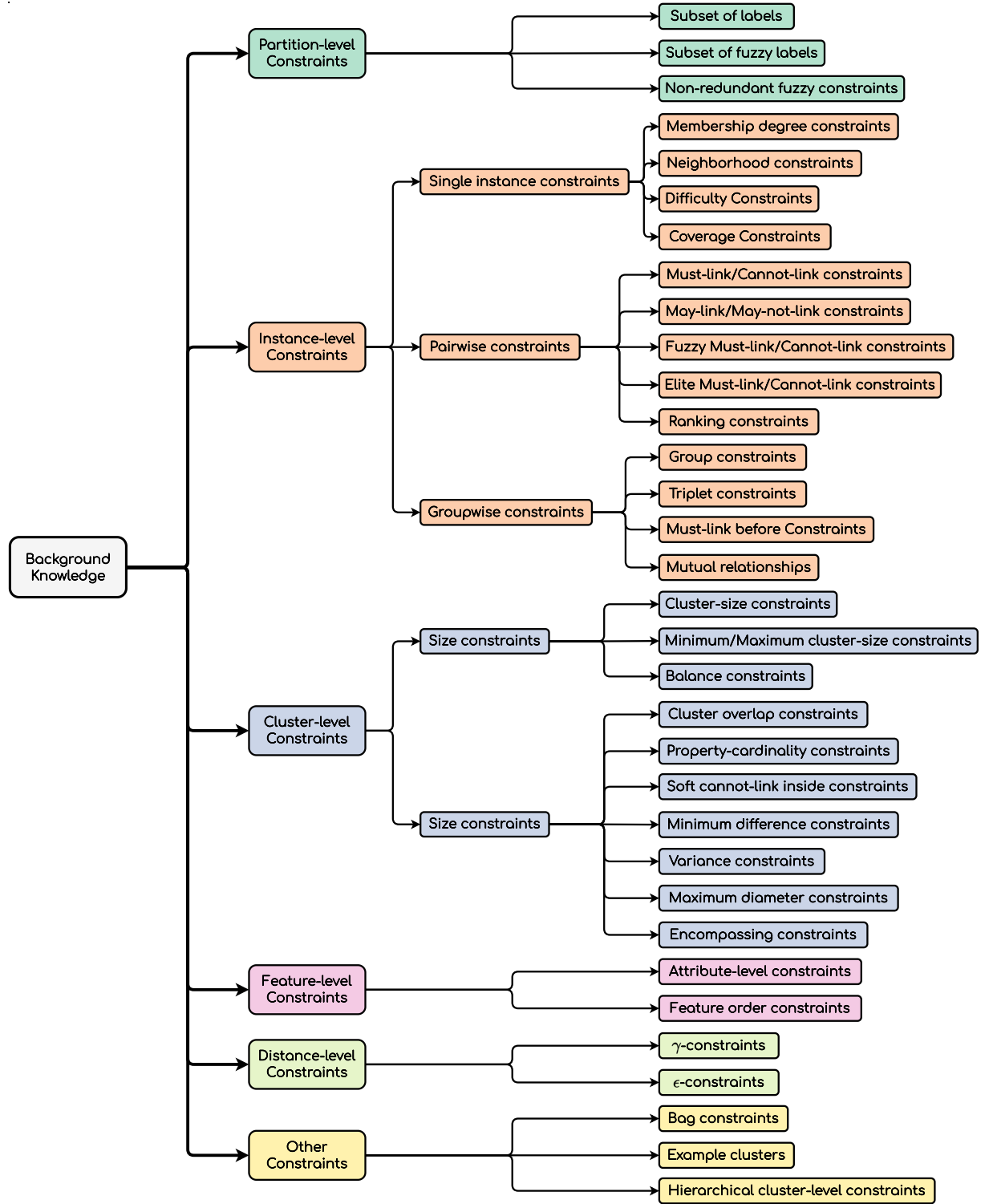


FIGURE 2.3 – Taxonomie de contraintes pour le clustering extraite de [González-Almagro et al. \(2025\)](#)

2.3.1 Contraintes utilisées

Nous donnons ici les définitions des contraintes employées dans nos contributions. Un exemple illustratif de ces contraintes est donné en Figure 2.4.

Définition 2.6 (Contrainte *must-link*). Soit $\mathcal{P} = \{C_1, \dots, C_k\}$ 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 :

$$ML(x_i, x_j) \stackrel{\text{def}}{=} \forall C_r \in \mathcal{P}, x_i \in C_r \iff x_j \in C_r, \quad (2.7)$$

Exemple 2.3. Dans l'exemple 2.2, la contrainte $ML(x_4, x_5)$ n'est pas satisfaite par la partition $[1, 2, 1, 1, 3]$. Pour la satisfaire, il existe trois possibilités : réaffecter x_4 au cluster 3, réaffecter x_5 au cluster 1 ou réaffecter x_4 et x_5 au cluster 2.

Définition 2.7 (Contrainte *cannot-link*). Soit $\mathcal{P} = \{C_1, \dots, C_k\}$ 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 :

$$CL(x_i, x_j) \stackrel{\text{def}}{=} \forall C_r \in \mathcal{P}, x_i \in C_r \implies x_j \notin C_r \quad (2.8)$$

Exemple 2.4. Dans l'exemple 2.2, la contrainte $CL(x_3, x_4)$ n'est pas satisfaite par la partition $[1, 2, 1, 1, 3]$. Pour la satisfaire, on peut réaffecter x_3 ou x_4 au cluster 2 ou 3 (quatre possibilités), ou réaffecter x_3 au cluster 2 et x_4 au cluster 3 et vice versa (deux possibilités).

La fermeture transitive sur un réseau de contraintes ML/CL permet d'obtenir des contraintes supplémentaires selon trois règles (Mallapragada et al., 2008) :

1. $ML(x_i, x_j) \wedge ML(x_j, x_k) \implies ML(x_i, x_k)$;
2. $ML(x_i, x_j) \wedge CL(x_j, x_k) \implies CL(x_i, x_k)$;
3. pour un clustering binaire ($k = 2$), $CL(x_i, x_j) \wedge CL(x_j, x_k) \implies ML(x_i, x_k)$.

Définition 2.8 (Contrainte triplet). Soit $\mathcal{P}$ une partition d'un jeu de données $\mathcal{X}$. Une contrainte triplet entre trois points x_{ref} , x_+ et x_- de $\mathcal{X}$ indique qu'un point de référence x_{ref} est plus similaire à x_+ (exemple positif) qu'à x_- (exemple négatif). Cela implique la relation suivante en fonction des affectations de $\mathcal{P}$:

$$\text{Triplet}(x_{ref}, x_+, x_-) \stackrel{\text{def}}{=} \forall C_r \in \mathcal{P}, x_{ref} \in C_r \wedge x_+ \notin C_r \implies x_- \notin C_r \quad (2.9)$$

Autrement dit, si x_{ref} et x_+ sont séparés, x_- doit également être séparé de x_{ref} (sans forcément être avec x_+) puisqu'un point plus similaire à x_{ref} en est séparé.⁹

Exemple 2.5. Dans l'exemple 2.2, la contrainte $\text{Triplet}(x_3, x_5, x_4)$ pour la partition $[1, 2, 1, 1, 3]$ implique que x_3 et x_4 devraient être séparés puisque x_5 , considéré comme plus similaire à x_3 que x_4 selon la contrainte, n'est pas dans le même cluster que x_3 .

9. Par contraposée, on peut déduire $x_{ref} \in C_r \wedge x_- \in C_r \implies x_+ \in C_r$.

Définition 2.9 (Étendue d'un ensemble de points). Soit S un ensemble de points d'un jeu de données $\mathcal{X}$. L'**étendue** (*span*) de S pour une partition $\mathcal{P}$ est :

$$Sp(S, \mathcal{P}) = \{p_x \mid x \in S\} \quad (2.10)$$

avec p_x le cluster affecté au point x dans $\mathcal{P}$.

Exemple 2.6. Dans l'exemple 2.2, l'étendue des trois premiers points $\{x_1, x_2, x_3\}$, d'affectations correspondantes $[1, 2, 1]$, est $Sp(\{x_1, x_2, x_3\}, [1, 2, 1, 1, 3]) = \{1, 2\}$.

Définition 2.10 (Contrainte d'étendue limitée). Soit $\mathcal{P}$ une partition d'un jeu de données $\mathcal{X}$ en k clusters. Une contrainte d'étendue limitée (Nghiem et al., 2022) sur $S \subseteq \mathcal{X}$ restreint l'étendue $Sp(S, \mathcal{P})$. On distingue deux catégories. Une contrainte d'étendue limitée *spécifique* contraint la plage de S par rapport à un ensemble arbitraire $C \subseteq \{1, \dots, k\}$ donné :

$$SSpan(S, C) \stackrel{\text{def}}{=} Sp(S, \mathcal{P}) \subseteq C \quad (2.11)$$

Une contrainte d'étendue limitée *générique* ne donne qu'une borne supérieure γ sur le nombre de clusters distincts dans l'étendue de S :

$$GSpan(S, \gamma) \stackrel{\text{def}}{=} |Sp(S, \mathcal{P})| \leq \gamma \quad (2.12)$$

Exemple 2.7. En reprenant l'exemple 2.6, on a $Sp(\{x_1, x_2, x_3\}) = \{1, 2\}$. L'ensemble $\{x_1, x_2, x_3\}$ respecte la contrainte $SSpan(\{x_1, x_2, x_3\}, \{1, 2\})$ et la contrainte $GSpan(\{x_1, x_2, x_3\}, 2)$.

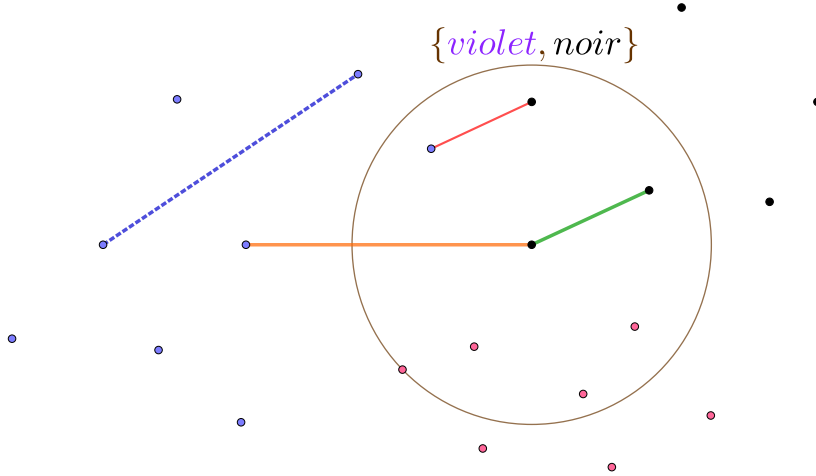


FIGURE 2.4 – Exemple de contraintes ML/CL, triplet et d'étendue limitée. La partie verte (resp. orange) de la contrainte triplet représente la relation (x_a, x_p) (resp. (x_a, x_n)) entre le point de référence et le point posé comme similaire (resp. différent). La contrainte d'étendue limitée est représentée visuellement par un cercle englobant les points concernés ainsi que la restriction associée. En l'occurrence, les points rose soumis à la contrainte doivent être réaffectés soit au cluster violet, soit au cluster noir.

La Figure 2.4 ainsi que les exemples 2.3, 2.4, 2.5 et 2.7 illustrent les contraintes utilisées dans nos travaux. Les contraintes sont généralement intégrées de deux façons :

1. Les contraintes sont traitées à part de l'optimisation de la fonction objectif : le problème est alors de trouver la partition optimale satisfaisant $\mathcal{C}$. Les contraintes sont dites rigides : aucune solution n'est retournée si la moindre contrainte reste insatisfaite.
2. Les contraintes sont intégrées au critère d'optimisation de l'algorithme, souvent sous la forme d'une pénalité. On parle alors de contraintes souples car il n'y a pas de garantie qu'elles soient toutes satisfaites. La partition résultante est un compromis entre le critère d'optimisation et la satisfaction des contraintes.

Une analyse de complexité de [Davidson et Ravi \(2007\)](#) a rapidement montré que si on peut vérifier en temps polynomial l'existence d'une partition satisfaisant des contraintes simples comme les ML, les combinaisons de contraintes variées sont susceptibles de produire des problèmes NP-complets. En particulier, tout problème impliquant des contraintes CL est NP-complet, de par les nombreuses possibilités de mettre deux points dans différents clusters.

2.4 Programmation par contraintes

La programmation par contraintes (PPC) est un paradigme de programmation apparu dans les années 90 pour résoudre des problèmes d'allocation de tâches et de planification industrielle. Il s'agit d'une évolution de la programmation logique, qui consiste à modéliser un problème sous forme d'un ensemble de formules logiques et dont est issu le langage Prolog. La PPC sacrifie la grande expressivité de la programmation logique au profit d'un système de résolution efficace basé sur la réduction de domaine des variables du problème. En tant qu'approche déclarative, ce paradigme intègre la modélisation du problème à résoudre au cœur du processus de conception de la solution au problème.

2.4.1 Problème de satisfaction de contraintes

Définition 2.11 (Contrainte). Une contrainte est un couple (V, R) avec V un ensemble de variables appelé **portée** de la contrainte, et R une relation entre les variables de V . Ces variables ont chacune un **domaine d'instanciation** fini comprenant les valeurs qu'il est possible de leur affecter. Le nombre de variables impliquées dans une contrainte est appelé **arité** de la contrainte. La satisfaction de la contrainte interdit l'affectation de certaines valeurs des variables de V : celles qui conduisent la relation à une contradiction.

Exemple 2.8. Soit deux variables A et B possédant un domaine $\{1, 2, 3\}$ et la contrainte :

$$A < B \tag{2.13}$$

La portée de la contrainte est $\{A, B\}$ et son arité est de 2. Sur les 9 combinaisons possibles initialement pour les valeurs de A et B , la contrainte réduit les possibilités : seules les paires $\{A = 1, B = 2\}$, $\{A = 2, B = 3\}$ et $\{A = 1, B = 3\}$ satisfont la contrainte.

Définition 2.12 (Problème de satisfaction de contraintes). Un problème de satisfaction de contraintes (ou CSP¹⁰) est un triplet $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ de trois ensembles finis tel que :

10. De l'anglais *constraint satisfaction problem*.

- $\mathcal{V}$ est l'ensemble des variables du problème ;
- $\mathcal{D}$ est l'ensemble des domaines des variables $\mathcal{D} = \{\mathcal{D}_x \mid x \in \mathcal{V}\}$, avec $\mathcal{D}_x$ l'ensemble des valeurs pouvant être affectées à la variable x ;
- $\mathcal{C}$ est l'ensemble des contraintes sur les variables de $\mathcal{V}$.

La résolution d'un CSP consiste à chercher une *instanciation* des variables de $\mathcal{V}$ qui satisfait les contraintes de $\mathcal{C}$.

Définition 2.13 (Instanciation). Dans un CSP, le fait d'attribuer à tout ou partie des variables du problème une valeur de leur domaine respectif produit une instanciation (ou affectation) des variables du problème. Une instanciation est **globale** (ou complète) si elle affecte une valeur à l'ensemble des variables du CSP ; dans le cas contraire, elle est **locale** (ou partielle). L'instanciation est dite vide si aucune variable n'est instanciée. Une instanciation est **consistante** si elle satisfait toutes les contraintes portant sur les variables instanciées.

Les instanciations globales consistantes sont les solutions du problème. Un algorithme de résolution d'un CSP est **robuste** si toutes les instanciations trouvées sont effectivement des solutions du CSP, et **complet** s'il parvient à trouver toutes les instanciations globales consistantes du problème pourvu qu'il ait assez de temps.

En pratique, il est souvent pertinent de chercher une unique solution au problème qui satisfait un critère spécifique. Le CSP est alors transformé en **problème d'optimisation sous contraintes**.

Définition 2.14 (Problème d'optimisation sous contraintes). Un problème d'optimisation sous contraintes (ou COP¹¹) est un CSP auquel est ajouté une **fonction objectif** f^* dépendant des variables du CSP. Là où un CSP peut admettre un grand nombre de solutions, un COP n'a pour solutions que celles maximisant ou minimisant la valeur de f^* .

La taille de l'espace de recherche est le produit des domaines de chacune des variables qui le constituent :

$$\prod_{i=1}^{|\mathcal{V}|} |\mathcal{D}_i| \quad (2.14)$$

La taille d'un espace de recherche évoluant de façon exponentielle en fonction du nombre de variables, l'approche naïve consistant à passer en revue chaque instanciation et à vérifier la satisfaction des contraintes est vite dépassée.

On décompose généralement la résolution d'un problème avec la programmation par contrainte en trois parties :

- **la modélisation du problème** : définir les variables, domaines et contraintes qui vont donner une représentation correcte du problème réel, et optimiser cette modélisation pour utiliser le moins de variables possibles, employer des contraintes globales (Section 2.4.6)...
- **la recherche des solutions** : instancier chaque variable avec une valeur de son domaine jusqu'à parvenir à une solution ou à une instanciation non consistante. L'ordre dans lequel les variables et les valeurs sont choisies a un impact significatif sur l'exploration de l'espace de recherche. Un mécanisme de *backtracking* (Section 2.4.3) évite de repartir de zéro en cas d'échec.

11. De l'anglais *constrained optimization problem*.

- **la propagation des contraintes** : après chaque instantiation de variable, enlever les valeurs des variables non instanciées qui ne peuvent plus mener à une solution.

Un solveur PPC est un algorithme qui va combiner ces trois parties pour résoudre des CSP et des COP. En voici une liste succincte :

- **Gecode** est un solveur en C++ qui permet de configurer en détail chaque composante de la stratégie de recherche ;
- **CP-SAT** est un solveur disponible dans la suite OR-Tools de Google. Il combine la résolution en PPC avec la satisfiabilité booléenne pour gagner en efficacité dans la recherche de solutions. Cependant, il est moins personnalisable que les autres ;
- **Choco** est un solveur en Java. Bien que Java soit un langage plus lent que C++, Choco arrive à trouver de bonnes solutions plus rapidement que Gecode lors des dernières compétitions MiniZinc. Il se distingue par les nombreuses contraintes globales implémentées et la facilité d'usage pour développer ses propres propagateurs.

2.4.2 Séparation et évaluation

Quand le problème à résoudre est un COP, on peut élaguer l'arbre de recherche en se servant des solutions déjà trouvées. On utilise alors un algorithme de séparation et évaluation (*branch and bound*, [Winston et Goldberg, 2004](#)) utilisé dans des paradigmes portés sur l'optimisation pure comme la programmation linéaire en nombres entiers (PLNE).

L'algorithme 2 décrit cette stratégie pour un objectif de maximisation¹². Elle consiste à comparer chaque noeud de l'arbre de recherche — correspondant soit à une instantiation partielle, soit à une solution potentielle — avec la meilleure solution trouvée.

Si le noeud correspond à une instantiation globale consistante, on a une valeur exacte de la fonction objectif f^* pour cette solution potentielle. Si cette valeur est meilleure que la meilleure valeur connue, cette instantiation devient la meilleure solution trouvée et sera utilisée dorénavant comme référence pour la suite de la recherche.

Si le noeud est une instantiation partielle, on calcule une borne supérieure donnant la meilleure valeur atteignable avec les instantiations de variables restantes. Si cette borne n'est pas meilleure que la valeur de la solution optimale actuelle, alors on peut se passer d'explorer le sous-arbre du noeud courant. Sinon, il faut explorer les sous-problèmes venant du noeud courant ; le choix des variables et valeurs à explorer en premier est effectué à l'aide d'une heuristique (Section 2.4.4).

12. Pour une minimisation, la logique est la même, il suffit d'inverser les comparaisons.

Algorithme 2 Séparation et Évaluation (Branch and Bound)

Entrée: Instance du COP $\mathcal{T} = (\mathcal{V}, \mathcal{D}, \mathcal{C}, f^*)$
Sortie: Solution optimale de $\mathcal{T}$ ou insatisfiabilité

```

1:  $S^* \leftarrow \emptyset$  ▷ Meilleure solution connue
2:  $v^* \leftarrow -\infty$ 
3:  $F \leftarrow \{\mathcal{T}\}$  ▷ File des sous-problèmes
4: tant que  $F$  n'est pas vide faire
5:   Sélectionner et retirer un sous-problème  $S$  de  $F$ 
6:   Calculer une borne optimale  $B^*(S)$  ▷ Meilleure valeur possible de  $f^*$  pour  $S$  avec
   les variables non instanciées
7:   si  $B^*(S) < v^*$  alors ▷ Évaluation
8:     continuer ▷ Élaguer cette branche
9:   si  $S$  est une instanciation globale consistante alors
10:    si  $f^*(S) > v^*$  alors
11:       $S^* \leftarrow S$ 
12:       $v^* \leftarrow f^*(S)$  ▷ Mettre à jour la meilleure solution
13:    sinon
14:      Diviser  $S$  en  $k$  sous-problèmes distincts  $S_1, S_2, \dots, S_k$  ▷ Séparation
15:      Ajouter  $S_1, S_2, \dots, S_k$  à  $F$ 
16: si  $S^* \neq \emptyset$  alors
17:   retourner  $S^*$ 
18: sinon
19:   retourner UNSAT

```

2.4.3 Backtracking

Le backtracking est un mécanisme qui permet de revenir sur une affectation passée en cas d'échec pour changer de région de l'espace de recherche. Dans le cas simple, le backtracking est déclenché si aucune valeur de la variable courante ne produit de solution du CSP. On revient alors en arrière dans l'ordre chronologique de l'affectation des variables. Des versions plus élaborées se proposent d'identifier la variable qui cause l'échec dans l'instanciation partielle pour élaguer plus efficacement l'espace de recherche.

L'algorithme 3 illustre le pseudo-code du *backtrack chronologique* (BT) de base pour résoudre les CSP. C'est une procédure d'énumération récursive en profondeur d'abord. Il commence par une affectation vide. À chaque itération, une valeur d'une variable courante est choisie 2. Le seul contrôle effectué consiste à vérifier que l'affectation résultante est satisfaite par l'ensemble des contraintes (lignes 12-17). Dans l'affirmative, l'algorithme continue avec cette nouvelle affectation partielle en ajoutant une nouvelle variable non encore affectée (ligne 9 puis 2). Sinon, il essaie une autre valeur de la variable courante (ligne 3). S'il n'y a plus de valeur inexplorée, il se trouve dans une voie sans issue. Par conséquent, il désinstancie la variable courante (il effectue un *backtrack*, ligne 10 puis potentiellement ligne 2 dû aux appels récursifs).

Algorithme 3 Backtrack simple

Entrée: CSP $\mathcal{T} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$, instantiation partielle $\mathcal{I}$ **Sortie:** booléen (instanciation satisfiable ou non)

```
1: Fonction BT( $\mathcal{T}, \mathcal{I}$ )
2:   Choisir une variable  $x_i \in \mathcal{V}$  qui n'est pas encore instanciée
3:   pour  $v_i \in \mathcal{D}(x_i)$  faire
4:      $\mathcal{U} \leftarrow \mathcal{I} \cup (x_i, v_i)$ 
5:     si ÉVALUERCONTRAINTES( $\mathcal{U}, \mathcal{C}$ ) alors
6:       si  $|\text{Variables}(\mathcal{U})| = |\mathcal{V}|$  alors
7:         retourner Vrai
8:       sinon si BT( $\mathcal{T}, \mathcal{U}$ ) alors
9:         retourner Vrai
10:    retourner Faux
11:
12: Fonction ÉVALUERCONTRAINTES( $\mathcal{U}, \mathcal{C}$ )
13:   pour  $c \in \mathcal{C}$  faire
14:     si  $\text{Portée}(c) \subseteq \text{Variables}(\mathcal{U})$  alors
15:       si  $\neg \text{Sat}(c, \mathcal{U})$  alors
16:         retourner Faux
17:   retourner Vrai
```

Dans le pire des cas, l'arbre de recherche est exploré en entier. Cet algorithme peut être amélioré en pratique par la propagation de contraintes (cf. Section 2.4.5). Ce filtrage consiste à supprimer les valeurs qui ne peuvent pas participer à une solution du CSP.

Une autre piste d'amélioration concerne le retour-arrière lui-même. Alors que le backtrack chronologique revient, en cas d'échec, sur le dernier choix effectué, d'autres mécanismes analysent les causes de l'échec soit pour choisir un meilleur point de retour (*backjumping*), soit pour identifier les instanciations partielles qui ne participent à aucune solution et ainsi éviter des explorations inutiles (*Local Change*, *Dynamic Backtracking*).

2.4.4 Heuristiques de choix

Ordonner les variables et les valeurs avant ou pendant la résolution d'un CSP est une autre voie importante d'amélioration pour l'algorithme de BT (Tsang, 1993). On distingue généralement deux types d'ordre : *statique*, dont les choix sont effectués avant la recherche et sont figés ; et *dynamique*, dont les choix dépendent du sous-arbre à explorer (par exemple, nombre de variables affectées, taille des domaines, etc.) et sont effectués à chaque nœud au cours de la recherche. Le choix de variable permet de réduire la taille de l'espace de recherche, généralement en provoquant le plus rapidement possible un échec. De nombreuses heuristiques de choix de variable ont été proposées. On distingue notamment l'heuristique *domaine minimum* favorisant les variables avec les domaines les plus petits, l'heuristique *degré maximum* qui favorise les variables dont les degrés sont les plus élevés et l'heuristique *domaine/degré* proposant de tirer parti des informations des deux heuristiques précédentes en favorisant les variables dont le ratio de la taille de leur domaine par rapport à leur degré est le plus faible. D'autre part, les heuristiques de choix de valeur cherchent à aiguiller la recherche vers les branches les plus prometteuses de

l'arbre de recherche. Parmi ces heuristiques, on retrouve l'heuristique lexicographique dont les valeurs sont rangées selon l'ordre de leur domaine, l'heuristique *look-ahead* qui classe les valeurs selon l'ordre croissant du nombre de retraits provoqués par l'affectation de celle-ci, l'heuristique basée sur les explications qui utilise les informations liées au retrait d'une valeur (c'est-à-dire aux affectations justifiant ce retrait), et l'heuristique basée sur les impacts qui ordonne les valeurs par ordre croissant d'impact. L'impact associé à une valeur mesure l'influence de son affectation sur la taille de l'espace de recherche.

2.4.5 Propagation de contraintes

L'instanciation d'une variable peut conduire à ce que certaines valeurs des variables restantes ne puissent plus amener à une solution. Il est donc inutile de les considérer dans la recherche. Le mécanisme de **propagation de contraintes** (Apt, 2003) réduit l'espace de recherche pour éviter les chemins ne pouvant pas mener à une solution, et détecter rapidement de tels chemins (par exemple si le domaine d'une variable devient vide à force de réductions).

Définition 2.15 (Support et viabilité d'une valeur). Soit deux variables A et B impliquées dans une même contrainte $C(A, B)$. On dit que (B, v_B) est le support de (A, v_A) pour C , avec $v_A \in \mathcal{D}_A$ et $v_B \in \mathcal{D}_B$, si $C(A, B)$ est satisfaite. Une valeur est viable si et seulement si elle possède au moins un support pour chacune des contraintes portant sur elle.

Exemple 2.9. Reprenons l'exemple 2.8 avec deux variables A et B de domaine $\{1, 2, 3\}$ et la contrainte $A < B$. Les valeurs 1 et 2 de A sont viables car B peut être le support de A avec les valeurs 2 ou 3. Par propagation de contraintes, la valeur 1 est enlevée du domaine de B car elle ne peut pas mener à une solution. De même, si on instancie A avec la valeur 3, la propagation de contraintes réduit le domaine de B à l'ensemble vide : le problème est insoluble.

La propagation de contraintes vise à s'assurer que chaque choix de valeurs pour les variables à instancier ait une chance de conduire à une solution. Les valeurs non viables sont supprimées du domaine : c'est ce qu'on appelle le *filtrage*. Quand un domaine a été réduit après filtrage, les valeurs d'autres variables peuvent devenir non viables car elles n'ont plus de support. Il faut donc vérifier à nouveau les variables pour *propager* l'information de la réduction de domaine. Quand on ne trouve plus de valeur à retirer et que cette réaction en chaîne s'arrête, on a alors atteint un *point fixe* du problème. Le CSP obtenu après propagation est dit **arc-cohérent**.

Définition 2.16 (Arc-cohérence). Un CSP est arc-cohérent si et seulement si aucun domaine n'est vide et si toutes les valeurs des domaines sont viables.

Cette propriété est d'autant plus importante que la complétude des algorithmes de résolution de CSP implique que l'arbre de recherche entier doit être parcouru. Il est donc crucial de pouvoir se limiter aux sections intéressantes de l'arbre.

La plupart des implémentations de solveurs de CSP intègrent des algorithmes génériques de rétablissement de l'arc-cohérence tels que AC3 (cf. Algorithme 5 (Rossi et al., 2006)), qui vont enlever les valeurs non viables de chaque domaine après chaque instanciation de

variable.

AC3 parcourt toutes les paires de variables impliquées dans une même contrainte, et vérifie la viabilité de chaque valeur (cf. Algorithme 4). Si une valeur n'est pas viable, elle est retirée et la variable à laquelle appartient cette valeur est à nouveau vérifiée. AC3 s'arrête quand les domaines de toutes les variables ne contiennent que des valeurs viables.

Algorithme 4 Fonction REVISE

Entrée: CSP $\mathcal{T} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$, variable x_i , contrainte c

Sortie: booléen (le domaine de x_i a été réduit ou non)

```

1:  $Del \leftarrow \text{Faux}$ 
2: pour toute  $v_i \in \mathcal{D}(x_i)$  faire
3:   si aucune instantiation de  $\text{Portée}(c) \mid x_i = v_i$  ne peut satisfaire  $c$  alors
4:      $\mathcal{D}(x_i) \leftarrow \mathcal{D}(x_i) \setminus \{v_i\}$   $\triangleright v_i$  n'est pas viable
5:      $Del \leftarrow \text{Vrai}$ 
6: retourner  $Del$ 

```

Algorithme 5 Algorithme AC3

Entrée: CSP $\mathcal{T} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$

Sortie: booléen (le CSP a pu être rendu arc-cohérent ou non)

```

1:  $Q \leftarrow \{(x_i, c) \mid c \in \mathcal{C} \wedge x_i \in \text{Portée}(c)\}$ 
2: tant que  $Q \neq \emptyset$  faire
3:   Prendre un couple  $(x_i, c)$  de  $Q$ 
4:    $Q \leftarrow Q \setminus \{(x_i, c)\}$ 
5:   si  $\text{REVISE}((\mathcal{V}, \mathcal{D}, \mathcal{C}), x_i, c)$  alors  $\triangleright$  Il y a eu réduction du domaine de  $x_i$ 
6:     si  $\mathcal{D}(x_i) = \emptyset$  alors
7:       retourner Faux
8:     sinon
9:        $Q \leftarrow Q \cup \{(x_j, c') \mid c' \in \mathcal{C} \wedge c' \neq c \wedge \{x_i, x_j\} \subseteq \text{Portée}(c') \wedge j \neq i\}$   $\triangleright$ 
       Propager la réduction de domaine aux autres variables de  $c$ 
10: retourner Vrai

```

La plupart des algorithmes de rétablissement d'arc-cohérence ne fonctionnent que sur des contraintes unaires ou binaires. Pour les relations plus complexes, on définit en général une contrainte globale avec un algorithme de propagation de contraintes dédié assurant l'arc-cohérence généralisée (*generalized arc-consistency* ou GAC) sur les variables concernées.

La propagation de contraintes peut aussi être intégrée au backtrack. C'est le principe du *forward checking* détaillé dans l'algorithme 6 : après affectation d'une variable, on parcourt les variables non instanciées pour supprimer les valeurs non viables en rapport avec l'affectation qui vient d'être effectuée (lignes 11 - 17).

Algorithme 6 BT avec *forward checking*

Entrée: CSP $\mathcal{T} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$, instantiation partielle $\mathcal{I}$, structure de suivi des domaines ED

Sortie: booléen (instantiation satisfiable ou non)

```

1: Fonction FC( $\mathcal{P}, \mathcal{I}, ED$ )
2:   si toutes les variables de  $\mathcal{V}$  sont instanciées alors
3:     retourner Vrai
4:   sinon
5:     Choisir une variable  $x_i \in \mathcal{V}$  qui n'est pas encore instanciée
6:     pour  $v_i \in ED(x_i)$  satisfaisant les contraintes unaires sur  $x_i$  faire
7:       non_instanciées  $\leftarrow \{x_j \in \mathcal{X} \mid x_j \text{ n'est pas encore instanciée}\} \setminus \{x_i\}$ 
8:        $D' \leftarrow ED$ 
9:        $D'(x_i) \leftarrow \{v_i\}$ 
10:      domaine_vide  $\leftarrow$  Faux
11:      tant que non_instanciées  $\neq \emptyset \wedge \neg$ domaine_vide faire
12:        Considérer une variable  $x_j$  de l'ensemble non_instanciées
13:        non_instanciées  $\leftarrow$  non_instanciées  $\setminus \{x_j\}$ 
14:         $D'(x_j) \leftarrow \{v_j \in ED(x_j) \mid \{(x_i, v_i), (x_j, v_j)\} \text{ satisfait les contraintes}\}$ 
15:        si  $D'(x_j) = \emptyset$  alors
16:          domaine_vide  $\leftarrow$  Vrai
17:        si  $\neg$ domaine_vide  $\wedge$  FC( $\mathcal{P}, \mathcal{I} \cup \{(x_i, v_i)\}, D'$ ) alors
18:          retourner Vrai
19:      retourner Faux

```

2.4.6 Contraintes globales

Lorsqu'on veut modéliser des relations complexes entre les variables d'un CSP, il est souvent nécessaire de déclarer un grand nombre de contraintes intermédiaires. Cela se révèle délétère tant au niveau de la modélisation du problème que de sa résolution. Pour contourner cet écueil, on peut définir une contrainte globale dotée d'un algorithme de propagation dédié. Cet algorithme a pour but de maintenir l'arc-cohérence généralisée de l'ensemble de variables qu'il reçoit en entrée. L'exemple le plus connu est la contrainte *AllDifferent* (Régim, 1994) pour déclarer des différences deux à deux d'un ensemble de variables. Au lieu de déclarer $\binom{k}{2}$ contraintes de différence deux à deux, le problème est plutôt vu comme la recherche de couplages couvrants dans un graphe biparti reliant des variables avec les valeurs de leur domaine. L'algorithme de filtrage de *AllDifferent* identifie les arêtes qui n'apparaissent dans aucun couplage couvrant : elles indiquent que si on donne à la variable concernée la valeur liée par l'arête, on ne pourra pas donner des valeurs différentes à chaque variable. Ces valeurs sont enlevées du domaine des variables incriminées.

Dans nos contributions, nous utilisons les contraintes globales *element*¹³ et *atmost_nvalue*¹⁴ (Bessière et al., 2006). La contrainte *element*(i, T, V) vérifie l'équivalence entre la valeur d'une variable V et une valeur dans une liste de constantes ou de variables T , donnée par

13. <https://sofdem.github.io/gccat/gccat/Celement.html> (lien visité le 14 octobre 2025)

14. https://sofdem.github.io/gccat/gccat/Catmost_nvalue.html (lien visité le 14 octobre 2025)

son indice i dans le tableau :

$$V = T[i] \quad (2.15)$$

Le propagateur de *element* maintient la cohérence en éliminant les valeurs du domaine de V qui ne sont pas dans la liste, les valeurs de l'indice i qui pointent vers des éléments incompatibles avec le domaine de V et en propageant les affectations de V et i vers les variables de T . Dans notre travail, elle sert à faire le lien entre l'affectation d'un point à un cluster, et la distance à ce cluster. La distance change quand l'affectation change, pour toujours correspondre au cluster actuel.

La contrainte $atmost_nvalue(N, T)$ (Bessière et al., 2006) limite à N le nombre de valeurs distinctes dans les affectations d'un groupe de variables T . Elle est utilisée pour encoder les contraintes d'étendue limitée génériques.

Exemple 2.10. Reprenons la partition $[1, 2, 1, 1, 3]$ de l'exemple 2.2. Supposons que des variables soient associées à chaque point, avec un domaine couvrant les affectations aux clusters connus (en l'occurrence $\{1, 2, 3\}$). La contrainte $atmost_nvalue(3, [x_1, x_2, x_5])$ est satisfaite (leurs affectations $[1, 2, 3]$ ne contient pas plus de trois valeurs distinctes), mais $atmost_nvalue(2, [x_1, x_2, x_5])$ ne l'est pas et nécessiterait de réaffecter l'un des trois points pour la satisfaire.

Le propagateur de $atmost_nvalue$ compte le nombre de valeurs distinctes dans T . En fonction des variables instanciées, il peut réduire le domaine des variables restantes en supprimant les valeurs qui mèneraient à avoir plus de N valeurs distinctes.

2.4.7 Contraintes et insatisfiabilité

Lorsqu'un CSP n'est pas satisfiable, il y a deux grandes causes possibles :

1. les domaines de certaines variables n'ont pas de valeurs capables de satisfaire certaines contraintes ;
2. il y a un conflit sur un sous-ensemble de $\mathcal{C}$: l'affectation de valeurs satisfaisant l'une d'entre elles rend les autres impossibles à satisfaire.

Dans le premier cas, il est rarement possible d'agir car les domaines des variables dépendent de la définition même du problème à résoudre. Dans le second cas, on peut identifier les causes du conflit pour trouver des contraintes erronées ou une relaxation du problème de départ en enlevant une partie des contraintes en conflit. On distingue plusieurs types d'ensembles de contraintes selon leur rôle dans le conflit ou sa résolution.

Définition 2.17 (Sous-ensemble insatisfiable minimal (MUS¹⁵)). Soit S un sous-ensemble de $\mathcal{C}$. S est un sous-ensemble minimal insatisfiable si et seulement si :

- S est insatisfiable ;
- $\forall c \in S, \{S \setminus c\}$ est satisfiable.

Définition 2.18 (Sous-ensemble satisfiable maximal (MSS¹⁶)). Soit S un sous-ensemble de $\mathcal{C}$. S est un sous-ensemble satisfiable maximal si et seulement si :

15. De l'anglais *Minimal Unsatisfiable Subset*.

16. De l'anglais *Maximal Satisfiable Subset*.

- S est satisfiable ;
- $\forall c \notin S, \{S \cup c\}$ est insatisfiable.

Définition 2.19 (Sous-ensemble de correction minimal (MCS ¹⁷)). Soit S un sous-ensemble de $\mathcal{C}$. S est un sous-ensemble de correction minimal 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.

Dans le contexte du clustering sous contraintes incrémentales, le MSS est une relaxation du problème d'origine qui répond au mieux aux retours utilisateur tandis que le MCS indique les erreurs qui ont pu être introduites par l'utilisateur, ou le fait de revenir sur une contrainte antérieure.

Définition 2.20 (Ensemble intersectant). Soit $\mathcal{E}$ une collection de sous-ensembles d'un ensemble $\mathcal{S}$. Un ensemble intersectant de $\mathcal{E}$ est un ensemble $H \subseteq \mathcal{S}$ qui contient au moins un élément de chaque ensemble de $\mathcal{E}$. H est dit *irréductible* si aucun sous-ensemble de H n'est intersectant, autrement dit si H contient un unique élément de chaque ensemble de $\mathcal{E}$.

Propriété 2.1 (Dualité entre MUS et MCS (Liffiton et Sakallah, 2008)). Soit $\mathcal{C}$ un ensemble de contraintes. Tout sous-ensemble $M \subseteq \mathcal{C}$ est un MCS si et seulement si M est un ensemble intersectant irréductible des MUS de $\mathcal{C}$. Symétriquement, tout sous-ensemble $M \subseteq \mathcal{C}$ est un MUS si et seulement si M est un ensemble intersectant irréductible des MCS de $\mathcal{C}$.

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.

2.4.8 Recherche d'ensembles insatisfiables

Le problème de la satisfiabilité d'un ensemble de contraintes est un enjeu primordial en PPC. Au-delà de savoir diagnostiquer la satisfiabilité d'un problème, de nombreux travaux ont cherché à calculer des ensembles de contraintes causant l'insatisfiabilité le cas échéant. Ils permettent également de découvrir des ensembles minimaux de contraintes à enlever pour rendre le problème satisfiable.

Les opérations élémentaires consistent à faire grossir un ensemble de départ (possiblement vide) en ajoutant des contraintes tant que l'ensemble reste satisfiable pour obtenir un MSS (cf. Algorithme 7) ou à l'inverse, d'enlever des contraintes de l'ensemble de départ insatisfiable pour obtenir un MUS (cf. Algorithme 8). Ce sont des descriptions génériques dont l'efficacité concrète dépend de l'implémentation en termes de choix d'ensemble de départ, d'ordre de parcours des contraintes ou de filtrage des candidats inutiles pour réduire l'espace de recherche.

DAA (Bailey et Stuckey, 2005) est un algorithme d'énumération des MSS et MUS d'un ensemble de contraintes. Il est *anytime* : on peut récupérer les ensembles trouvés à tout moment sans attendre la fin de l'énumération. Il calcule des MSS avec **grow** puis en déduit des MUS en calculant le MCS complémentaire puis le MUS dual (Prop. 2.1).

17. De l'anglais *Minimal Correction Subset*.

Algorithme 7 grow	Algorithme 8 shrink
Entrée: ensemble de contraintes insatisfiable $\mathcal{C}$, sous-ensemble satisfiable initial $graine \subseteq \mathcal{C}$ (par défaut $\emptyset$) Sortie: MSS de $\mathcal{C}$ 1: pour $c \in \mathcal{C} \setminus graine$ faire 2: si SAT($graine \cup \{c\}$) alors 3: $graine \leftarrow graine \cup \{c\}$ 4: retourner $graine$	Entrée: ensemble de contraintes insatisfiable $\mathcal{C}$, sous-ensemble insatisfiable initial $graine \subseteq \mathcal{C}$ (par défaut $\mathcal{C}$) Sortie: MUS de $\mathcal{C}$ 1: pour $c \in graine$ faire 2: si UNSAT($graine \setminus \{c\}$) alors 3: $graine \leftarrow graine \setminus \{c\}$ 4: retourner $graine$

FIGURE 2.5 – Méthodes de base pour trouver un MSS (à gauche) et un MUS (à droite)

CAMUS (Liffiton et Sakallah, 2008) fait aussi l'énumération de MUS en suivant deux étapes. D'abord, l'algorithme énumère tous les MCS d'un ensemble de contraintes, puis CAMUS déduit les MUS duals. La première étape est effectuée avec un solveur SAT qui recherche des MSS de taille décroissante et ajoute une clause bloquant chaque solution déjà trouvée pour tout énumérer. Il ne s'arrête pas avant d'avoir énuméré tous les MUS.

L'algorithme MARCO (Liffiton et Malik, 2013) combine l'aspect *anytime* de DAA et le principe des clauses bloquantes de CAMUS. MARCO (cf. Algorithme 10) repose sur une "carte" de l'espace de recherche représentée par une formule booléenne. Cette formule est au départ une tautologie et chaque contrainte est liée à une variable booléenne. MARCO cherche un modèle¹⁸ de la carte et agit selon la satisfiabilité de l'ensemble de contraintes lié aux variables du modèle :

- si l'ensemble est satisfiable, MARCO l'augmente avec **grow** pour obtenir un MSS puis ajoute une clause à la carte qui empêche de trouver un sous-ensemble du MSS (on sait déjà qu'ils sont satisfiables) ;
- si l'ensemble est insatisfiable, MARCO le réduit avec **shrink** pour obtenir un MUS puis ajoute une clause à la carte qui empêche de trouver un sur-ensemble du MUS (on sait déjà qu'ils sont insatisfiables).

Quand la carte n'a plus de modèle, cela signifie que toutes les régions ont été explorées et que MARCO a énuméré tous les MSS et MUS.

QUICKXPLAIN (Junker, 2004) 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 des contraintes.

Certains travaux ont cherché à expliquer pourquoi une instantiation d'un CSP est une solution, en construisant des explications pas à pas à l'aide de MUS (Bogaerts et al., 2020 ; Gamba et al., 2023). Mais à notre connaissance, il n'y a pas de travaux sur l'explication de décisions de relâchement d'une contrainte, autrement dit d'expliquer la composition d'un MCS.

18. Dans le contexte de la satisfiabilité booléenne, un modèle est une affectation des variables qui valide la formule.

Algorithme 9 DAA (*Dualize And Advance*)

Entrée: ensemble de contraintes insatisfiable $\mathcal{C}$

Sortie: MSS et MUS de $\mathcal{C}$

```

1:  $ensMCS, ensMUS, graine \leftarrow \emptyset$ 
2:  $enumère \leftarrow Vrai$ 
3: tant que  $enumère$  faire
4:    $MSS \leftarrow grow(C, graine)$  ▷ Algorithme 7
5:   yield  $MSS$ 
6:    $ensMCS \leftarrow ensMCS \cup \{C \setminus MSS\}$ 
7:    $enumère \leftarrow Faux$ 
8:   pour  $candidat \in \{ENSINTERSECT(ensMCS) \setminus ensMUS\}$  faire ▷ MUS duaux
     des MCS trouvés (Prop. 2.1)
9:     si  $SAT(candidat)$  alors
10:       $graine \leftarrow candidat$  ▷ Nouvelle graine de MSS
11:       $enumère \leftarrow Vrai$ 
12:      break
13:     sinon
14:      yield  $candidat$ 
15:       $ensMUS \leftarrow ensMUS \cup \{candidat\}$ 
16: retourner  $ensMSS, ensMUS$ 

```

Algorithme 10 MARCO (**M**apping **R**egions of **C**onstraint sets)

Entrée: ensemble de contraintes insatisfiable $\mathcal{C}$

Sortie: MUS et MSS de $\mathcal{C}$

```

1:  $Map \leftarrow \top$  ▷ Initialiser la carte comme une tautologie
2: Créer une variable booléenne  $b_i$  pour chaque contrainte  $c_i \in \mathcal{C}$ 
3: tant que  $SAT(Map)$  faire ▷ Il reste des régions non explorées
4:    $m \leftarrow GETMODEL(Map)$  ▷ Obtenir un modèle de la carte
5:    $graine \leftarrow \{c_i \in \mathcal{C} \mid m(x_i) = Vrai\}$  ▷ Contraintes vraies dans le modèle
6:   si  $SAT(graine)$  alors
7:      $MSS \leftarrow grow(graine)$  ▷ Algorithme 7
8:     yield  $MSS$ 
9:      $Map \leftarrow Map \wedge BLOQUERDESCENDANT(MSS)$  ▷ Cf. ligne 16
10:  sinon
11:     $MUS \leftarrow shrink(graine)$  ▷ Algorithme 8
12:    yield  $MUS$ 
13:     $Map \leftarrow Map \wedge BLOQUERASCENDANT(MUS)$  ▷ Cf. ligne 14
14: Fonction  $BLOQUERDESCENDANT(MSS)$  ▷ Élimine tous les sous-ensembles de  $MSS$ 
15:   retourner  $\bigvee_{c_i \notin MSS} b_i$ 
16: Fonction  $BLOQUERASCENDANT(MUS)$  ▷ Élimine tous les sur-ensembles de  $MUS$ 
17:   retourner  $\bigvee_{c_i \in MUS} \neg b_i$ 

```

Dans notre travail, nous avons cherché à gérer les conflits pouvant survenir entre les contraintes par le calcul d'un unique MCS à présenter à l'utilisateur, sans énumération. Il doit être possible d'expliquer pourquoi une contrainte se trouve dans le MCS pour que l'utilisateur n'aie pas l'impression que ses spécifications sont retirées arbitrairement. C'est l'objet du chapitre 8.

2.5 Méthodologie et données

Avant d'entrer dans l'état de l'art, nous allons terminer en introduisant des notions importantes pour la méthodologie expérimentale employée dans nos travaux. Nous détaillons également l'ensemble des jeux de données utilisées et en particulier une série temporelle d'images satellite utilisée comme cas d'étude dans nos contributions.

2.5.1 Comparaison par analyse bayésienne

Afin de comparer la performance de différents algorithmes (ou différents paramétrages du même algorithme) sur la même tâche et sur plusieurs jeux de données, nous faisons appel à la comparaison bayésienne par paires (Benavoli et al., 2017) pour évaluer deux algorithmes sur les jeux de données. Le principe de cette méthode repose sur le théorème de Bayes (Gelman et al., 2013) :

$$P(H|E) = P(E|H) \frac{P(H)}{P(E)} \quad (2.16)$$

Ici, H représente une hypothèse expérimentale et E la preuve donnée par l'expérience. Par conséquent, $P(H|E)$ est la probabilité *a posteriori* qu'une hypothèse soit vraie connaissant des résultats expérimentaux. D'après le théorème de Bayes, cela équivaut à calculer la fonction de vraisemblance $P(E|H)$ c'est-à-dire la probabilité d'obtenir les résultats expérimentaux si l'hypothèse est vraie et $P(H)$, la probabilité *a priori* que l'hypothèse soit vraie¹⁹. Ces fonctions peuvent être décrites comme des distributions de probabilité représentant l'évolution de la croyance en fonction du résultat obtenu. L'hypothèse nulle pour nos expériences est : "*Il n'y a pas de différence notable de performance entre les méthodes testées*". Autrement dit, les variations observées sont purement dues à des effets aléatoires. Cette hypothèse peut être modélisée par une distribution gaussienne centrée sur 0 (absence de différence de performance) ou équivalent. Nous utilisons un modèle hiérarchique bayésien (Corani et al., 2017) pour la fonction de vraisemblance. Ce modèle permet de distinguer les variations de chaque jeu de données et ainsi de distinguer les variations dues aux différences entre les jeux de données des effets uniquement dus aux algorithmes testés. On obtient alors une distribution *a posteriori* qui montre l'évolution de la croyance dans l'hypothèse nulle après avoir pris en compte les résultats expérimentaux. On échantillonne cette distribution 50, 000 fois avec la méthode de Monte Carlo par chaînes de Markov (Andrieu et al., 2003) pour estimer la probabilité qu'une des deux méthodes soit meilleure que l'autre ($P(A > B)$ et $P(A < B)$), ainsi que la probabilité d'être dans la région d'équivalence en pratique (ou *rope*²⁰), qui correspond à la probabilité de l'hypothèse nulle au seuil de différence r de la *rope* $P_r(H_0)$.

Ce processus d'échantillonnage permet donc de simuler la répétition du protocole expérimental et d'estimer la fiabilité de nos résultats. Nous fixons à 1% de différence de performance la région d'équivalence en pratique.

19. Dans notre cas d'application, il n'est pas nécessaire de calculer l'évidence $P(E)$ dans la mesure où elle n'impacte pas les paramètres des autres distributions de probabilité et n'est pas nécessaire pour l'échantillonnage avec une méthode de Monte Carlo. On a donc $P(H|E) = P(E|H)P(H)$.

20. De l'anglais *region of practical equivalence*.

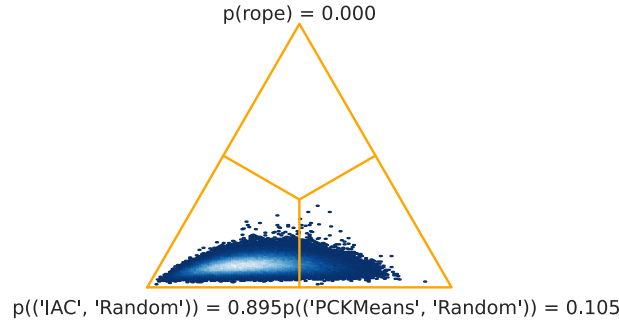


FIGURE 2.6 – Simplexe obtenu suite à l'échantillonnage d'un modèle bayésien hiérarchique. Les points sont des échantillons, placés dans le simplexe selon les probabilités $P(A > B)$, $P(B > A)$ et $P_r(H_0)$ observées pour l'échantillon. Les probabilités obtenues au final correspondent au nombre d'échantillons dans chaque zone du simplexe par rapport au nombre total d'échantillons. Dans cet exemple, on peut tirer l'observation suivante : "la méthode CoActiVe avec une sélection aléatoire de contraintes a 89.5% de chances d'être meilleure que PCKMEANS".

Nous utilisons l'implémentation en Python donnée par la librairie `baycomp`²¹ pour effectuer les tests.

Pour comparer plus de deux méthodes, nous avons réalisé $\binom{i}{2}$ duels entre les i méthodes et compté le nombre de victoires pour chaque méthode. Dans un duel entre une méthode A et une méthode B , on considère que A gagne le duel si $P(A > B) > P(B > A) + P_r(H_0)$. Les méthodes peuvent ensuite être classées selon le nombre de duels gagnés.

2.5.2 Jeux de données utilisés dans les expériences

Dans nos expériences, nous avons employé une vingtaine de jeux de données différents pour estimer la performance de nos contributions :

- 10 jeux de données d'applications réelles venant du dépôt de l'UC Irvine²² ;
- 6 jeux de données synthétiques venant de la suite FCPS présentant des problèmes classiques de clustering (Ultsch et Lötsch, 2020). Nous n'utilisons pas les jeux `tetra`, `hepta` et `twodiamonds` car ce sont des problèmes très simples, sur lesquels la plupart des algorithmes de clustering ne font aucune erreur. Une approche incrémentale a donc peu d'intérêt sur ces données ;
- 5 jeux de données synthétiques utilisés comme exemple de la librairie Python `scikit-learn`²³ ;

21. <https://baycomp.readthedocs.io/en/latest/index.html>

22. <https://archive.ics.uci.edu/ml/index.php>

23. <https://scikit-learn.org/stable/modules/clustering.html>

UCI				FCPS			
Name	(N , A , K)	Name	(N , A , K)	Name	(N , A , K)	Name	(N , A , K)
iris	(150, 4, 3)	ionosphere	(351, 34, 2)	lsun	(400, 2, 3)	chainlink	(1000, 3, 2)
wine	(178, 13, 3)	yeast	(1484, 8, 10)	target	(770, 2, 6)	wingnut	(1016, 2, 2)
sonar	(208, 60, 2)	statlog	(2310, 19, 7)	atom	(800, 3, 2)	engytime	(4096, 2, 2)
glass	(214, 9, 6)	Letters	(20000, 16, 26)				
ecoli	(336, 7, 8)	MNIST	(70000, 784, 10)				
sklearn				Mixed data			
Name	(N , A , K)	Name	(N , A , K)	Name	(N , A , K)		
varied	(500, 2, 3)	blobs	(500, 2, 3)	uci_digits	(2000, (76, 216), 10)		
anisotropic	(500, 2, 3)	circles	(500, 2, 2)	treecut	(243988, (11, 6), 3)		
halfmoons	(500, 2, 2)			treecut_sub	(1917, (11, 6), 3)		

TABLE 2.3 – Caractéristiques des jeux des données considérés, avec N le nombre de points, A le nombre d’attributs et K le nombre de clusters.

- 2 jeux de données avec deux vues : l’une utilisée pour le clustering et l’autre pour notre méthode de clustering actif (cf. Chapitre 7).
- `uci_digits` est une variante du jeu de données MNIST avec des attributs supplémentaires correspondant à des propriétés des images ;
- `treecut` est une série temporelle d’images satellites présentée en détail dans la Section 2.5.3. Un sous-ensemble nommé `treecut_sub` est obtenu par échantillonnage aléatoire de façon à obtenir un nombre équivalent des trois classes présentes.

Les caractéristiques des jeux de données sont données en Table 2.3. Les jeux de données UCI et FCPS sont extraits de la librairie Python `clustbench` (Gagolewski, 2022) de benchmarking pour le clustering. Les jeux de données *scikit-learn* sont générés en utilisant les méthodes associées du module `datasets` de *scikit-learn* avec les valeurs par défaut des différents paramètres.

2.5.3 Cas d’étude : série temporelles d’images satellite

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 2.4) (Lampert et al., 2019). 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 2.4. 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

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 2.4 – Dates des prises de vue de la série temporelle et nombre d’apparitions de zones de coupes claires à chaque période

de l’expert(e) pour dissiper la confusion est primordiale. La figure 2.7 présente la répartition en pixels des classes annotées et met en exergue la complexité de la tâche.



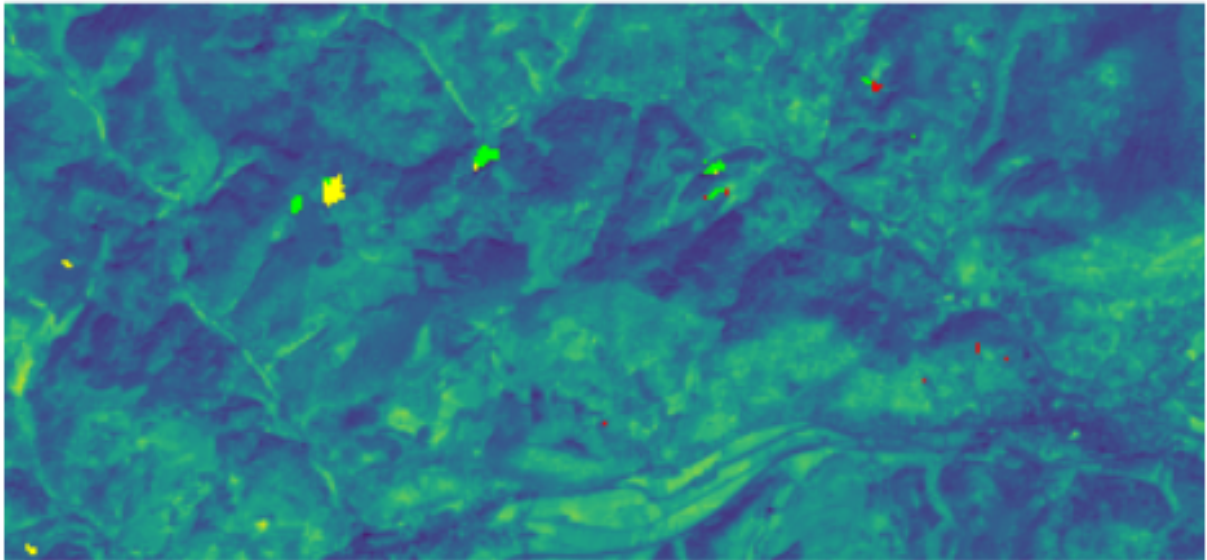
FIGURE 2.7 – Répartition des clusters sur la série d’images satellite du jeu de données *Coupes de bois*. La forêt est représentée en **vert** (156684 pixels), les zones d’activité humaine en **bleu** (86665 pixels) et les zones de coupes claires en **jaune** (639 pixels).

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.

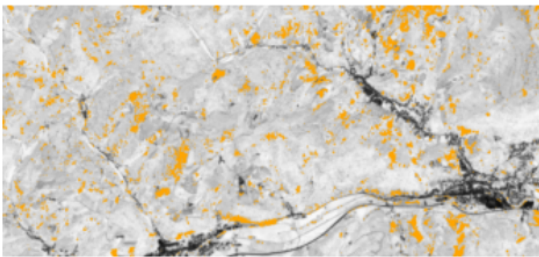
Les données de départ décrivent l’évolution temporelle du NDVI des images satellites. Une donnée x_i est donc constituée de 11 valeurs NDVI $x_{i1}, \dots, x_{i11}$. Pour tenter de trouver un cluster proche des annotations, nous avons suivi la méthode décrite dans [Lampert et al. \(2019\)](#) consistant 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 2.8b. 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 2.8a, 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.

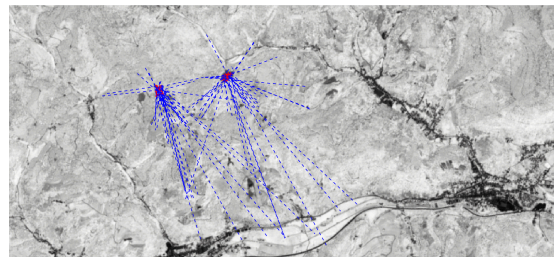
Contraintes. On dispose d'annotations données par des experts humains en télémétrie. Elles sont représentées sous la forme de 179 contraintes ML/CL (cf. Figure 2.8c), qui portent principalement sur les deux plus grandes zones de coupes de bois. Les contraintes CL relient ces zones à des zones agricoles susceptibles d'avoir des évolutions cycliques se confondant avec l'évolution des zones de coupes de bois sur l'échelle étudiée.



(a) Image d'origine.



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



(c) Annotations sous forme de contraintes ML/CL.

FIGURE 2.8 – La première image utilise les canaux d'origine, la seconde est en niveaux de gris selon la valeur NDVI. Le cluster utilisé pour l'étude de cas est affiché en orange dans la figure 2.8b. Les vrais positifs sont en vert, les faux positifs en rouge, les points de coupe absents des données considérées en jaune.

Transformation par shapelets. Dans les expériences du chapitre 7, nous avons construit une seconde vue du jeu de données en ayant recours à la transformation par *shapelets* (*shapelet transform*). Les shapelets sont des sous-séquences des séries temporelles. Dans la pratique, on s'intéresse surtout aux shapelets représentant des variations caractéristiques partagées par une partie d'un jeu de données de séries temporelles. La similarité d'une série temporelle à un shapelet est la distance entre ce shapelet et la partie de la série temporelle qui en est le plus proche. En calculant les similarités de n séries temporelles d'un jeu de données à m' shapelets, on obtient une matrice de similarité $n \times m'$ mettant à jour de nouvelles informations par rapport au jeu de données de départ, et dont l'interprétabilité dépend des shapelets utilisés pour faire la transformation.

Chapitre 3

Clustering sous contraintes

Table des matières

3.1	Topologie succincte du clustering sous contraintes	37
3.2	Clustering sous contraintes incrémentales	40
3.2.1	Contraintes utilisateur et contraintes de l'utilisateur	42
3.2.2	Algorithmes de clustering sous contraintes incrémentales	42
3.2.3	Discussion	53

3.1 Topologie succincte du clustering sous contraintes

Le développement du domaine du clustering sous contraintes peut être découpé en quatre vagues successives (Dao et Vrain, 2024).

Les premiers algorithmes de clustering sous contraintes sont apparus au début des années 2000, avec COP-COBWEB (Wagstaff et Cardie, 2000) et COP-KMEANS (Wagstaff et al., 2001). Ce sont des **adaptations de techniques classiques** qui fonctionnent avec des contraintes ML/CL¹. Par exemple, COP-KMEANS alterne comme KMEANS entre affectation de points aux centroïdes des clusters, et déplacement de ces centroïdes pour les rapprocher du centre des points qui lui sont affectés (cf. Algorithme 1); la phase d'affectation de COP-KMEANS vérifie également pour chaque point que son affectation ne contredit pas la moindre contrainte (cf. Algorithme 11). Si aucune partition satisfaisant toutes les contraintes ne peut être trouvée, COP-KMEANS ne produit pas de partition. Des adaptations similaires ont été réalisées pour le clustering hiérarchique (Davidson et Ravi, 2005) et le clustering spectral (Wang et Davidson, 2010).

Définition 3.1 (Centroïde). Soit $C = \{x_1, \dots, x_n\}$ un ensemble de points. Le centroïde de C est un point artificiel μ situé à la moyenne arithmétique des n points :

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i \quad (3.1)$$

1. Abréviation de *must-link/cannot-link*.

Définition 3.2 (Médoïde). Soit $C = \{x_1, \dots, x_n\}$ un ensemble de points. Le médoïde de C est un point $m \in C$ de distance minimale au reste des points :

$$m = \arg \min_{x \in C} \sum_{i=1}^n d(x, x_i) \quad (3.2)$$

Algorithme 11 VIOLATE-CONSTRAINTS

Entrée: Point $x_i \in \mathcal{X}$, affectation p_i de x_i dans $\mathcal{P}$, ensemble de contraintes *must-link* $\mathcal{C}_{ML}$, ensemble de contraintes *cannot-link* $\mathcal{C}_{CL}$ tel que $\mathcal{C}_{ML} \cap \mathcal{C}_{CL} = \emptyset$

Sortie: booléen indiquant si p_i est une affectation impossible d'après $\mathcal{C}_{ML}$ et $\mathcal{C}_{CL}$

- 1: **pour** chaque contrainte $ML(x_i, x_j) \in \mathcal{C}_{ML}$ avec $j \neq i$ **faire**
 - 2: **si** $p_j \neq p_i$ **alors**
 - 3: **retourner** Vrai ▷ p_i et p_j sont des affectations différentes
 - 4: **pour** chaque contrainte $CL(x_i, x_j) \in \mathcal{C}_{CL}$ avec $j \neq i$ **faire**
 - 5: **si** $p_j = p_i$ **alors**
 - 6: **retourner** Vrai ▷ p_i et p_j sont des affectations identiques
 - 7: **retourner** Faux ▷ Toutes les contraintes sont satisfaites si x_i est affecté à p_i
-

Par la suite, des adaptations utilisant les contraintes souples émergent pour éviter les échecs. C'est le cas de PCKMEANS (Basu et al., 2004) qui intègre des termes de pénalisation des contraintes non satisfaites dans la fonction objectif de KMEANS :

$$\sum_{x_i \in \mathcal{X}} \|x_i - \mu_i\|^2 + \sum_{ML(x_i, x_j) \in \mathcal{C}_{ML}} w_{i,j} \mathbb{I}(p_i \neq p_j) + \sum_{CL(x_i, x_j) \in \mathcal{C}_{CL}} \overline{w}_{i,j} \mathbb{I}(p_i = p_j) \quad (3.3)$$

Le premier terme de l'équation 3.3 est l'objectif de clustering : la somme des carrés des distances euclidiennes de chaque point x_i au centroïde μ_i de son cluster. Les deux autres termes ajoutent une pénalité individuelle de non-satisfaction des contraintes ML ($w_{i,j}$) et CL ($\overline{w}_{i,j}$) respectivement. PCKMEANS cherche donc un compromis entre une partition optimale au sens de KMEANS et la satisfaction des contraintes. Il donne toujours un résultat au détriment de la garantie de satisfaire l'ensemble des contraintes. En outre, PCKMEANS initialise les centroïdes en se basant sur des voisinages provenant des contraintes (cf. Section 4.3 pour une explication détaillée des voisinages).

La deuxième vague commence aux alentours de 2005 et se concentre sur le concept du *metric learning* : l'idée est d'exploiter les contraintes à disposition pour modifier l'espace de clustering afin de le rendre propice à la découverte de clusters denses et bien séparés. Dans le cas des contraintes ML/CL par exemple, le *metric learning* consiste à rapprocher les points liés en ML et éloigner ceux liés en CL (Klein et al., 2002). On obtient un espace qui encode directement les informations des contraintes. L'algorithme sera ensuite étendu dans MPC-KMEANS en intégrant un mécanisme de *metric learning* basé sur les contraintes ML/CL (Bilenko et al., 2004). Concrètement, le critère d'optimisation est :

$$\sum_{x_i \in \mathcal{X}} \left(\|x_i - \mu_i\|_{A_{p_i}}^2 - \log(\det(\mathbf{A}_{p_i})) \right) + \sum_{ML(x_i, x_j) \in \mathcal{C}_{ML}} w_{ij} f_M(x_i, x_j) \mathbb{I}[p_i \neq p_j] + \sum_{CL(x_i, x_j) \in \mathcal{C}_{CL}} \overline{w}_{ij} f_C(x_i, x_j) \mathbb{I}[p_i = p_j] \quad (3.4)$$

Par rapport au critère de PCKMEANS (Éq. 3.3), le premier terme de l'équation 3.4 est maintenant calculé dans un espace euclidien paramétré par un ensemble de matrices $\mathbf{A}$ qui permet à chaque cluster d'avoir sa métrique locale. A chaque itération, les matrices $\mathbf{A}_C$ de chaque cluster sont mises à jour en fonction du nouveau centroïde et des contraintes portant sur des points du cluster. Dans les termes de pénalisation des contraintes insatisfaites, deux fonctions sont ajoutées. f_M (Éq. 3.5) pénalise encore plus les contraintes ML entre des points éloignés en faisant la différence des distances entre les points de la contrainte dans les métriques des clusters affectés :

$$f_M(ML(x_i, x_j)) = \frac{1}{2} \|x_i - x_j\|_{\mathbf{A}_{p_i}}^2 + \frac{1}{2} \|x_i - x_j\|_{\mathbf{A}_{p_j}}^2 \quad (3.5)$$

Si les deux points sont dans le même cluster, la différence est nulle.

f_C (Éq. 3.6) pénalise plus fortement les contraintes CL entre des points proches en comparant la distance entre les points contraints à la plus grande distance entre deux points dans la métrique locale du cluster ($x'_{p_i} - x''_{p_i}$) :

$$f_C(CL(x_i, x_j)) = \|x'_{p_i} - x''_{p_i}\|_{\mathbf{A}_{p_i}}^2 - \|x_i - x_j\|_{\mathbf{A}_{p_i}}^2 \quad (3.6)$$

De cette façon, MPC-KMEANS intègre mieux les informations des contraintes ML/CL au prix d'un coût en temps plus élevé dû au calcul des espaces locaux. De façon similaire, HMRF-KMEANS (Basu et al., 2006) utilise le cadre bayésien des modèles de Markov cachés pour calculer des espaces paramétrés avec plusieurs types de mesures (distance euclidienne, similarité cosinus, divergence de Kullback-Leibler).

La troisième vague est celle des **approches déclaratives**, à partir du début des années 2010. Elle fait partie d'un effort plus global pour relier la fouille de données à des paradigmes qui permettent de décrire facilement des problèmes spécifiques sans écrire d'algorithme dédié. Ils offrent notamment la capacité de combiner facilement plusieurs types de contraintes et un plus grand contrôle sur leur satisfaction. Des travaux ont été menés en ce sens en s'appuyant sur la satisfiabilité booléenne (Davidson et al., 2010), la programmation linéaire en nombres entiers (Quali et al., 2016) ou la programmation par contraintes (Dao et al., 2017).

La vague la plus récente est celle de l'**apprentissage profond** (*deep learning*). Comme la plupart des domaines en IA, le clustering a tenté d'exploiter la puissance des réseaux de neurones afin d'obtenir de meilleures performances. Dans le contexte particulier du clustering sous contraintes, le défi est l'intégration des connaissances utilisateur dans le réseau pour orienter le résultat. Cette intégration passe souvent par la mise au point de fonctions de perte pour les différents types de contraintes (Zhang et al., 2021) ou pour traiter des formules en logique propositionnelle (Nghiem et al., 2022)².

2. Un état de l'art détaillé du clustering sous contraintes est proposé dans González-Almagro et al. (2025) pour plus de références.

3.2 Clustering sous contraintes incrémentales

Prêcher le faux pour savoir le vrai
(locution verbale)

Technique de manipulation consistant à affirmer délibérément un mensonge pour obliger son interlocuteur à rétablir la vérité, afin de connaître cette dernière.

La plupart des méthodes de clustering sous contraintes calcule une partition d'un jeu de données à l'aide d'un ensemble de contraintes prédéfini, supposé disponible avant le clustering. Or cette façon de construire les partitions pose implicitement deux hypothèses :

1. l'ensemble de contraintes fourni est suffisant pour satisfaire les attentes concernant la partition désirée ;
2. toutes les contraintes utiles peuvent être directement produites avant le calcul de la partition.

Ces hypothèses sont rarement vérifiées en pratique. L'utilisateur va avoir tendance à formuler des contraintes qui seraient de toute façon satisfaites par la plupart des algorithmes de clustering classiques (Gançarski et al., 2020). Il ou elle peut soudainement avoir un déclic permettant d'obtenir des contraintes qui paraissaient inutiles auparavant. Plus simplement, attendre d'un humain une vision claire du résultat désiré n'est pas réaliste : soit sa spécification est suffisamment explicite pour que des méthodes plus simples suffisent à le calculer, soit elle ne l'est pas et il faut pousser l'utilisateur à clarifier ses attentes par le questionnement. Comme le résument Cohn et al. (2001), un bon algorithme de clustering sous contraintes « [...] does not assume that the user is conscious of what they think defines a good clustering but that, as with art, they will "know it when they see it". ».

Le problème que nous appelons sous le terme de *clustering sous contraintes incrémentales*³ consiste à chercher une partition $\mathcal{P}$ d'un jeu de données $\mathcal{X}$ considérée comme satisfaisante d'après une fonction f inconnue représentant les attentes de l'utilisateur. Pour trouver $\mathcal{P}$, il faut approximer f avec des lots de contraintes qui donnent une partie des attentes et avec lesquels on peut modifier une partition de départ pour améliorer $f(\mathcal{P})$. Après chaque lot de contraintes, on a d'un côté une meilleure approximation de f et de l'autre une partition qui satisfait les informations connues sur f et qui est donc plus susceptible d'être considérée comme satisfaisante.

Les algorithmes de clustering sous contraintes incrémentales ont deux critères d'arrêt : la satisfaction de l'utilisateur et l'épuisement de l'espace de contraintes. Comme le montre la table 3.1, même si la taille de l'espace de recherche des contraintes évolue de façon polynomiale⁴, ce dernier devient vite trop grand par rapport au nombre de requêtes que l'on peut raisonnablement espérer poser à l'utilisateur (de quelques dizaines à quelques milliers

3. En anglais, nous employons le terme de *constraint-incremental clustering* pour expliciter le fait que c'est bien l'ensemble des contraintes qui grandit au fur et à mesure et pas le jeu de données, ce que l'accord en genre en français suffit à exprimer.

4. L'évolution d'un coefficient binomial $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ en fonction de n pour k fixé est une fonction polynomiale d'ordre k .

Algorithme 12 Clustering sous contraintes incrémentales

Entrée: jeu de données $\mathcal{X}$, ensemble de contraintes $\mathcal{C}$ potentiellement vide.

Sortie: partition finale $\mathcal{P}_f$

- 1: générer une partition initiale $\mathcal{P}_0$
 - 2: $t \leftarrow 0$
 - 3: **tant que** la partition n'est pas satisfaisante pour l'utilisateur et qu'il existe des contraintes à ajouter **faire**
 - 4: demander à l'utilisateur et/ou générer automatiquement un lot de contraintes L_{t+1} basé sur $\mathcal{X}$ et $\mathcal{P}_t$
 - 5: $\mathcal{C} \leftarrow \mathcal{C} \cup L_{t+1}$
 - 6: calculer la partition $\mathcal{P}_{t+1}$ avec L_{t+1} et tout ou partie de $\mathcal{C}$
 - 7: $t \leftarrow t + 1$
 - 8: **retourner** $\mathcal{P}_f = \mathcal{P}_t$
-

dans le meilleur des cas). Hormis des applications très simples, c'est donc l'utilisateur qui décide de l'intérêt de faire une nouvelle itération.

L'algorithme 12 donne une procédure générale pour résoudre ce problème. On calcule une première partition du jeu de données en entrée avec une méthode de clustering classique, que l'on présente à l'utilisateur. Il ou elle apporte des retours sous la forme d'un lot de contraintes si la partition n'est pas satisfaisante. La partition est modifiée pour satisfaire le lot de contraintes, ainsi que tout ou partie des contraintes antérieures. On présente à nouveau la partition à l'utilisateur, répétant les étapes précédentes jusqu'à valider l'un des critères d'arrêt.

Le choix des contraintes utilisées pour la modification est un enjeu important. Il est possible de n'utiliser que le lot de contraintes obtenu à chaque itération — cela revient à considérer que les contraintes précédentes sont implicitement encodées dans la partition — mais cela ne permet pas de détecter des contradictions entre différentes itérations : on peut dire que deux points doivent être groupés ensemble au début du processus, puis dire l'inverse quelques itérations plus tard. D'autre part, garder toutes les contraintes passées pour chaque modification n'est pas forcément souhaitable non plus. Outre la complexité potentiellement accrue du problème à résoudre à cause de l'augmentation du nombre de variables, cela signifie aussi qu'on considère toutes les contraintes comme équivalentes. Pourtant, dans le cas où l'utilisateur change d'objectif au cours du temps, les premières contraintes devraient avoir moins d'importance que les dernières puisqu'elles décrivent une partition dont il ou elle s'est détourné.

Contrainte	Nombre de possibilités
ML/CL	$\binom{n}{2} = \frac{n(n-1)}{2}$ (nombre de combinaisons de 2 points)
Triplet	$n(n-1)(n-2)$ (nombre de permutations de 3 points)
Étendue limitée	$\sum_{i=1}^n \sum_{c=1}^k \binom{n}{i} \binom{k}{c}$

TABLE 3.1 – Taille de l'espace de recherche pour chaque type de contrainte considérée en Section 2.3.1).

3.2.1 Contraintes utilisateur et contraintes de l'utilisateur

L'avantage majeur du contexte incrémental est l'interaction forte avec l'utilisateur, qui peut significativement améliorer les résultats du processus. Cependant le simple fait d'avoir un humain dans la boucle amène ses propres contraintes principalement dues à la difficulté de le maintenir engagé dans l'interaction.

Les principales contraintes liées à l'utilisateur sont les suivantes :

- **Le temps d'attente entre les interactions.** Si l'utilisateur ne voit pas de progression dans le processus, il peut vite perdre tout intérêt pour le système.
- **La lisibilité.** L'utilisateur doit pouvoir comprendre ce qui se passe, au moins de façon générale, et surtout pouvoir estimer l'impact de ses contributions sur le système.
- **La gestion de la charge cognitive.** Il doit être facile de fournir des retours au système. Si ce n'est pas le cas, la difficulté doit être justifiée par l'efficacité de la tâche.

Le temps d'attente impose que chaque étape soit aussi rapide que possible ; en termes de conception, il s'agit aussi d'identifier les goulots d'étranglement éventuels et de les contourner. La contrainte de lisibilité peut être satisfaite en faisant en sorte de retracer une évolution cohérente entre le premier et le dernier résultat, ce qui signifie que deux résultats successifs doivent rester similaires. Enfin, la façon la plus évidente de diminuer la charge cognitive est d'utiliser un système actif qui va proposer des contraintes à l'utilisateur si ce dernier n'a pas de contraintes à intégrer. Il ou elle se trouve alors dans une situation de validation des choix du système demandant moins d'efforts que de produire les contraintes directement.

3.2.2 Algorithmes de clustering sous contraintes incrémentales

Plusieurs termes sont utilisés dans la littérature scientifique pour désigner ce que nous appelons le clustering sous contraintes incrémentales : *incremental constrained clustering*, *cluster refinement*, *anytime constrained clustering*, *assisted clustering*... Dans l'état de l'art sur le clustering interactif de Bae et al. (2020), les méthodes correspondant à notre définition et aux travaux utilisant ces termes sont classés comme de la correction de clusters existants par l'ajout de contraintes.

La première référence à la prise en compte du retour utilisateur dans le clustering est dans Cohn et al. (2001) qui propose une procédure simple d'itération sur le retour utilisateur jusqu'à satisfaction pour le clustering de documents textuels. L'utilisateur donne des contraintes ML/CL en se basant sur une partition existante. Les contraintes servent à recalculer une mesure de similarité basée sur la divergence de Kullback-Leibler. Plus tard, Basu et al. (2010) proposent le « clustering assisté » (*assisted clustering*) dont le principe est de laisser l'utilisateur former des clusters de façon sémantique via une interface graphique, puis de faire des suggestions à l'aide de *metric learning* et de classification : proposer les objets les plus pertinents pour un cluster donné, ou le cluster le plus approprié pour un nouvel objet. On retrouve l'idée de faire des modifications d'une partition déjà existante ; ici la partition est également formée par l'utilisateur au lieu d'être calculée automatiquement. Lai et al. (2014) décrivent un algorithme très similaire à l'Algorithme 12 avec une initialisation par le calcul d'une première partition, puis l'alternance de retours

utilisateurs et de modification de clustering. Cependant toute la méthode repose sur BIRCH (Zhang et al., 1996) – une variante de DBSCAN – et ses structures de données. C’est donc une méthode liée à un algorithme spécifique.

Davidson et al. (2007) étudient le problème d’ajouter (ou enlever) une contrainte ML/CL à un ensemble de contraintes satisfaites par une partition. Ils démontrent que ce problème est dans NP pour sa formulation générale, mais qu’il est facile à résoudre si le graphe des contraintes ML/CL est q -inductif, c’est-à-dire qu’il est possible d’ordonner les sommets de telle façon que tout sommet choisi est voisin avec au plus q sommets se trouvant après lui dans l’ordre considéré. Cela signifie que l’heuristique consistant à affecter un cluster aux points les plus contraints en premier permet de trouver efficacement une solution. Ils présentent un algorithme réalisant la modification de partition dans ces conditions, traitant les contraintes au cas par cas et effectuant les modifications nécessaires en fonction du type de contrainte et de l’affectation des points contraints. Prabakara Raj et Ravindran (2013) décrivent 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. Mitchell (2016), 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. Awasthi et al. (2015) introduisent la définition d’algorithme local pour désigner une procédure de modification de partition qui n’agit que sur les points sur lesquels l’utilisateur a posé des requêtes. Ils proposent des techniques de fusion et de scission de clusters sur requête de l’utilisateur sous une hypothèse dite de stabilité de la partition cible (dans le résultat final, tout sous-ensemble d’un cluster doit avoir une meilleure similarité au reste du cluster qu’à tout autre ensemble hors du cluster). Certaines méthodes de clustering actif comme Xiong et al. (2016) peuvent être vues comme du clustering sous contraintes incrémentales parce qu’elles reposent sur l’alternance entre obtention de contraintes et clustering. Nous verrons cela plus en détail dans le chapitre suivant (Section 4.2).

Après ce tour d’horizon rapide, nous allons présenter en détail quelques méthodes qui nous semblent les plus représentatives du domaine.

3.2.2.1 Construction incrémentale de partition hiérarchique

Une famille d’algorithmes se base sur le principe du clustering hiérarchique en se guidant par des requêtes entre les clusters pour réaliser les fusions ou scissions. Ainsi COBRA (Van Craenendonck et al., 2017, cf. Algorithme 13) commence par calculer un surclustering avec KMEANS de façon à obtenir un grand nombre N_S de clusters (ligne 1)

appelés *super-instances* et représentés par leur médoïde (le point du jeu de données le plus proche du centre géométrique de la super-instance). COBRA réalise l'agglomération en sélectionnant les clusters les plus proches (ligne 6) et en demandant s'ils doivent être regroupés ensemble (ligne 10). La fusion des médoïdes implique la fusion des points représentés dans les super-instances concernées. Si la fusion est réalisée, les distances sont recalculées avec le nouveau cluster pour trouver la prochaine requête de fusion. COBRA s'arrête lorsqu'aucune fusion entre les paires existantes n'est acceptée.

Algorithme 13 COBRA (*CO*nstraint-Based Repeated Aggregation) (Van Craenendonck et al., 2017)

Entrée: jeu de données $\mathcal{X}$, nombre de super-instances N_S

Sortie: partition $\mathcal{P}$

```

1:  $\mathcal{P} \leftarrow \text{KMEANS}(X, N_S)$  ▷ Surclustering
2:  $C_i \leftarrow \{s_i\} \forall i \in \{1, \dots, N_S\}$  ▷ Clusters représentés par leur médoïde
3:  $\mathcal{C}_{CL} \leftarrow \emptyset$ 
4:  $\text{fusion} \leftarrow \text{Vrai}$ 
5: tant que  $\text{fusion}$  faire
6:    $\text{paires} \leftarrow \{(C_1, C_2) \mid \nexists x_i \in C_1, x_j \in C_2 \mid CL(x_i, x_j) \in \mathcal{C}_{CL}\}$  triées par distance  $C_1 - C_2$ 
7:    $\text{fusion} \leftarrow \text{Faux}$ 
8:   pour chaque  $(C_1, C_2) \in \text{paires}$  faire ▷ Clusters pouvant être fusionnés
9:      $(s_a, s_b) \leftarrow \arg \min_{s_1 \in C_1, s_2 \in C_2} \|s_1 - s_2\|_2$  ▷ Super-instances à la frontière entre  $C_1$  et  $C_2$ 
10:    si  $\text{ML}(s_a, s_b)$  alors
11:       $C_1 \leftarrow C_1 \cup C_2$  ▷ Fusion de  $C_1$  et  $C_2$ 
12:       $\mathcal{P} \leftarrow \mathcal{P} \setminus \{C_2\}$ 
13:       $\text{fusion} \leftarrow \text{Vrai}$ 
14:      break
15:    sinon
16:       $\mathcal{C}_{CL} \leftarrow \mathcal{C}_{CL} \cup \{CL(s_a, s_b)\}$ 
17: retourner  $\mathcal{P}$ 

```

Les auteurs bornent le nombre de requêtes faites durant l'exécution de COBRA entre $N_S - k + \binom{k}{2}$ (cas optimiste où on fusionne toujours la première paire de clusters proposée jusqu'à atteindre les k clusters de la partition finale⁵) et $\binom{N_S}{2}$ (cas pessimiste où on passe en revue toutes les paires issues du surclustering initial). Dans le pire cas, le clustering initial en petits groupes est la partition attendue ; aucune fusion n'est nécessaire. Plus le nombre de clusters générés est proche du nombre contenu dans la partition finale, plus le nombre réel de requêtes se rapproche de la borne inférieure.

COBRAS (Craenendonck et al., 2018, cf. Algorithme 14) est une amélioration de COBRA qui réduit le nombre de requêtes nécessaires quand le surclustering contient beaucoup plus de clusters que la partition recherchée. Cette abondance de clusters est utile pour éviter que les super-instances contiennent des points de différents clusters de la partition recherchée, mais augmente sensiblement le nombre de requêtes. COBRAS inverse

5. Rappelons qu'en pratique, k n'est *a priori* pas connu, ce qui rend difficile l'estimation concrète du nombre de requêtes.

le raisonnement en partant d'un cluster englobant l'ensemble des données. COBRAS divise la super-instance contenant le plus de points en plus petits clusters avec KMEANS, puis utilise COBRA pour effectuer les éventuels regroupements. La granularité de la division, autrement dit le nombre de clusters, est déterminée en opérant une suite de scissions binaires du cluster à diviser et en demandant à l'utilisateur à chaque étape si les clusters obtenus doivent bien être séparés (ce qui donne une contrainte CL entre les médoïdes des deux clusters), jusqu'à obtention d'un ML signifiant que les deux clusters ne doivent pas être séparés et donc que la granularité est devenue trop petite (cf. étapes A et D de la figure 3.1 et ligne 12 de l'algorithme 14). Le facteur de granularité correspond au nombre de noeuds du dernier niveau de l'arbre binaire sans ML, soit 2^d avec d la profondeur de ce niveau. Cette technique permet d'utiliser COBRA de manière plus précise en économisant les requêtes et facilite la création de clusters de tailles inégales.

Algorithme 14 COBRAS (*CO*nstraint-Based *Re*peated *Ag*gregation and *S*plitting) (Craenendonck et al., 2018)

Entrée: jeu de données $\mathcal{X}$, budget de requêtes b

Sortie: partition $\mathcal{P}$

```

1:  $\mathcal{C}_{ML} \leftarrow \emptyset, \mathcal{C}_{CL} \leftarrow \emptyset$ 
2:  $S \leftarrow \{\mathcal{X}\}, C \leftarrow \{S\}, \mathcal{P} \leftarrow \{C\}$ 
3: tant que  $|\mathcal{C}_{ML}| + |\mathcal{C}_{CL}| < b$  faire
4:    $S_{div}, C_{orig} \leftarrow \arg \max_{S \in C, C \in \mathcal{P}} |S|$  ▷ Plus grande super-instance
5:    $d, \mathcal{C}_{ML}, \mathcal{C}_{CL} \leftarrow \text{GRANULARITÉ}(S_{div})$  ▷ Cf. ligne 12
6:    $S_1^{new}, \dots, S_d^{new} \leftarrow \text{KMEANS}(S_{div}, d)$  ▷ Scission selon la granularité choisie
7:    $C_{orig} \leftarrow C_{orig} \setminus \{S_{div}\}$ 
8:    $\mathcal{P} \leftarrow \mathcal{P} \cup \{\{S_1^{new}\}, \dots, \{S_d^{new}\}\}$  ▷ Nouvelles super-instances comme clusters
9:    $\mathcal{P}, \mathcal{C}_{ML}, \mathcal{C}_{CL} \leftarrow \text{COBRA}(\mathcal{P}, \mathcal{C}_{ML}, \mathcal{C}_{CL})$  ▷ Fusion éventuelle des nouveaux clusters
10: retourner  $\mathcal{P}$ 
11:
12: Fonction GRANULARITÉ( $S$ )
13:    $d \leftarrow 0, \mathcal{C}_{ML} \leftarrow \emptyset, \mathcal{C}_{CL} \leftarrow \emptyset$ 
14:   tant que  $\mathcal{C}_{ML} = \emptyset$  faire
15:      $S_1, S_2 \leftarrow \text{KMEANS}(S, 2)$  ▷ Scission de la super-instance en deux
16:     si ML(MÉDOÏDE( $S_1$ ), MÉDOÏDE( $S_2$ )) alors
17:        $\mathcal{C}_{ML} \leftarrow \mathcal{C}_{ML} \cup \text{ML}(\text{MÉDOÏDE}(S_1), \text{MÉDOÏDE}(S_2))$ 
18:        $d \leftarrow \max(d, 1)$  ▷ Niveau adéquat de granularité
19:       retourner  $2^d, \mathcal{C}_{ML}, \mathcal{C}_{CL}$ 
20:     sinon ▷ Granularité plus fine nécessaire
21:        $\mathcal{C}_{CL} \leftarrow \mathcal{C}_{CL} \cup \text{CL}(\text{MÉDOÏDE}(S_1), \text{MÉDOÏDE}(S_2))$ 
22:        $S \leftarrow S_1$  ou  $S_2$  choisi aléatoirement
23:        $d \leftarrow d + 1$ 

```

Cette approche a été adaptée aux séries temporelles sous le nom de COBRAS^{TS} (Van Craenendonck et al., 2018) avec une interface utilisateur⁶. L'adaptation consiste à utiliser des distances et des algorithmes développés pour les séries temporelles pour réaliser

6. <https://dtai.cs.kuleuven.be/software/cobras>, lien visité le 2 juillet 2025.

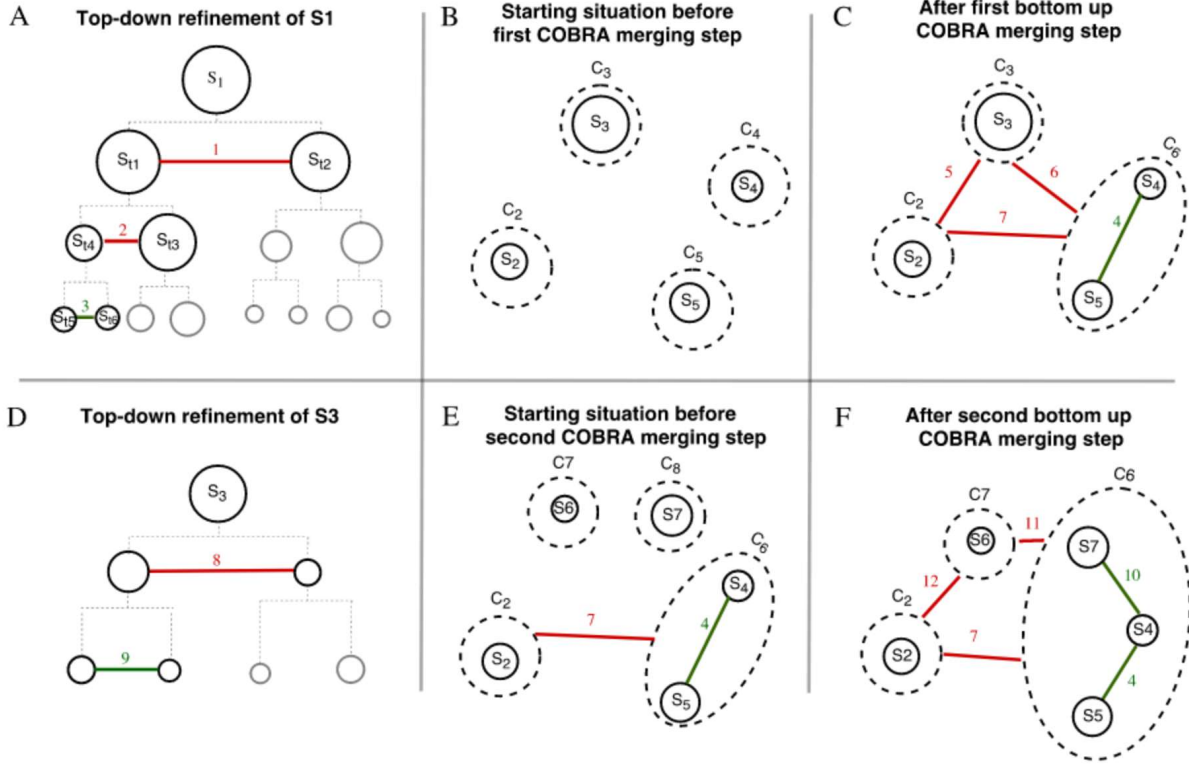


FIGURE 3.1 – Exemple illustratif du fonctionnement de COBRAS, tiré de [Craenendonck et al. \(2018\)](#). Les contraintes ML sont en vert et les CL en rouge.

les scissions (DTW ([Sakoe et Chiba, 1978](#)), SBD ([Paparrizos et Gravano, 2016](#))), en lieu et place de KMEANS et de la distance euclidienne.

Enfin, une amélioration récente nommée COBRAS+ ([Yurtman et al., 2021](#)) diminue encore le nombre de requêtes en déterminant automatiquement le statut (ML ou CL) d’une contrainte entre deux clusters si elle est très similaire à une contrainte déjà connue sur des points des mêmes clusters. La requête ML classique (ligne 10 de l’algorithme 13, ligne 16 de l’algorithme 14) est remplacée par la fonction `analyse_relation` (cf. Algorithme 15). La fonction de dissimilarité utilisée (Éq. 3.7) mesure la distance euclidienne entre les points de chaque contrainte :

$$d((x_{A_1}, x_{A_2}), (x_{B_1}, x_{B_2})) = \frac{\min [\max(|x_{A_1}x_{B_1}|, |x_{A_2}x_{B_2}|), \max(|x_{A_1}x_{B_2}|, |x_{A_2}x_{B_1}|)]}{\frac{|x_{A_1}x_{A_2}| + |x_{B_1}x_{B_2}|}{2}} \quad (3.7)$$

Si la dissimilarité avec la contrainte la plus proche est plus faible qu’un seuil arbitraire τ , le statut de la contrainte proche est directement propagé à la paire considérée pour requête. COBRAS+ peut de plus prendre un ensemble de contraintes préexistant en entrée de l’algorithme pour réaliser ces inférences dès la première itération.

3.2.2.2 Approches déclaratives

Plusieurs approches déclaratives ont été proposées dans le but de modifier une partition existante pour satisfaire des contraintes utilisateur. La flexibilité des paradigmes déclaratifs

Algorithme 15 analyze_relation

Entrée: paire de points de la requête $\mathbf{s}_a, \mathbf{s}_b$, ensembles de contraintes ML et CL $\mathcal{C}_{ML}, \mathcal{C}_{CL}$, seuil de dissimilarité τ

Sortie: r : relation entre $\mathbf{s}_a$ et $\mathbf{s}_b$ (ML ou CL), ensembles de contraintes mis à jour $\mathcal{C}_{ML}, \mathcal{C}_{CL}$

1: **si** $\mathcal{C}_{ML} \cup \mathcal{C}_{CL} \neq \emptyset$ **alors**

2: $Z^* \leftarrow \arg \min_{(s'_a, s'_b)} \{d((s_a, s_b), (s'_a, s'_b)) \mid s'_a \in p_a, s'_b \in p_b, (s'_a, s'_b) \in \mathcal{C}_{ML} \cup \mathcal{C}_{CL}\}$ ▷

 Contrainte la plus similaire

3: **si** $d((s_a, s_b), Z^*) \leq \tau$ **alors** ▷ Inférence de la relation si contraintes similaires

4: **si** $Z^* \in \mathcal{C}_{ML}$ **alors**

5: $r \leftarrow ML(s_a, s_b)$

6: **sinon**

7: $r \leftarrow CL(s_a, s_b)$

8: **sinon**

9: $r \leftarrow \text{requête}(s_a, s_b)$

10: **sinon**

11: $r \leftarrow \text{requête}(s_a, s_b)$

▷ Fonctionnement normal de COBRAS

12: **si** $r = ML(s_a, s_b)$ **alors**

13: $\mathcal{C}_{ML} \leftarrow \mathcal{C}_{ML} \cup \{ML(s_a, s_b)\}$

14: **sinon**

15: $\mathcal{C}_{CL} \leftarrow \mathcal{C}_{CL} \cup \{CL(s_a, s_b)\}$

16: **retourner** $r, \mathcal{C}_{ML}, \mathcal{C}_{CL}$

permet de combiner différents types de contraintes ou de changer facilement la spécification d'une contrainte particulière.

Modification minimale de clustering. Kuo et al. (2017) présente un modèle en programmation par contraintes (cf. Section 2.4) qui, partant d'une partition $\mathcal{P}_t$, produit une partition modifiée $\mathcal{P}_{t+1}$ de façon à respecter des contraintes de diamètre des clusters. Ces contraintes limitent la plus grande distance entre deux points d'un même cluster. Dans ce modèle, il est possible de définir le diamètre attribut par attribut : on peut donner une distance à respecter pour chaque attribut (on compare alors les distances entre les points par rapport à cet attribut uniquement), et choisir de ne que limiter le diamètre que selon certains attributs. Le modèle minimise le nombre de points déplacés par rapport à la partition en entrée (Éq. 3.8), encodés par les variables binaires z_i (Éq. 3.9) qui valent 1 si le point i a changé d'affectation entre $\mathcal{P}_t$ et $\mathcal{P}_{t+1}$ et 0 sinon. Les affectations des points dans $\mathcal{P}_{t+1}$ sont réalisées via une matrice d'allocation binaire de dimensions $n \times k$ encodée par les variables binaires $G_{C,i}$. Dans cette matrice, on a $G_{C,i} = 1$ si le point i est affecté au cluster C dans $\mathcal{P}_{t+1}$. En l'occurrence, k est connu : il s'agit du nombre de clusters de $\mathcal{P}_t$.

$$\arg \min \sum_{i=1}^n z_i \quad (3.8)$$

tel que

$$z_i = \mathbb{I}[\mathcal{P}_{t+1} \neq \mathcal{P}_t], \quad \forall i \in \{1, n\} \quad (3.9)$$

$$G_{C,i} = \mathbb{I}[\mathcal{P}_{t+1} = C], \quad \forall C, i \quad (3.10)$$

Les contraintes de diamètre pour chaque cluster et chaque attribut sont formulées par l'équation 3.13 où $L_{C,t}$ et $H_{C,t}$ sont les bornes inférieure et supérieure des valeurs des points affectés au cluster C dans $\mathcal{P}_{t+1}$ pour l'attribut t , autrement dit le diamètre de C pour l'attribut t . $D'_{C,t}$ représente le diamètre de C spécifié par l'utilisateur pour l'attribut t . Les bornes sont calculées en fonction de l'étendue des attributs (voir Définition 2.1), en particulier de leurs valeurs minimale E_t^- et maximale E_t^+ . Pour résumer, les contraintes 3.11 et 3.12 permettent de calculer le diamètre des clusters de $\mathcal{P}_{t+1}$ en fonction des affectations du modèle ; la contrainte 3.13 vérifie que les diamètres choisis répondent aux attentes de l'utilisateur.

$$L_{C,t} = \min_{i=1}^n \{G_{C,i}(x_{i,t} - E_t^+)\} + E_t^+, \quad \forall C, t \quad (3.11)$$

$$H_{C,t} = \max_{i=1}^n \{G_{C,i}(x_{i,t} - E_t^-)\} + E_t^-, \quad \forall C, t \quad (3.12)$$

$$H_{C,t} - L_{C,t} \leq D'_{C,t}, \quad \forall C, t \quad (3.13)$$

La complexité du modèle est décrite par les tables 3.2a et 3.2b. Les contraintes de diamètre ont l'avantage de limiter le nombre de contraintes. Il faudrait n^2 contraintes ML/CL pour représenter ce qui est exprimé par une unique contrainte de diamètre. On évite donc une croissance quadratique du nombre de contraintes. La possibilité de spécifier partiellement les diamètres réduit aussi le nombre de variables et de contraintes. Le modèle peut être étendu pour gérer des contraintes de séparation entre les clusters ou de taille de cluster.

Variables	Nombre	Domaine
$\mathcal{P}_{t+1}$	n	$\{1, \dots, k\}$
z	n	$\{0, 1\}$
G	nk	$\{0, 1\}$
L	km	E_t
H	km	E_t

(a) Nombre et domaines des variables.

Contraintes	Nombre	Portée
Encoder z (3.9)	n	2
Encoder G (3.10)	nk	2
Encoder L (3.11)	km	$n + 1$
Encoder H (3.12)	km	$n + 1$
Diamètre (3.13)	km	2

(b) Nombre de contraintes et variables impliquées.

TABLE 3.2 – Complexité du modèle de [Kuo et al. \(2017\)](#) en termes de variables et de contraintes. Les domaines des variables indiquent les valeurs pouvant leur être affectées. La portée d’une contrainte est le nombre de variables qui y sont impliquées. E_t est l’étendue d’un attribut quelconque t (cf. Définition 2.1) parmi les m attributs de $\mathcal{X}$.

Par souci de clarté, le modèle complet s’écrit comme suit :

$$\arg \min \sum_{i=1}^n z_i \quad (3.14)$$

tel que

$$z_i = \mathbb{I}[\mathcal{P}_{t+1} \neq \mathcal{P}_t], \quad \forall i \in \{1, n\} \quad (3.15)$$

$$G_{C,i} = \mathbb{I}[\mathcal{P}_{t+1} = C], \quad \forall C, i \quad (3.16)$$

$$L_{C,t} = \min_{i=1}^n \{G_{C,i}(x_{i,t} - E_t^+)\} + E_t^+, \quad \forall C, t \quad (3.17)$$

$$H_{C,t} = \max_{i=1}^n \{G_{C,i}(x_{i,t} - E_t^-)\} + E_t^-, \quad \forall C, t \quad (3.18)$$

$$H_{C,t} - L_{C,t} \leq D'_{C,t}, \quad \forall C, t \quad (3.19)$$

Ajout des contraintes en post-traitement. Une autre approche basée sur la programmation linéaire en nombres entiers (PLNE) ([Nghiem et al., 2020](#)) permet de modifier une partition existante pour qu’elle satisfasse des contraintes. La PLNE est un autre paradigme déclaratif où les contraintes sont linéaires (pas d’inégalités strictes). La résolution consiste à chercher la solution optimale dans un polygone formé par les équations linéaires à l’aide de méthodes comme l’algorithme du simplexe ou des points intérieurs ([Karmarkar, 1984](#)). La partition en entrée du problème est représentée sous la forme d’une matrice d’allocation probabiliste des points aux clusters $\mathcal{M}^a$ de dimensions $n \times k$. $\mathcal{M}_{i,C}^a$ est une valeur entre 0 et 1 représentant la probabilité que le point x_i appartienne au cluster C . Cette matrice peut être obtenue avec des méthodes de *deep clustering* calculant la probabilité d’affectation des points comme IDEC ([Guo et al., 2017](#)). La PLNE effectue une binarisation de la matrice d’allocation en entrée. En effet, la partition en sortie est représentée par une matrice d’allocation binaire B de mêmes dimensions que $\mathcal{M}^a$ où $B_{i,C}$ vaut 1 si le point x_i est affecté au cluster C et 0 sinon. Les variables du PLNE sont les cases de la matrice B . La fonction objectif (Éq. 3.20) vise à affecter — dans la partition binarisée — chaque point au cluster auquel il a le plus de chances d’appartenir selon $\mathcal{M}^a$.

$$\arg \max \sum_{i=1}^n \sum_{C=1}^k \mathcal{M}_{i,C}^a B_{i,C} \quad (3.20)$$

Cette binarisation peut être sujette aux contraintes suivantes :

— ML/CL entre deux points x_i et x_j :

$$B_{i,C} = B_{j,C}, \quad \forall C \in [1, k] \quad (3.21)$$

$$B_{i,C} + B_{j,C} \leq 1, \quad \forall C \in [1, k] \quad (3.22)$$

— Triplet entre les points x_a, x_p, x_n :

$$B_{p,C} \geq B_{n,C} + B_{a,C} - 1, \quad \forall C \in [1, k] \quad (3.23)$$

$$B_{n,C} \geq B_{p,C} + B_{a,C} - 1, \quad \forall C \in [1, k]$$

— Contrainte de chevauchement : on peut affecter un point à plusieurs clusters en même temps dans la matrice d'allocation binaire B . Cette possibilité de chevauchement des clusters peut être contrôlée en fixant des bornes α et β au nombre d'affectations possibles pour un même point. S'il n'y a pas de contrainte sur le chevauchement, le modèle va affecter tous les points à tous les clusters car cela maximise l'objectif. Pour obtenir une partition classique où chaque point est affecté à un seul cluster, il suffit de fixer $\alpha = \beta = 1$.

$$\alpha \leq \sum_{C=1}^k B_{i,C} \leq \beta, \quad \forall i \in [1, n] \quad (3.24)$$

— Taille de cluster : de manière similaire, on peut exprimer qu'un cluster doit contenir au moins α et au plus β points. Les contraintes de chevauchement portent sur les lignes de la matrice d'allocation, les contraintes de taille sur les colonnes.

$$\alpha \leq \sum_{i=1}^n B_{i,C} \leq \beta, \quad \forall C \in [1, k] \quad (3.25)$$

— Limite d'attribut par cluster : pour assurer qu'un cluster ne contient pas que des points avec les mêmes valeurs pour certains attributs (pour des questions d'équité par exemple), on peut raffiner la contrainte de taille. Soit $p(i)$ une fonction booléenne donnant 1 si le point i a la propriété p et 0 sinon⁷. La contrainte suivante borne le nombre de points ayant la propriété p dans un cluster entre α et β :

$$\alpha \leq \sum_{i=1}^n p(i) B_{i,C} \leq \beta, \quad \forall C \in [1, k] \quad (3.26)$$

— Limite de clusters par attribut : à l'inverse de la contrainte précédente, il peut être utile que certaines propriétés des points soient restreintes à certains clusters (pour

7. La propriété peut être un attribut binaire, un seuil sur la valeur d'un attribut numérique ou un sous-ensemble des modalités d'un attribut catégoriel.

les décrire facilement par exemple). A l'aide de variables binaires supplémentaires t telles que $t_C = 1$ si le cluster C contient au moins un point vérifiant la propriété p , on peut contrôler le nombre de clusters possédant p :

$$\begin{aligned} t_C &\leq \sum_{i=1} p(i)B_{i,C} \quad \forall C \in [1, k] \\ t_C &\geq p(i)B_{i,C} \quad \forall i \in [1, n], C \in [1, k] \\ \alpha &\leq \sum_{C=1}^k t_C \leq \beta \end{aligned} \tag{3.27}$$

Cette contrainte est très similaire à la contrainte d'étendue limitée générique (cf. Définition 2.10) qui n'a pas de condition sur les propriétés des points mais porte sur un groupe de points désigné par l'expert(e).

- Contrainte de voisinage : soit $N_i(j)$ une fonction booléenne donnant 1 si i est voisin de j et 0 sinon⁸. Une contrainte de voisinage oblige chaque point à avoir la même affectation qu'une proportion minimale α de ses voisins.

$$\sum_{j=1}^n N_i(j)B_{j,C} \geq \alpha \left(\sum_{j=1}^n N_i(j) \right) B_{i,C}, \quad \forall i \in [1, n], C \in [1, k] \tag{3.28}$$

			Contraintes	Portée
Variables	Nombre	Domaine	ML/CL (3.21, 3.22)	2
			Triplet (3.23)	3
			Chevauchement (3.24)	k
			Taille de cluster (3.25)	n
			Voisinage (3.28)	nk
			Limite d'attribut par cluster (3.26)	n
			Limite de clusters par attribut (3.27)	$k + nk$

(a) Nombre et domaines des variables.

(b) Contraintes utilisateur gérées par le PLNE.

TABLE 3.3 – Complexité du modèle de (Nghiem et al., 2020) en termes de variables et de contraintes.

3.2.2.3 Clustering collaboratif

Plus récemment, Lafabregue et al. (2022) présente un système de clustering collaboratif sous contraintes appelé I-SAMARAH dont le fonctionnement général est décrit en Figure 3.2. Plusieurs agents - des algorithmes de clustering - calculent différentes partitions d'un jeu de données, puis I-SAMARAH identifie les conflits entre les partitions des agents pour les modifier.

8. On ne se préoccupe pas de la façon dont cette relation est trouvée. Un exemple simple est le calcul des k plus proches voisins de chaque point.

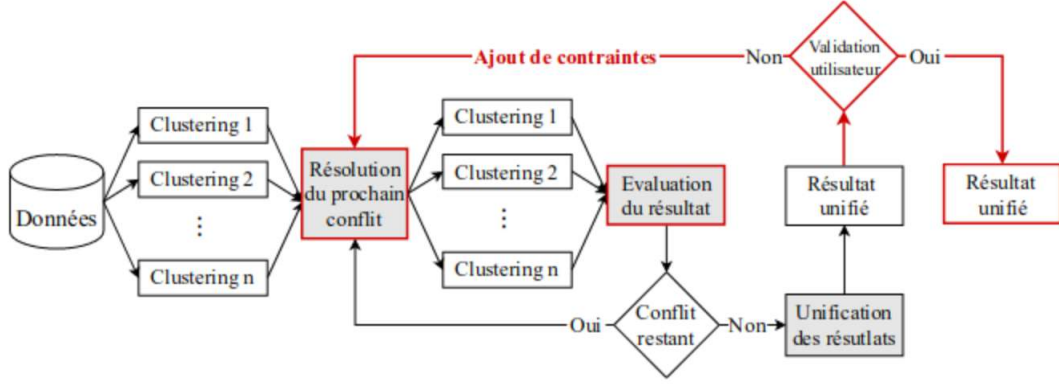


FIGURE 3.2 – Schéma de fonctionnement de I-SAMARAH tiré de Lafabregue (2021).

Quand tous les conflits ont été résolus, un résultat final est désigné par vote des agents. La résolution de conflit cherche une solution qui améliore le plus un critère local inter-agents :

$$\gamma^{(i,j)} = \frac{1}{2} \left(p_s \left(\frac{1}{k_i} \sum_{c=1}^{k_i} \omega_c^{(i,j)} + \frac{1}{k_j} \sum_{c=1}^{k_j} \omega_c^{(j,i)} \right) + p_q(\delta^i + \delta^j) + p_c(\theta^i + \theta^j) \right) \quad (3.29)$$

où $\omega_c^{(i,j)}$ est la similarité entre le cluster c de l'agent i et le cluster qui se rapproche le plus de c chez l'agent j , δ^i est la qualité interne de la partition de l'agent i (mesuré avec des métriques comme l'indice de Silhouette (Rousseeuw, 1987) qui favorise les clusters compacts et éloignés entre eux) et θ^i est le taux de contraintes satisfaites chez l'agent i . Les poids $p_s + p_q + p_c = 1$ quantifient l'importance donnée à la similarité entre les agents, à la qualité interne des partitions et à la satisfaction des contraintes respectivement. Le critère global est la moyenne des critères locaux pour tous les agents différents deux à deux. Les contraintes ML/CL sont dispersées entre les différents agents et servent à provoquer des différences par la création de nouveaux clusters (Algorithme 16), pour créer de nouveaux conflits et contrebalancer la tendance de I-SAMARAH à l'homogénéisation des agents.

Algorithme 16 I-SAMARAH - Perturbation des agents avec les contraintes

Entrée: Ensemble d'agents $\mathcal{A}$, ensemble de contraintes $\mathcal{C}$

- 1: **pour** chaque contrainte $c \in \mathcal{C}$ **faire**
 - 2: $A' = \emptyset$ ▷ Agents qui ne satisfont pas la contrainte c
 - 3: **pour** chaque agent $a_i \in \mathcal{A}$ **faire**
 - 4: **si** a_i ne satisfait pas c **alors**
 - 5: $A' = A' \cup a_i$
 - 6: **si** $A' \neq \emptyset$ **alors**
 - 7: Choisir aléatoirement $a \in A'$ tel que M^a est l'ensemble des centroïdes de a
 - 8: **si** c est une contrainte ML **alors**
 - 9: $M^a = M^a \cup \{avg(\{c_1; c_2\})\}$ ▷ Ajouter un centroïde entre les points
 - 10: **sinon si** c est une contrainte CL **alors**
 - 11: $M^a = M^a \cup \{c_1; c_2\}$ ▷ Ajouter les deux points comme centroïdes
-

3.2.3 Discussion

Les méthodes qui viennent d'être présentées peuvent toutes être vues comme des implémentations concrètes de l'algorithme 12. Soit leur fonctionnement peut être décomposé en une étape de sélection des contraintes (manuelle ou assistée par un algorithme) et une étape de modification d'une partition actuelle, soit elles ne font qu'une modification mais se placent dans un contexte de raffinement d'une partition déjà calculée, pouvant nécessiter plusieurs itérations pour obtenir un résultat satisfaisant. Certaines méthodes ont également une étape d'initialisation bien définie, comme [Lai et al. \(2014\)](#) avec BIRCH. La Table 3.4 catégorise les méthodes selon la façon dont elles implémentent ces étapes, le type de contraintes géré et le critère de satisfaction.

Types de contraintes

Quasiment toutes les méthodes se focalisent sur un type de contraintes en particulier, en général les ML/CL. Seul le modèle en PLNE de [Nghiem et al. \(2020\)](#) utilise la flexibilité des approches déclaratives pour intégrer des contraintes de différents types, agissant au niveau des clusters et des points. Cependant ce modèle nécessite d'avoir une matrice d'allocation probabiliste, ce qui induit un prétraitement supplémentaire pouvant potentiellement introduire des biais si l'algorithme de clustering utilisé au départ ne produit pas de partition souple. Chez [Kuo et al. \(2017\)](#), la flexibilité réside dans la possibilité de définir un diamètre pour chaque attribut du jeu de données en entrée. Cela permet de relâcher le problème, soit en modifiant le diamètre pour un attribut, soit en réduisant les attributs devant respecter un diamètre défini.

Gestion des conflits

Seuls [Lai et al. \(2014\)](#) mentionnent que des contradictions peuvent émerger entre les différentes itérations de leur système ; ils traitent ce problème en utilisant une stratégie LRU (*Least Recently Used*) consistant à oublier progressivement les retours les plus anciens.

Aucun modèle sans satisfaction souple des contraintes ne gère les conflits entre les contraintes. Cela est d'autant plus important pour [Nghiem et al. \(2020\)](#) où la diversité des types de contraintes accroît la complexité de la détection d'un conflit, y compris par un humain.

Assistance à l'utilisateur

La moitié des approches fournissent une aide pour la sélection des contraintes, cependant elles donnent rarement la possibilité de donner également ses propres contraintes. Seul I-SAMARAH [Lafabregue et al. \(2022\)](#) permet explicitement de faire les deux.

Quand les approches ne proposent pas d'aide à la sélection de contraintes, ce sont pour la plupart des méthodes visuelles qui sont éprouvées sur des données textuelles et/ou des images qu'un humain peut ranger facilement. L'applicabilité de ces méthodes sur des données plus complexes à appréhender est donc discutable.

Dans notre travail, nous avons utilisé la programmation par contraintes pour permettre la gestion de types de contraintes variées, comme le font [Kuo et al. \(2017\)](#) et [Nghiem et al.](#)

(2020). Cependant, nous intégrons également un mécanisme pour gérer les conflits entre les contraintes.

En outre, notre proposition pour l'étape de modification est focalisée sur le maintien de la similarité entre l'entrée et la sortie, un aspect qui n'est quasiment pas abordé dans la littérature, si ce n'est Davidson et al. (2007) qui comparent le nombre de réaffectations effectuées par leur méthode par rapport à un reclustering total.

Approche	Contraintes	Satisfaction	Sélection	Modification
Cohn et al. (2001)	CL	souple	heuristique	EM
Davidson et al. (2007)	ML/CL	rigide	manuelle	heuristique
Basu et al. (2010)	ML/CL	souple	<i>metric learning</i> classification	manuelle
Prabakara Raj et Ravindran (2013)	ML/CL	souple	matrice d'utilité	clustering sous contraintes
Lai et al. (2014)	ML/CL	rigide	manuelle	EM
Awasthi et al. (2015)	fusion/scission	rigide	manuelle	fusion/scission
INCREMENT (Mitchell, 2016)	étiquetage	rigide	sous-clustering	KMEANS en espace latent
URASC (Xiong et al., 2016)	ML/CL	rigide	réduction d'incertitude	clustering spectral sous contraintes
COBRA COBRAS (Craenendonck et al., 2018) COBRAS ^{TS} COBRAS+ (Yurtman et al., 2021) (Kuo et al., 2017)	ML/CL	rigide	sous-clustering	fusion/scission
	diamètre	rigide	manuelle	PPC
(Nghiem et al., 2020)	ML/CL triplet chevauchement taille voisinage	rigide	manuelle	PLNE
I-SAMARAH (Lafabregue et al., 2022)	ML/CL	souple	manuelle clustering actif	clustering collaboratif

TABLE 3.4 – Comparaison des approches présentées dans la Section 3.2.2. La colonne "Sélection" indique la méthode utilisée pour obtenir les contraintes. La colonne "Modification" indique la méthode employée pour recalculer ou modifier la partition existante. On entend par "manuelle" une étape qui ne propose aucune aide à l'utilisateur.

Chapitre 4

Clustering actif

Table des matières

4.1	Apprentissage actif	58
4.1.1	Scénarios d'apprentissage	58
4.1.2	Stratégies de requête	59
4.2	Clustering actif	61
4.3	Voisinages	62
4.4	Travaux antérieurs en clustering actif	64
4.4.1	Algorithmes basés sur les voisinages	64
4.4.2	Apprentissage de contraintes triplet	70
4.4.3	Estimer le changement apporté à la partition par une nouvelle contrainte	71
4.4.4	Propager les contraintes avec les voisins	72
4.4.5	Ordonner les voisins de points étiquetés	74
4.4.6	Sélection séquentielle d'un ensemble de contraintes	74
4.4.7	Discussion	77

Le résultat d'un algorithme de clustering sous contraintes va fortement dépendre des contraintes qu'il a reçu. Davidson et al. (2006) ont notamment montré que l'utilisation de contraintes ne garantit pas de converger vers un résultat prédéterminé, et qu'elles peuvent même avoir un impact délétère sur la performance. De plus, l'intérêt d'une contrainte dépend de l'algorithme qui l'emploie : si la contrainte est déjà satisfaite dans une partition obtenue sans contraintes, elle est inutile pour l'algorithme qui a calculé la partition.

Choisir les bonnes contraintes est donc un problème en soi. Le clustering sous contraintes incrémentales pose aussi la question de la **pénurie de contraintes** : que faire si l'utilisateur n'est pas satisfait d'un résultat, mais qu'il ou elle n'est pas pour autant capable de donner de retours ? Une telle situation peut pousser à quitter le processus, ou à donner des contraintes peu pertinentes pour pouvoir faire une modification supplémentaire. La capacité de générer automatiquement des contraintes candidates que l'expert(e) doit simplement vérifier devient alors primordiale.

Le clustering actif offre une solution aux problèmes évoqués. Un algorithme analysant le jeu de données et la partition courante pour faire des requêtes à l'utilisateur débouchant sur des contraintes pertinentes permet (1) d'éviter toute pénurie de contraintes et (2) que

le lot de contraintes apporte de l'information sur la partition voulue.

Dans ce chapitre, nous abordons l'apprentissage actif (Section 4.1) pour présenter les enjeux et les stratégies employées pour identifier les requêtes à faire à l'utilisateur. Nous présentons ensuite le domaine du clustering actif (Section 4.2) qui transpose les techniques d'apprentissage actif de l'étiquetage à la sélection de contraintes. Nous nous attardons sur la structure des *voisinages* employée dans plusieurs algorithmes de l'état de l'art ainsi que dans nos contributions (Section 4.3). Nous faisons un état de l'art du clustering actif, avec une focalisation sur les algorithmes utilisant les voisinages tout en présentant des algorithmes utilisant d'autres stratégies afin de montrer la diversité du domaine (Section 4.4).

4.1 Apprentissage actif

La classification consiste à entraîner un algorithme d'apprentissage supervisé sur des données étiquetées selon différentes classes, afin d'apprendre un modèle mathématique capable de séparer les classes. Pour assurer un entraînement efficace, ces algorithmes ont besoin de grandes masses de données. Or il n'est pas toujours possible d'avoir accès à de grandes quantités de données : l'étiquetage peut être difficile dû à la complexité du domaine de connaissances, ou les données d'intérêt peuvent n'être disponibles qu'en nombre limité (c'est le cas de certains types de données historiques par exemple). On peut aussi vouloir réduire intentionnellement l'utilisation des données dans une démarche d'IA frugale.

L'apprentissage actif¹ repose sur l'idée qu'on peut entraîner efficacement un classifieur avec peu de données si celles-ci contribuent fortement à différencier les classes à apprendre. L'avantage principal est de réduire l'effort demandé à l'humain : un algorithme se charge de choisir quel point étiqueter, et on ne va étiqueter d'une quantité limitée de points. À partir d'un jeu de données $\mathcal{X}$ et d'un modèle d'apprentissage supervisé, l'apprentissage actif vise à trouver le sous-ensemble de $\mathcal{X}$ qui contribue le plus à la performance du modèle tout en étant aussi petit que possible.

4.1.1 Scénarios d'apprentissage

Différents scénarios d'interaction avec l'utilisateur chargé d'étiqueter les données ont émergé au fur et à mesure :

- la génération de requête (Angluin, 1988) : on crée un point de données à partir de l'espace de classification pour poser une requête. Le risque de ce scénario est de générer des requêtes qui ne correspondent à aucune classe. Un exemple est donné dans Baum et Lang (1992) sur un jeu de données d'images de chiffres où la génération peut mener à des images ne correspondant à aucun chiffre en particulier. Cela se produit car les classes recherchées ne couvrent pas l'ensemble de l'espace de classification : il est tout à fait possible de générer une image en noir et blanc sans qu'elle ne ressemble à un quelconque chiffre.

1. L'état de l'art de Settles (2009) est une référence dans le domaine ; on peut aussi citer un chapitre dédié de Aggarwal (2014), l'étude de Kumar et Gupta (2020) qui traite également de clustering actif, et enfin l'étude de Li et al. (2024) focalisée sur l'apprentissage profond.

- le tirage séquentiel (*stream-based sampling*) (Cohn et al., 1994) : on traite le jeu de données en entrée comme un flux ; à chaque point entrant, l'algorithme d'apprentissage décide s'il est utile de l'étiqueter ou non. Ce scénario peut être utile sur de grands jeux de données, lorsque la mémoire est limitée ou lorsque l'entrée peut être représentée par une distribution uniforme (on se rapproche alors du premier scénario).
- l'échantillonnage du jeu de données (*pool-based sampling*) (Lewis et Gale, 1994) : on analyse l'ensemble des données connues avant de choisir un point particulier d'après un critère. Contrairement au tirage séquentiel, on est certain de choisir le meilleur point possible d'après le critère choisi et on peut converger vers un résultat satisfaisant en limitant les requêtes.

Les algorithmes de clustering actif que nous examinerons dans l'état de l'art (Section 4.4) ainsi que notre contribution dans ce domaine (Chapitre 7) se placent dans ce dernier scénario, tout comme la majorité des approches d'apprentissage actif. On peut tenter d'expliquer cette prédilection pour l'échantillonnage en notant que lors de l'émergence du clustering actif dans les années 2000, le domaine de l'apprentissage actif s'était éloigné des approches formelles des années 80² pour être étudié par des chercheurs plus orientés sur le côté expérimental. Or les jeux de données couramment utilisés à l'époque pour les expériences se prêtent bien à l'échantillonnage, ne reposent pas sur des distributions uniformes et sont pour la plupart assez petits pour ne pas nécessiter de tirage séquentiel.

4.1.2 Stratégies de requête

Quel que soit le scénario utilisé pour faire des requêtes à l'utilisateur, la question centrale de l'apprentissage actif est de déterminer quels vont le plus améliorer la performance d'un classifieur une fois étiquetés. De nombreuses stratégies ont été développées dans ce but, qu'on peut répartir en trois grandes catégories :

- Méthodes basées sur l'hétérogénéité : le but est de cibler les zones de transition entre les classes pour améliorer rapidement la capacité du modèle à les séparer correctement. Ces méthodes dépendent de trois grandes stratégies :
 - Échantillonnage par incertitude : l'algorithme sélectionne les points sur lesquels il est le moins confiant ; par exemple ceux qui n'ont pas une forte probabilité d'appartenance à une classe particulière (Culotta et McCallum, 2005 ; Lewis et Gale, 1994) ;
 - Requête par vote (*Query-by-committee*) : un ensemble de modèles est utilisé pour étiqueter les points ; on demande à l'expert(e) d'étiqueter les points pour lesquels les modèles sont le plus en conflit (Melville et Mooney, 2004 ; Seung et al., 1992) ;
 - Changement attendu du modèle : les points sont choisis selon leur capacité à impacter les paramètres ou la fonction objectif du modèle (Settles et al., 2007).
- Méthodes basées sur la performance : les résultats du modèle sur les points non étiquetés sont utilisés comme base pour choisir les prochaines requêtes. Il s'agit donc d'une évaluation globale, là où les approches basées sur l'hétérogénéité se concentrent sur chaque point individuellement (MacKay, 1992 ; Roy et McCallum,

2. Une partie des concepts utilisés par Angluin (1988) peuvent être rapprochés de la programmation logique inductive (Cropper et Dumančić, 2022), apparue elle aussi au début des années 90.

- 2001);
- Échantillonnage exploratoire : ces méthodes s'assurent de sélectionner des points venant de régions variées de l'espace de clustering pour éviter les requêtes redondantes (Brinker, 2003 ; Wu et al., 2006 ; Xu et al., 2007).

Les différentes stratégies correspondent au **compromis entre exploitation et exploration** que l'on retrouve dans de nombreux domaines de l'apprentissage automatique et plus généralement dans toute discipline qui étudie la prise de décision d'agents dans un environnement (Mehlhorn et al., 2015). Ici, ce compromis consiste à arbitrer entre l'optimisation du classifieur dans les régions connues, au risque de mal généraliser dans le reste de l'espace, ou l'exploration de régions diverses pour s'assurer de connaître toutes les zones de transition entre les classes quitte à moins bien les séparer. Dans la Figure 4.1, les requêtes de points très éloignés de la classe positive (*exploration*) ont permis, au prix d'une dizaine d'étiquetages négatifs, de délimiter une zone d'incertitude réduite sur laquelle les requêtes restantes peuvent se focaliser pour préciser la séparation entre la classe positive et la classe négative (*exploitation*).

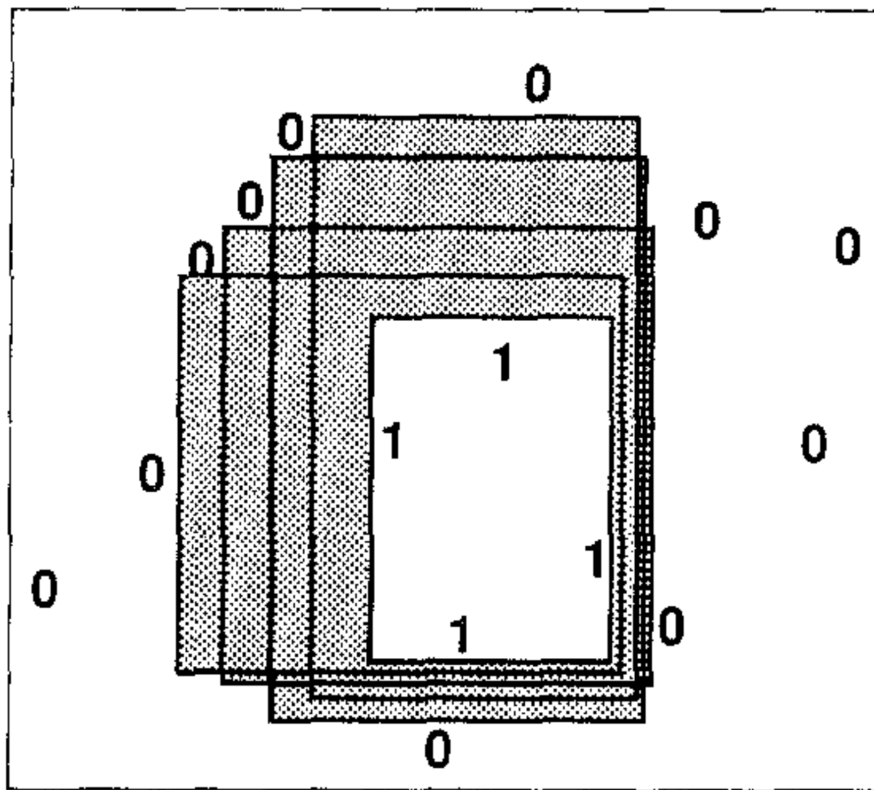


FIGURE 4.1 – Problème illustratif de classification binaire avec apprentissage actif. Chaque rectangle est borné par les exemples étiquetés par l'utilisateur, soit positifs (les **1** à l'intérieur), soit négatifs (les **0** à l'extérieur) et délimite une partie de l'espace possible pour la classe positive. La zone d'incertitude (en gris hachuré) se trouve entre l'enveloppe convexe des exemples positifs et l'espace peuplé par les exemples négatifs. Image extraite de Cohn et al. (1994).

4.2 Clustering actif

Le clustering actif reprend les principes de l'apprentissage actif décrits dans la section précédente pour l'appliquer au cas où l'expert(e) ne peut pas étiqueter les points. Les requêtes cherchent à obtenir de l'information de façon plus indirecte, par exemple en demandant les relations entre certains points, sans avoir de précision sur ce qui les relie. On peut catégoriser les méthodes de clustering actif selon leur sortie :

- **Méthodes agnostiques** : ces algorithmes opèrent indépendamment d'un quelconque algorithme de clustering. Partant d'un jeu de données et éventuellement d'une partition, ils produisent un ensemble de contraintes à l'aide d'un budget de requêtes fixé à l'avance. Ces contraintes peuvent ensuite être utilisées par tout algorithme de clustering sous contraintes capable de les gérer. Les techniques qui ne se basent que sur les données ont tendance à être plus déterministes : à moins de contenir une composante aléatoire, le même ensemble de contraintes sera produit pour le même jeu de données, quelle que soit la partition calculée.
- **Méthodes composites** : des techniques combinant sélection de contraintes et clustering, les deux parties étant conçues pour fonctionner de concert. Les requêtes sont générées dynamiquement suivant l'état de la partition construite au cours de l'exécution, sans budget fixé. Ce sont des algorithmes *anytime*, que l'on peut arrêter à tout moment pour récupérer la partition courante en sortie.

On peut considérer les méthodes composites comme un sous-ensemble des algorithmes de clustering sous contraintes incrémentales. En effet, elles alternent entre la sélection de contraintes et leur utilisation pour modifier ou recalculer une partition. COBRAS (Craenendonck et al., 2018) est un exemple de méthode composite : les requêtes faites à l'utilisateur sous forme de ML/CL entre des super-instances sont utilisées pour modifier la partition, d'abord pour déterminer la granularité de division d'un cluster, puis pour les fusions. Le clustering sous contraintes incrémentales n'est pas équivalent à cette famille du clustering actif car il ne suppose pas que les contraintes soient sélectionnées automatiquement. Comme on l'a vu dans la Table 3.4, la moitié des méthodes examinées laissent entièrement à l'utilisateur le soin de donner ses contraintes.

Les contraintes déduites par les requêtes peuvent prendre des formes variées. Cependant la plupart de la littérature s'est focalisée sur les requêtes sur des paires de points. On passe donc de "À quelle classe appartient ce point ?" pour l'apprentissage actif à "Est-ce que ces deux points devraient être groupés ensemble ?" donnant selon la réponse une contrainte *must-link* ou *cannot-link*. D'autres méthodes, comme Xiong et al. (2015), cherchent à obtenir des contraintes triplet, avec des requêtes du type : "Est-ce que le point x_i est plus similaire au point x_j qu'au point x_k ?". La contrainte triplet obtenue est soit (x_i, x_j, x_k) , soit (x_i, x_k, x_j) selon que x_j est plus proche ou non de x_k d'après l'expert(e).

Il peut arriver que l'expert(e) ne sache pas quelle réponse donner à une requête, ou qu'elle ne soit pas considérée comme intéressante. Certains algorithmes ne considèrent pas du tout cette option et nécessitent qu'une réponse soit donnée ; d'autres autorisent une réponse "je ne sais pas" qui permet de passer à la requête suivante sans ajouter de contrainte et d'éviter de poser à nouveau une requête dessus plus tard. Enfin, on peut remarquer que Xiong et al. (2015) estime la probabilité que l'expert(e) affirme ne pas

savoir comment ordonner un triplet et prend cette possibilité en compte dans le calcul du meilleur triplet à proposer.

4.3 Voisinages

L'accumulation de contraintes ML/CL au fil des requêtes tisse un réseau de contraintes reliant les points sélectionnés entre eux. En clustering, ce réseau tend à se composer en "îlots" de points reliés entre eux par des contraintes ML et aux autres îlots par des contraintes CL. Dans la littérature, cette structure est appelée le **squelette** de la partition et les îlots sont désignés sous le terme de **voisinages**³. En se basant sur cette idée, certaines des méthodes les plus connues en clustering actif se fondent sur la construction de voisinages, formant des groupes de points qui se trouvent dans le même cluster d'après l'utilisateur : ils représentent ainsi la structure des clusters désirés par l'expert(e).

Définition 4.1 (Voisinage). Soit $\mathcal{X}$ un jeu de données et $\mathcal{C}$ un ensemble de contraintes clos par transitivité. Un voisinage $N \subseteq \mathcal{X}$ est un sous-ensemble de $\mathcal{X}$ tel que :

$$\forall x_i, x_j \in \mathcal{X}, (x_i \in N \wedge x_j \in N) \iff ML(x_i, x_j) \in \mathcal{C} \quad (4.1)$$

La présence de contraintes ML dans un voisinage n'est nécessaire qu'à partir de deux points ; N peut contenir un seul point et $\mathcal{C}$ être vide. Un point ne peut appartenir qu'à un unique voisinage.

Propriété 4.1. Soit $\mathcal{X}$ un jeu de données, $\mathcal{C}$ un ensemble de contraintes clos par transitivité et $\mathcal{N}$ un ensemble de voisinages sur $\mathcal{X}$. $\mathcal{N}$ est un squelette de partition et vérifie :

$$\forall x_i, x_j \in \mathcal{X}, N, N' \in \mathcal{N}, N \neq N', (x_i \in N \wedge x_j \in N') \iff CL(x_i, x_j) \in \mathcal{C} \quad (4.2)$$

Démonstration. La preuve par l'absurde repose sur l'hypothèse que tout point de $\mathcal{N}$ est relié aux autres points par une contrainte, ML ou CL.⁴ Dans le premier sens, soit $x_i \in N$, $x_j \in N'$ tel que $N \neq N'$ et $CL(x_i, x_j) \notin \mathcal{C}$. En supposant l'hypothèse vraie, $CL(x_i, x_j) \notin \mathcal{C} \implies ML(x_i, x_j) \in \mathcal{C}$. Or par définition d'un voisinage, $ML(x_i, x_j) \in \mathcal{C}$ implique que x_i et x_j font partie du même voisinage et donc que $N = N'$: il y a contradiction. Dans l'autre sens : $CL(x_i, x_j) \in \mathcal{C} \implies ML(x_i, x_j) \notin \mathcal{C}$. Par définition, x_i et x_j ne peuvent pas faire partie du même voisinage. $\square$

Exemple 4.1. Soit l'ensemble de contraintes $\mathcal{C} = \{ML(x_a, x_b), ML(x_a, x_c), CL(x_c, x_d)\}$. La fermeture transitive de $\mathcal{C}$ donne les contraintes supplémentaires $ML(x_b, x_c)$, $CL(x_a, x_d)$, $CL(x_b, x_d)$. D'après ces contraintes, il existe deux voisinages constituant le squelette $\mathcal{N} = \{\{x_a, x_b, x_c\}, \{x_d\}\}$.

Un exemple d'illustration sur le jeu de données **iris** est donné en Figure 4.2.

On peut noter que l'ensemble des voisinages d'un jeu de données peut être représenté sous la forme d'un graphe où les sommets sont les points présentés à l'utilisateur et les

3. Le terme *neighborhoods* est introduit dès PCKMEANS (Basu et al., 2004) pour désigner les ensembles de points reliés par des ML dans les contraintes en entrée de l'algorithme.

4. Cette hypothèse sous-tend les algorithmes basés sur les voisinages décrits en Section 4.4.1. Une réponse indifférente pour un point ne l'ajoute à aucun voisinage.

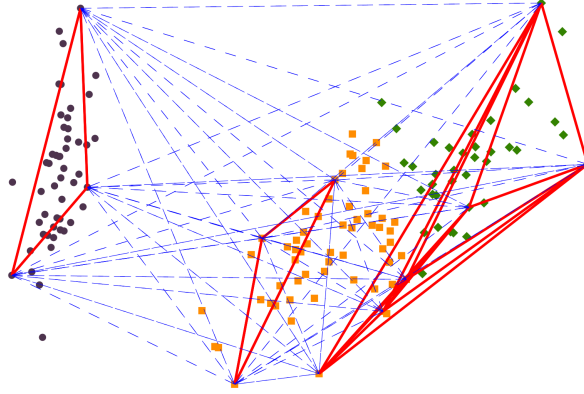


FIGURE 4.2 – Exemple de voisinages obtenus avec la méthode MinMax après 15 requêtes sur le jeu de données *iris*. On distingue trois groupes de points distincts reliés par des contraintes ML (en rouge), et des contraintes CL entre toutes les paires de points de groupes différents (en bleu pointillé).

arêtes sont des contraintes ML/CL⁵. Un voisinage est alors une clique dont les arêtes sont toutes ML.

L'idée commune derrière les algorithmes utilisant des voisinages est de sélectionner itérativement des points et de poser comme requête la relation entre le point sélectionné et un point de chaque voisinage connu : "est-ce que ces deux points sont dans le même voisinage?". Une réponse positive de l'utilisateur correspond à une contrainte ML et permet de déterminer le voisinage auquel appartient le point sélectionné ; une réponse négative retourne une contrainte CL et contraint à faire une requête sur un autre voisinage. Si aucun voisinage ne mène à l'obtention d'une contrainte ML, le point sélectionné est considéré comme faisant partie de son propre voisinage jusqu'ici inconnu. Une fois connu le voisinage d'un point, des contraintes ML/CL sont vers les autres points sont ajoutées suivant la définition 4.1 et la propriété 4.1.

Les voisinages sont une structure utile en clustering actif parce qu'elles permettent d'accumuler l'information des retours utilisateurs qui peut être exploitée pour proposer de nouvelles contraintes (Basu et al., 2004 ; Cai et al., 2016 ; Chen et Jin, 2020 ; Mallapragada et al., 2008 ; Xiong et al., 2014).

Soit $\mathcal{P}_e$ une **partition espérée** en k_e clusters que l'évaluation humaine trouvera assez satisfaisante pour terminer la procédure de clustering sous contraintes incrémentales⁶. Partant de $\mathcal{N} = \emptyset$ et en supposant que les contraintes sont exemptes d'erreur par rapport à $\mathcal{P}_e$, le squelette de $\mathcal{P}_e$ possède les propriétés suivantes :

$$\sum_{i=1}^{|\mathcal{N}|} |N_i| = |\mathcal{X}| \iff \mathcal{P} = \mathcal{P}_e \quad (4.3)$$

5. La distinction entre ML et CL peut être faite à l'aide de poids positifs et négatifs, comme le montre un travail précurseur du clustering sous contraintes (Harary, 1953).

6. Nous ne faisons pas l'hypothèse que $\mathcal{P}_e$ est connue - la tâche étant inutile le cas échéant - mais plutôt que l'algorithme termine si la partition existante est suffisamment proche de $\mathcal{P}_e$ car le résultat remplit l'objectif visé en recourant au clustering (par exemple que des clusters d'intérêt se dégagent assez pour employer d'autres techniques de fouille de données).

$$|\mathcal{N}| \leq k_e \quad (4.4)$$

Les propriétés 4.3 et 4.4 assurent que $\mathcal{N}$ converge vers $\mathcal{P}_e$. Le nombre de voisinages croît jusqu'à atteindre le nombre espéré de clusters, k_e , quel que soit le nombre de clusters présents dans la partition courante.

De plus, la structure de $\mathcal{N}$ correspond à $\mathcal{P}_e$ selon les propriétés suivantes :

$$\forall N \in \mathcal{N}, \exists C_i \in \mathcal{P}_e \mid N \subseteq C_i \quad (4.5)$$

La propriété 4.5 indique que tout voisinage N est un sous-ensemble d'un unique cluster de $\mathcal{P}_e$, noté $C_N^{\mathcal{P}_e}$.

$$\forall N, N' \in \mathcal{N}, N \neq N' \implies C_N^{\mathcal{P}_e} \neq C_{N'}^{\mathcal{P}_e} \quad (4.6)$$

La propriété 4.6 indique le non-chevauchement des voisinages. Deux voisinages différents ne peuvent pas correspondre au même cluster de $\mathcal{P}_e$.

Ces propriétés ne peuvent être maintenues que si toutes les contraintes sont cohérentes avec $\mathcal{P}_e$. En cas d'erreur, ou si $\mathcal{P}_e$ est modifiée car l'utilisateur a changé d'objectif au cours du processus, il faut enlever les contraintes incriminées du graphe de contraintes et rétablir au besoin les propriétés des voisinages. Enfin, si les voisinages sont construits par des contraintes ML/CL, d'autres types de contraintes peuvent être intégrés en les transformant en ML/CL. Les contraintes triplet indiquent une relation entre trois points qui peut facilement être transformée en une paire de ML/CL dont le statut dépend de l'affectation des différents points. En ce qui concerne la contrainte d'étendue limitée, les points du groupe qui sont déjà affectés dans un cluster de l'étendue de la contrainte peuvent être reliés avec des ML aux autres points du groupe de même affectation. Reste donc les points du groupe devant être réaffectés pour entrer dans la plage de clusters, dont on peut dériver des ML/CL après modification de la partition pour respecter la contrainte d'étendue limitée.

4.4 Travaux antérieurs en clustering actif

Dans la littérature, les travaux sur le clustering actif utilisent régulièrement trois termes et leurs variantes : *active clustering*, *active learning of constraints* ou *active query selection*. Les approches utilisant le terme *active learning of constraints* peuvent souvent se destiner à de l'apprentissage actif, mais le cadre repose sur la même hypothèse de travail : l'impossibilité de demander directement un étiquetage des points. Tout au plus, certaines méthodes supposent l'accès à un petit sous-ensemble de données déjà étiquetées.

Nous allons d'abord présenter en détail les développements majeurs dans les méthodes utilisant les voisinages (Section 4.4.1) puis détailler des méthodes plus diverses tant par leur stratégie que par les requêtes posées à l'utilisateur.

4.4.1 Algorithmes basés sur les voisinages

La première stratégie de requête en clustering actif - publiée en même temps que PCK-MEANS (Basu et al., 2004) - introduit la notion de voisinage et propose l'algorithme FFQS

(FURTHEST-FIRST QUERY SELECTION) constitué de deux étapes appelées EXPLORE et CONSOLIDATE. La première étape (cf. Algorithme 17) consiste à trouver au moins un point de chaque voisinage en traversant le jeu de données par la recherche des points les plus éloignés de ceux déjà connus (*farthest-first traversal*, [Gonzalez, 1985](#)). Il faut donc connaître le nombre de clusters à trouver, sans quoi la méthode se limite à l'étape EXPLORE. La seconde étape (cf. Algorithme 18) sélectionne des points aléatoirement et propose des requêtes en commençant par le voisinage le plus proche. Si $K - 1$ requêtes sont faites sans produire de contrainte ML, on attribue automatiquement le point au dernier voisinage puisque cette étape part du principe que tous les voisinages sont connus. La sélection aléatoire peut sembler naïve mais les auteurs montrent que cela permet de réduire le nombre de requêtes moyen à faire pour déterminer le voisinage d'un point par rapport à EXPLORE, passant de $K \binom{K}{2}$ à environ $K^2 \log K$ requêtes nécessaires pour trouver un point de chaque cluster.

Notons que FFQS est une méthode hybride : EXPLORE assure la représentativité des k clusters à retrouver, tandis que CONSOLIDATE se focalise sur l'informativité.

Algorithme 17 EXPLORE

Entrée: jeu de données $\mathcal{X}$, nombre de clusters k , budget de requêtes b .

Sortie: squelette constitué de k voisinages au plus.

```

1:  $q \leftarrow 0$  ▷ Nombre de requêtes
2:  $\mathcal{N} \leftarrow \{\}$  ▷ Voisinages
3: Choisir le premier point  $x_{init}$  aléatoirement
4:  $\mathcal{N} \leftarrow \{\{x_{init}\}\}$ 
5: tant que  $|\mathcal{N}| < k \wedge q < b$  faire ▷ Fin si  $k$  voisinages trouvés ou budget épuisé
6:    $x^* \leftarrow$  point le plus éloigné de  $\mathcal{N}$ 
7:   pour chaque voisinage  $N$  de  $\mathcal{N}$  faire
8:      $A \leftarrow$  requête( $x^*, y$ ),  $y \in N$ 
9:      $q \leftarrow q + 1$ 
10:    si  $A = ML(x^*, y)$  alors
11:       $N \leftarrow N \cup \{x^*\}$  ▷ Ajout à un voisinage existant
12:    si aucun ML obtenu alors
13:       $\mathcal{N} \leftarrow \mathcal{N} \cup \{\{x^*\}\}$  ▷ Nouveau voisinage
14: retourner  $\mathcal{N}$ 

```

Comme décrit précédemment, il est trivial de dériver les contraintes ML/CL obtenues à partir du squelette.

La phase CONSOLIDATE sera par la suite améliorée dans l'algorithme MINMAX (ou MMFFQS, MIN-MAX FURTHEST-FIRST QUERY SELECTION) en supplantant l'approche aléatoire par un problème d'optimisation dans le choix du point à présenter ([Mallapragada et al., 2008](#)).

A partir d'une matrice de similarité $\mathcal{M}_s$ de dimensions $n \times n$ calculée avec une fonction noyau gaussien, on estime l'incertitude sur l'affectation d'un point à un cluster via sa similarité avec les voisinages de $\mathcal{N}$. De là, le point le plus incertain x^* est celui qui minimise la plus grande similarité entre x^* et $\mathcal{N}$:

4.4. TRAVAUX ANTÉRIEURS EN CLUSTERING ACTIF

Algorithme 18 CONSOLIDATE

Entrée: jeu de données $\mathcal{X}$, nombre de clusters K , budget de requêtes b , voisinages $\mathcal{N}$.

Sortie: voisinages $\mathcal{N}$ mis à jour.

```

1:  $q \leftarrow 0$  ▷ Nombre de requêtes
2: calculer les centroïdes  $\mu$  de chaque voisinage
3: tant que  $q < b$  faire ▷ Fin si budget épuisé
4:   choisir  $x^*$  au hasard parmi  $\mathcal{X} \setminus \bigcup_{N \in \mathcal{N}} N$ 
5:    $cl \leftarrow 0$  ▷ Nombre de requêtes pour  $x^*$ 
6:   pour chaque voisinage  $N \in \mathcal{N}$  dans l'ordre croissant de  $\|x^* - \mu_N\|$  faire
7:     si  $cl = K - 1$  alors
8:        $N \leftarrow N \cup \{x^*\}$  ▷ Ajout au dernier voisinage sans requête
9:       break
10:     $cl \leftarrow cl + 1$ 
11:     $A \leftarrow \text{requête}(x^*, y), y \in N$ 
12:    si  $A = ML(x^*, y)$  alors
13:       $N \leftarrow N \cup \{x^*\}$  ▷ Ajout à un voisinage existant
14:   $q \leftarrow q + cl$ 
15: retourner  $\mathcal{N}$ 

```

$$x^* = \arg \min_{x_i \in \mathcal{X} \setminus \bigcup_{N \in \mathcal{N}} N} \max_{x_j \in \bigcup_{N \in \mathcal{N}} N} \mathcal{M}_s(x_i, x_j) \quad (4.7)$$

L'algorithme 19 décrit la version de CONSOLIDATE utilisée dans MINMAX, avec les modifications signalées en [bleu](#).

Algorithme 19 MINMAX-CONSOLIDATE

Entrée: jeu de données $\mathcal{X}$, nombre de clusters K , budget de requêtes b , voisinages $\mathcal{N}$.

Sortie: voisinages $\mathcal{N}$ mis à jour.

```

1:  $q \leftarrow 0$  ▷ Nombre de requêtes
2:  $\mathcal{M}^s \leftarrow e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}, \forall i, j \in \mathcal{X}$  ▷ Matrice de similarité
3: tant que  $q < b$  faire ▷ Fin si budget épuisé
4:    $x^* \leftarrow \arg \min_{x_i \in \mathcal{X} \setminus \mathcal{N}} \max_{x_j \in \mathcal{N}} \mathcal{M}_s(x_i, x_j)$  ▷ Équation 4.7
5:    $cl \leftarrow 0$  ▷ Nombre de requêtes pour  $x^*$ 
6:   pour chaque voisinage  $N$  de  $\mathcal{N}$  dans l'ordre décroissant de  $\max_{x_N \in N} \mathcal{M}_s(x^*, x_N)$  faire
7:     si  $cl = K - 1$  alors
8:        $N \leftarrow N \cup \{x^*\}$  ▷ Ajout au dernier voisinage sans requête
9:       break
10:     $cl \leftarrow cl + 1$ 
11:     $A \leftarrow \text{requête}(x^*, y), y \in N$ 
12:    si  $A = ML(x^*, y)$  alors
13:       $N \leftarrow N \cup \{x^*\}$  ▷ Ajout à un voisinage existant
14:   $q \leftarrow q + cl$ 
15: retourner  $\mathcal{N}$ 

```

Il faut noter que MINMAX est plus lent à exécuter que FFQS, avec une différence de

deux à trois ordres de grandeur en moyenne. De plus, la complexité de MINMAX dépend de la taille des voisinages, ce qui n'est pas le cas de FFQS. En conséquence, le temps de calcul de MINMAX augmente au fil des requêtes dans un cadre incrémental où les voisinages sont conservés entre les itérations.

Plus tard, dans [Xiong et al. \(2014\)](#), un algorithme appelé NPU (*Normalized Pointwise Uncertainty*) alterne clustering sous contraintes et construction de voisinages avec une estimation de l'incertitude basée sur l'entropie de Shannon ([Shannon, 1948](#)). Il prend aussi en compte le nombre de requêtes à faire avant de trouver le voisinage d'un point.

De façon analogue à MINMAX, l'instance la plus informative est calculée à chaque itération. La partition obtenue à l'étape de clustering sous contraintes tient le rôle d'étiquettes pour entraîner une forêt aléatoire. Une fois la forêt apprise, une matrice de similarité $\mathcal{M}_s$ de dimensions $n \times n$ comptabilise pour chaque paire de points le nombre de fois où ils atteignent la même feuille d'un arbre, normalisée par la taille de la forêt. On estime ensuite la probabilité d'appartenance d'un point x quelconque à un voisinage N donné via la similarité moyenne des points de N à x , normalisé par l'ensemble des points de $\mathcal{N}$:

$$P(x \in N) = \frac{\frac{1}{|N|} \sum_{x_j \in N} \mathcal{M}_s(x, x_j)}{\sum_{p=1}^{|\mathcal{N}|} \frac{1}{|N_p|} \sum_{x_j \in N_p} \mathcal{M}_s(x, x_j)} \quad (4.8)$$

Cela produit pour chaque point de $\mathcal{X}$, une distribution de probabilité d'appartenance aux voisinages dont on peut calculer l'entropie de Shannon :

$$H(\mathcal{N} \mid x) = - \sum_{i=1}^{|\mathcal{N}|} P(x \in N_i) \log_2 P(x \in N_i) \quad (4.9)$$

Ce critère d'informativité est combiné à l'espérance du nombre de requêtes à faire avant de trouver le voisinage d'un point, en exploitant encore les probabilités de l'Équation 4.8 :

$$E(\text{requête}(x)) = \sum_{i=1}^{|\mathcal{N}|} i * P(x \in N_i) \quad (4.10)$$

En supposant que les voisinages sont rangés par ordre décroissant de leur probabilité, $H(\mathcal{N} \mid x)$ et $E(\text{requête}(x))$ sont maximales pour une distribution de probabilités équiprobable. Or on cherche à maximiser l'entropie pour l'informativité, et à minimiser l'espérance pour le coût en requêtes. Le choix de x^* se base donc sur le ratio entre ces critères :

$$x^* = \arg \max_{x \notin \mathcal{N}} \frac{H(\mathcal{N} \mid x)}{E(\text{requête}(x))} \quad (4.11)$$

Plus ce ratio est élevé, plus le gain d'information est grand et moins on a de requêtes à présenter.

Si l'on met de côté l'étape de clustering sous contraintes, NPU a une complexité temporelle en $O(N_A n \log n + nb)$, avec N_A le nombre d'arbres de la forêt aléatoire ; le temps d'exécution est du même ordre de grandeur que MINMAX mais permet de converger plus vite vers la partition désirée. NPU est une méthode composite, mais il est possible de la transformer en méthode agnostique en enlevant l'étape de clustering sous contraintes et

Algorithme 20 NPU

Entrée: jeu de données $\mathcal{X}$, budget de requêtes b .

Sortie: partition $\mathcal{P}$.

```

1:  $q \leftarrow 0$  ▷ Nombre de requêtes
2:  $\mathcal{C} \leftarrow \emptyset$  ▷ Contraintes ML/CL
3:  $\mathcal{N} \leftarrow \{\}$  ▷ Voisinages
4: Choisir le premier point  $x_{init}$  aléatoirement
5:  $\mathcal{N} \leftarrow \{\{x_{init}\}\}$ 
6: tant que  $q < b$  faire ▷ Fin si budget épuisé
7:    $\mathcal{P} \leftarrow \text{CLUSTERINGSOUSCONTRAINTES}(\mathcal{X}, \mathcal{C})$ 
8:    $x^* \leftarrow \text{MOSTINFORMATIVE}(\mathcal{X}, \mathcal{P}, \mathcal{N})$ 
9:   pour chaque  $N \in \mathcal{N}$  dans l'ordre décroissant de  $P(x^* \in N)$  faire
10:    si  $q < b$  alors
11:       $A \leftarrow \text{requête}(x^*, y), y \in N$ 
12:       $q \leftarrow q + 1$ 
13:    sinon
14:      break
15:    si  $A = ML(x^*, y)$  alors
16:       $\mathcal{C} \leftarrow \mathcal{C} \cup ML(x^*, y)$ 
17:       $N \leftarrow N \cup \{x^*\}$  ▷ Ajout à un voisinage existant
18:      break
19:    sinon
20:       $\mathcal{C} \leftarrow \mathcal{C} \cup CL(x^*, y)$ 
21:    si aucun ML obtenu alors
22:       $\mathcal{N} \leftarrow \mathcal{N} \cup \{\{x^*\}\}$  ▷ Nouveau voisinage
23: retourner  $\text{CLUSTERINGSOUSCONTRAINTES}(\mathcal{X}, \mathcal{C})$ 

```

Algorithme 21 MOSTINFORMATIVE

Entrée: jeu de données $\mathcal{X}$, partition $\mathcal{P}$, voisinages $\mathcal{N}$.

Sortie: point le plus informatif x^* .

```

1: Apprendre une forêt aléatoire  $RF$  contenant  $N_A$  arbres sur  $\mathcal{P}$  ▷  $N_A = 50$  dans l'article
2: Calculer  $\mathcal{M}_s$  tel que  $\mathcal{M}_s(i, j)$  est le nombre de feuilles de  $RF$  où  $i$  et  $j$  sont ensemble
   normalisé par le nombre d'arbres de  $RF$ 
3: pour chaque  $x \in \mathcal{U} = \mathcal{X} \setminus \mathcal{N}$  faire
4:   pour  $i$  allant de 1 à  $|\mathcal{N}|$  faire
5:     Calculer  $P(x \in N_i)$  avec l'Équation 4.8
6:   Calculer  $H(\mathcal{N} \mid x)$  avec l'Équation 4.9
7:   Calculer  $E(\text{requête}(x))$  avec l'Équation 4.10
8: retourner  $\arg \max_{x \in \mathcal{U}} \frac{H(\mathcal{N} \mid x)}{E(\text{requête}(x))}$  ▷ Équation 4.11

```

FFQS (Basu et al., 2004)	$O(n)$
MinMax (Mallapragada et al., 2008)	$O(n)$
NPU (Xiong et al., 2014)	$O(n \log n)$
Cai (Cai et al., 2016)	$O(n^2)$
ALPCS (Chen et Jin, 2020)	$O(n)$

TABLE 4.1 – Complexité temporelle en moyenne des méthodes basées sur les voisinages.

en passant à la place une partition en entrée de l'algorithme (cf. Annexe B). En utilisant un algorithme de clustering sous contraintes incrémentales au lieu d'un algorithme classique (comme COPKMEANS, utilisé dans l'article), beaucoup de calculs redondants peuvent être évités⁷.

Des travaux plus récents ont cherché à améliorer le tandem EXPLORE-CONSOLIDATE en ajoutant une phase préalable de sélection d'un sous-ensemble du jeu de données sur lequel opérer, que ce soit en comptant le nombre de voisins assignés à un cluster différent (Cai et al., 2016) ou utilisant la Maximum Mean Discrepancy (MMD) entre la distribution de points étiquetés et non étiquetés pour trouver un ensemble de points qui minimise cette mesure (Atwa et Emam, 2019 ; Atwa et Li, 2014).

Par ailleurs, le principe de NPU a été adapté au clustering spectral (Xiong et al., 2016), en remplaçant le calcul de forêt aléatoire pour obtenir une matrice de similarité par la matrice d'affinité utilisée en clustering spectral et en calculant la capacité d'une contrainte à réduire l'incertitude du modèle via les gradients des vecteurs propres de la matrice d'affinité correspondant à chaque cluster.

ALPCS (Chen et Jin, 2020) intègre des mécanismes tirés d'autres méthodes pour améliorer la phase de sélection proposées dans Cai et al. (2016) et générer plus de contraintes. Il utilise notamment l'initialisation de l'algorithme AIPC (Zhong et al., 2019) qui calcule l'entropie des points avec FUZZY C-MEANS, un algorithme de clustering flou reprenant le fonctionnement de KMEANS pour calculer les probabilités d'appartenance de chaque point à chaque cluster. ALPCS emploie l'entropie de Shannon pour sélectionner un sous-ensemble des points les plus informatifs, mais le critère utilisé dans les phases Explore et Consolidate pour choisir les points à ajouter aux voisinages est la divergence de Kullback-Leibler symétrique (ou divergence de Jeffreys, Kullback et Leibler, 1951). ALPCS ne s'est pas directement comparé à NPU en termes de performance, et l'analyse de complexité temporelle montre que ALPCS trouve un point informatif en temps linéaire comme MINMAX et FFQS.

La table 4.1 résume les complexités temporelles des différentes méthodes basées sur les voisinages. La plupart des méthodes fonctionnent en temps linéaire à part NPU qui a une complexité en $O(n \log n)$ à cause du calcul de forêt aléatoire, et la méthode de Cai et al. est basée sur le calcul des k plus proches voisins et effectue un calcul de matrice de distances paire à paire de complexité quadratique⁸.

7. Cela est mentionné explicitement par les auteurs : « *Another possibility [to scale up our method] would be to develop and apply an incremental semi-supervised clustering method that updates the clustering solution incrementally when new constraints are incorporated.* »

8. La complexité de cette approche peut cependant être réduite à $O(n \log n)$ en utilisant des optimisations connues du problème des k plus proches voisins, comme l'utilisation d'un arbre k -d pour partitionner l'espace de recherche.

4.4.2 Apprentissage de contraintes triplet

Xiong et al. (2015) décrit une approche probabiliste pour identifier la requête la plus informative sur un triplet de points. Elle consiste à estimer l'incertitude sur l'affectation y_{ijk} des points du triplet (x_i, x_j, x_k) selon la réponse à la requête q_{ijk} parmi "oui", "non", ou "je ne sais pas" (nsp). Le triplet optimal est celui qui réduit le plus cette incertitude :

$$(x_i, x_j, x_k)^* = \arg \max (1 - p(q_{ijk} = nsp)) H(y_{ijk} | \mathcal{C}_T) - \sum_{r \in \{\text{oui}, \text{non}\}} p(q_{ijk} = r | \mathcal{C}_T) H(y_{ijk} | q_{ijk} = r, \mathcal{C}_T) \quad (4.12)$$

$H(y_{ijk} | \mathcal{C}_T)$ est l'entropie associée aux affectations du triplet y_{ijk} compte tenu des triplets déjà connus dans $\mathcal{C}_T$:

$$H(y_{ijk} | \mathcal{C}_T) = - \sum_{h \in \{i, j, k\}} \sum_{c=1}^K p(y_h = c | \mathcal{C}_T) \log p(y_h = c | \mathcal{C}_T) \quad (4.13)$$

$H(y_{ijk} | q_{ijk} = r, \mathcal{C}_T)$ est la valeur de cette entropie sachant en plus la réponse r à la requête q_{ijk} , calculée à l'aide de la règle de Bayes :

$$H(y_{ijk} | q_{ijk} = r, \mathcal{C}_T) = - \sum_{y_{ijk} \in [1..K]^3} \frac{p(q_{ijk} = r | y_{ijk}) p(y_{ijk} | \mathcal{C}_T)}{p(q_{ijk} = r | \mathcal{C}_T)} \times \log \frac{p(q_{ijk} = r | y_{ijk}) p(y_{ijk} | \mathcal{C}_T)}{p(q_{ijk} = r | \mathcal{C}_T)} \quad (4.14)$$

Les probabilités de chaque réponse possible pour une requête donnée dérivent des probabilités d'affectation de chaque point :

$$\begin{aligned} p(q_{ijk} = \text{oui} | \mathcal{C}_T) &= \sum_{c=1}^K p(y_i = c | \mathcal{C}_T) p(y_j = c | \mathcal{C}_T) (1 - p(y_k = c | \mathcal{C}_T)) \\ p(q_{ijk} = \text{non} | \mathcal{C}_T) &= \sum_{c=1}^K p(y_i = c | \mathcal{C}_T) p(y_k = c | \mathcal{C}_T) (1 - p(y_j = c | \mathcal{C}_T)) \\ p(q_{ijk} = \text{nsp} | \mathcal{C}_T) &= 1 - p(q_{ijk} = \text{oui} | \mathcal{C}_T) - p(q_{ijk} = \text{non} | \mathcal{C}_T) \end{aligned} \quad (4.15)$$

Enfin, la probabilité d'affectation d'un point en fonction des triplets connus $p(y_h | \mathcal{C}_T)$ est estimée en apprenant une métrique à l'aide des triplets connus et en effectuant un clustering dans cet espace. Les auteurs reprennent ensuite la méthode qu'ils ont employée pour NPU : entraîner une forêt aléatoire et utiliser les prédictions des arbres pour estimer les probabilités d'affectation.

Algorithme 22 Active Learning from Relative Comparisons

Entrée: jeu de données $\mathcal{X}$, budget de requêtes b

Sortie: ensemble de contraintes triplet $\mathcal{C}_T$;

```

1:  $\mathcal{C}_T = \emptyset$ 
2:  $q \leftarrow 0$ 
3: Choisir aléatoirement un ensemble  $R_p$  de triplets de  $\mathcal{X}$ 
4: tant que  $q < b$  faire
5:   Estimer  $p(y \mid \mathcal{C}_T)$  for chaque point de  $\mathcal{X}$ 
6:   pour chaque triplet  $r_{ijk} \in R_p$  faire
7:     Calculer  $p(q_{ijk} = r \mid \mathcal{C}_T)$  for  $r = \{ \text{où, non, nsp} \}$  ▷ Éq. 4.15
8:     Calculer  $I(y_{ijk}; q_{ijk} \mid \mathcal{C}_T) = H(y_{ijk} \mid \mathcal{C}_T) - H(y_{ijk} \mid q_{ijk} = r, \mathcal{C}_T)$  ▷ Éqs. 4.13 et 4.14
9:      $(x_i, x_j, x_k)^* = \arg \max_{(x_i, x_j, x_k) \in R_p} I(y_{ijk}; q_{ijk} \mid \mathcal{C}_T)$ 
10:     $r \leftarrow \text{requête}((x_i, x_j, x_k)^*)$ 
11:     $\mathcal{C}_T = \mathcal{C}_T \cup (x_i, x_j, x_k)^*$ ,  $R_p = R_p \setminus (x_i, x_j, x_k)^*$ .
12: retourner  $\mathcal{C}_T$ 

```

Cette méthode est destinée au *metric learning* à l'aide de triplets (Schultz et Joachims, 2003) pour la classification, mais on peut aisément utiliser les contraintes produites en clustering. Il faut toutefois noter que la méthode suppose de connaître au préalable les classes, ou le nombre de clusters.

4.4.3 Estimer le changement apporté à la partition par une nouvelle contrainte

Biswas et Jacobs (2014) proposent un algorithme de clustering hiérarchique, Active-HACC, reprenant l'algorithme de Kruskal (Kruskal, 1956) adapté pour prendre en compte les contraintes ML/CL dans la construction d'une forêt d'arbres couvrants de poids minimal (un pour chaque cluster). La composante active réside dans la sélection de la paire de points la plus susceptible d'induire un changement dans la partition existante (cf. Algorithme 23) après chaque clustering. La probabilité que chaque paire de points de $\mathcal{X}$ se trouve groupée ensemble à l'issue du clustering est estimée en faisant la moyenne de plusieurs partitions calculées par KMEANS avec des initialisations aléatoires des centroïdes. L'indice de Jaccard est utilisé pour comparer deux partitions en comptant la proportion d'affectations identiques :

$$JCC_{\mathcal{P}}(\mathcal{P}') = \frac{|\mathcal{P} \cap \mathcal{P}'|}{|\mathcal{P} \cup \mathcal{P}'|}$$

On note alors $J_{q+1}^{x_i, x_j, o}$ et $J_{q+1}^{x_i, x_j, n}$ l'indice de Jaccard entre la partition existante $\mathcal{P}$ et la partition obtenue avec un nouveau clustering hiérarchique incluant une contrainte ML ou CL (respectivement) entre les points x_i et x_j . Le changement attendu apporté par la contrainte $E_J(x_i, x_j)$ combine la probabilité que x_i et x_j soient groupés ensemble par défaut et l'indice de Jaccard d'une partition inversant la relation actuelle de x_i et x_j . S'ils sont dans le même cluster, on calcule l'indice de Jaccard de la partition où ils sont séparés et vice versa. La paire produisant le grand changement attendu est proposée à l'utilisateur

qui indique si elle est correspond à une contrainte ML ou CL, puis on calcule une nouvelle partition avec cette contrainte. L'algorithme termine quand le budget de requêtes est épuisé.

Algorithme 23 Active-HACC : Active Image Clustering with Pairwise Constraints

Entrée: jeu de données $\mathcal{X}$, nombre de clusters k , budget b

Sortie: partition $\mathcal{P}$

```

1:  $M \leftarrow \emptyset, C \leftarrow \emptyset$  ▷ Contraintes ML/CL
2:  $q \leftarrow 0$  ▷ Nombre de requêtes
3: Calculer  $S$  partitions de  $\mathcal{X}$  via KMEANS avec initialisation aléatoire ▷ S=100 dans l'article
4: Calculer les matrices binaires  $A_s$  de co-occurrence des points pour chaque partition
5: Calculer la matrice de probabilité  $\mathbf{P} = \frac{1}{S} \sum_{s=1}^S \mathbf{A}_s$ 
6: tant que  $q \leq b$  faire
7:    $\mathcal{P} \leftarrow \text{HACC}(\mathcal{X}, M, C)$ 
8:   pour chaque paire  $(x_i, x_j) \in \mathcal{X} \times \mathcal{X}$  faire
9:     Calculer le changement attendu selon la réponse utilisateur  $J_{q+1}^{x_i, x_j, o}$  et  $J_{q+1}^{x_i, x_j, n}$ 
10:    si  $x_i$  et  $x_j$  sont dans le même cluster dans  $\mathcal{P}$  alors
11:       $E_J(x_i, x_j) = |(1 - \mathbf{P}(x_i, x_j))(1 - J_{q+1}^{x_i, x_j, n})|$ 
12:    sinon
13:       $E_J(x_i, x_j) = |\mathbf{P}(x_i, x_j)(1 - J_{q+1}^{x_i, x_j, o})|$ 
14:     $(\hat{x}_i, \hat{x}_j) \leftarrow \arg \max_{x_i, x_j} E_J(x_i, x_j)$ 
15:     $A \leftarrow \text{requête}((\hat{x}_i, \hat{x}_j))$ 
16:    si  $A = ML$  alors
17:       $M \leftarrow M \cup \{(\hat{x}_i, \hat{x}_j)\}$ 
18:    sinon
19:       $C \leftarrow C \cup \{(\hat{x}_i, \hat{x}_j)\}$ 
20:     $q \leftarrow q + 1$ 
21: retourner  $\mathcal{P} \leftarrow \text{HACC}(\mathcal{X}, M, C)$ 

```

Cet algorithme est très coûteux en temps : il nécessite de simuler les changements possibles pour chacune des n^2 paires du jeu de données, menant à une complexité totale en $O(n^4 \log n)$. Les auteurs ont proposé une version optimisée nommée Fast-Active-HACC évitant de nombreux calculs redondants ou inutiles pour se rapprocher de la complexité de l'algorithme de Kruskal en $O(n^2 \log n)$.

4.4.4 Propager les contraintes avec les voisins

Certains algorithmes intègrent un mécanisme permettant de poser automatiquement de nouvelles contraintes en partant des contraintes existantes. On parle alors de *propager les contraintes* dans le jeu de données⁹. C'est le cas d'ASC (Vu et al., 2010, 2012) qui utilise la notion de *chemin fort* dans un graphe des k plus proches voisins pour propager des contraintes ML/CL. Construire un graphe des k plus proches voisins à partir d'un jeu

9. À ne pas confondre avec le mécanisme de propagation de contraintes en programmation par contraintes, ce sont deux notions distinctes.

de données consiste à considérer chaque point comme un sommet du graphe et à ajouter pour chaque sommet des arcs vers les k plus proches sommets.

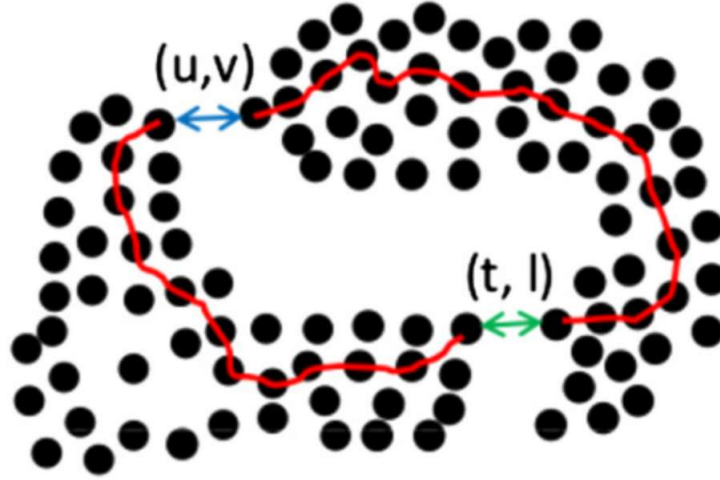


FIGURE 4.3 – Exemple de chemin fort (en rouge) permettant de propager la contrainte CL entre t et l aux points u et v . Illustration tirée de Vu et al. (2012).

Un *chemin fort* entre deux sommets du graphe existe si tous les sommets intermédiaires successifs ont au moins $\theta \leq k$ voisins en commun ou si ils sont reliés par une contrainte ML (cf. Figure 4.3). Ce mécanisme permet par exemple de propager une contrainte CL entre deux points proches à une autre paire de points, chacun étant relié à un des points de la contrainte par un chemin fort. Les requêtes présentées à l'utilisateur sont des paires de points choisies selon une mesure de densité parmi les paires de points qui ont moins de θ voisins en commun afin de cibler les régions de l'espace qui servent de transition entre les clusters. De plus, les paires de points reliés par un chemin fort sont écartées à l'avance de la sélection.

Algorithme 24 ACTIVE CONSTRAINT SELECTION (ACS)

Entrée: Jeu de données $\mathcal{X}$, nombre de voisins k , seuil θ

Sortie: Lot de contraintes L

- 1: $L \leftarrow \emptyset$
 - 2: Construire un graphe des k plus proches voisins de $\mathcal{X}$
 - 3: $C \leftarrow \{(u, v) \mid \text{poids}(u, v) < \theta\}$ ▷ Poids = nombre de voisins communs
 - 4: $C \leftarrow \text{REFINEMENT}(C, \theta)$
 - 5: **tant que** l'utilisateur n'est pas satisfait et $C \neq \emptyset$ **faire**
 - 6: choisir une paire $(u, v) \in C$
 - 7: $A \leftarrow \text{requête}((u, v))$
 - 8: **si** $A \in \{ML, CL\}$ **alors**
 - 9: $L \leftarrow L \cup \{A(u, v)\}$
 - 10: $L \leftarrow \text{PROPAGATION}(C, L, \theta)$
 - 11: $C \leftarrow \text{REFINEMENT}(C, \theta)$
 - 12: **sinon** ▷ L'utilisateur ne sait pas
 - 13: garder (u, v) en mémoire pour ne pas reposer la requête
 - 14: **retourner** L
-

La limite majeure de ASC est l'hypothèse posée pour la propagation des contraintes : avoir beaucoup de voisins en commun n'implique pas toujours d'appartenir au même cluster. En particulier, ASC risque de propager de mauvaises contraintes dans des clusters anisotropes, dont la densité varie selon les régions.

4.4.5 Ordonner les voisins de points étiquetés

Dans certains cas, il est possible d'avoir un petit nombre d'étiquettes sur les données sans pouvoir en obtenir davantage. Certaines méthodes cherchent à exploiter l'information de cet étiquetage et à la propager dans le reste des données en étant guidé par l'utilisateur. ALNO (Qian et al., 2013) propose de présenter des requêtes d'ordonnancement des voisins d'un point étiqueté pour propager cette étiquette sous forme de poids d'un graphe de connectivité.

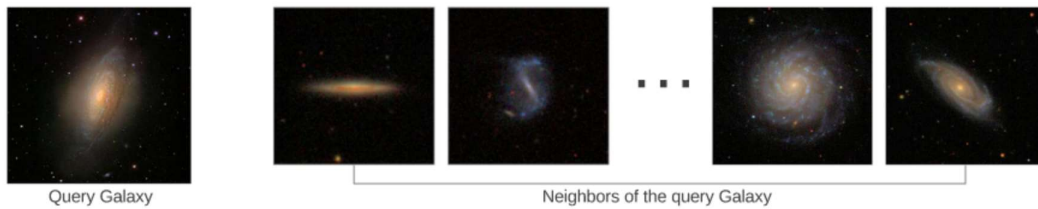


FIGURE 4.4 – Exemple de requête proposée par ALNO sur des images de galaxie. L'utilisateur doit donner un ordre sur les voisins par rapport à leur ressemblance à l'image de référence. Illustration tirée de Qian et al. (2013).

L'ensemble des voisins d'un point étiqueté sont calculés par la résolution d'un problème de couverture d'ensemble : un PLNE calcule le nombre minimal de groupes de voisins nécessaires pour couvrir l'ensemble des instances. Le PLNE est volontairement relâché pour fournir plusieurs solutions quasi-optimales. L'informativité d'un groupe donné correspond alors au nombre de solutions dans lequel il est présent. L'utilisateur va ordonner le groupe le plus informatif par rapport à sa similarité avec le point étiqueté et peut se contenter de donner un ordre partiel s'il ou elle n'est pas capable de tout ordonner.

Ces requêtes sont traduites en contraintes sur les poids du graphe reliant les points. La construction de cette matrice est effectuée via un programme quadratique qui minimise l'erreur de reconstruction des points en se basant sur leurs voisins, tout en intégrant les contraintes de type $w_{ia} \geq w_{ib}$ avec w_{ix} le poids du point a pour reconstruire i . Par transitivité, on peut retranscrire l'ordre donné par l'utilisateur. Le résultat obtenu est un graphe de connectivité dont les poids ont été influencés par l'utilisateur avec ses ordres relatifs sur des sous-ensembles de points. Même si le cas d'application le plus évident est la propagation de l'information dérivée d'étiquettes connues, ALNO peut aussi être utilisé quand aucune étiquette n'est connue.

4.4.6 Sélection séquentielle d'un ensemble de contraintes

La méthode SASC (Abin et Beigy, 2014) construit un ensemble de contraintes en les sélectionnant une par une et en cherchant à obtenir des contraintes de divers zones de

l'espace de clustering. Pour cela, les auteurs se basent sur deux hypothèses caractérisant les contraintes les plus informatives en partant d'une décomposition de l'espace en régions denses :

1. Les contraintes impliquant des points proches dans l'espace mais de régions différentes
2. Les contraintes impliquant des points éloignés dans l'espace mais appartenant à la même région

SASC utilise d'abord une méthode de détection de frontières basée sur les vecteurs de support (SVDD, [Tax et Duin, 2004](#)). Le principe est d'utiliser une fonction noyau pour atteindre un espace de représentation de plus grande dimension, de calculer une sphère englobante basée sur les données dans ce nouvel espace (un peu comme une enveloppe convexe), puis d'utiliser les points sur la sphère pour délimiter des zones dans l'espace de départ.

Elle permet d'identifier les différentes régions du jeu de données, et les vecteurs de support qui se trouvent sur ces frontières seront utilisés pour trouver des contraintes.

Les contraintes à utiliser sont sélectionnées selon plusieurs métriques : d'abord, une métrique d'utilité individuelle de la contrainte, qui représente les suppositions évoquées plus haut :

$$u_{ij}^{indA} = v_{ij}^{out} \left(1 - \frac{d_{ij}}{d_{max}}\right) \quad (4.16)$$

v_{ij}^{out} représente la part du segment reliant i et j qui se trouve hors d'une région définie lors de la détection de frontières (cf. Figure 4.5) et d_{max} la distance maximale entre deux points. Le second terme privilégie les contraintes entre des points proches. Cette métrique représente la première supposition. En inversant la pondération des termes, on obtient une métrique pour la seconde supposition :

$$u_{ij}^{indB} = (1 - v_{ij}^{out}) \frac{d_{ij}}{d_{max}} \quad (4.17)$$

La première métrique est utilisée pour évaluer les contraintes entre deux régions, tandis que la deuxième évalue les contraintes dans une même région.

En outre, la sélection d'une contrainte est suivie d'une réévaluation des contraintes restantes, de façon à choisir des contraintes éloignées de celles déjà choisies :

$$u_{ij}^{t+1} = u_{ij}^{ind} \times \frac{d_{ijC^t}}{d_{max_{C^t}}} \quad (4.18)$$

L'utilité est donc pondérée en fonction de sa proximité avec la contrainte la plus proche de l'ensemble C^t des contraintes sélectionnées. Le calcul de distance entre deux contraintes peut être effectué de multiples façons ; en l'occurrence, c'est le minimum des deux façons de sommer les distances entre les points des contraintes qui est employé : $d_{ijkl} = \min(d_{ik} + d_{jl}, d_{il} + d_{jk})$

L'ensemble de contraintes optimal est donc celui qui pondère au mieux les contraintes à chaque étape de la séquence de sélection :

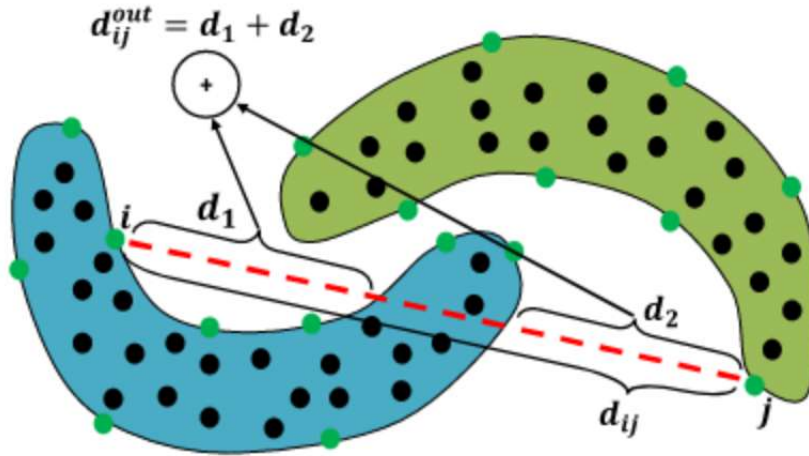


FIGURE 4.5 – $v_{ij}^{out} = \frac{d_{ij}^{out}}{d_{ij}}$ est la proportion de la distance entre i et j hors de toute région détectée. Les points utilisés comme vecteurs de support sont en vert. Illustration tirée de [Abin et Beigy \(2014\)](#).

$$\arg \max_{C^\lambda} \sum_{t=1}^{\lambda} u_{ij}^t$$

C'est un problème combinatoire difficile, auquel est préféré un algorithme glouton qui à tout instant t cherche la meilleure contrainte pour $t + 1$ au lieu de chercher une solution optimale globale. En particulier, le fonctionnement séquentiel de la recherche donne énormément d'impact au choix de la première contrainte.

L'auteur de SASC développera par la suite plusieurs algorithmes de clustering actif en reprenant les concepts clés de SASC, que nous allons rapidement passer en revue. QUFL ([Abin, 2016](#)) propose de transformer le problème de recherche de contraintes dans le jeu de données en un problème d'emplacement d'installations bien connu en recherche opérationnelle ([Wolsey et al., 1990](#)). Ce problème consiste à trouver sur quels sites ouvrir des installations pour couvrir tous les clients en minimisant le coût. Ici, les sites potentiels sont les points et les clients sont les points qui n'ont pas été désignés comme "installation". On pose ensuite des requêtes ML/CL entre les "points-installations", ce qui permet d'obtenir des contraintes bien réparties dans l'espace. RWACS ([Abin, 2018](#)) utilise des propriétés liées à la marche aléatoire sur un graphe, notamment les temps de trajet entre les sommets, afin de trouver des contraintes entre des points de régions différentes de l'espace. QECT ([Abin, 2019](#)) approfondit le principe en calculant des plongements de dimensions variées et en calculant l'utilité des contraintes selon leur propension à faire partie de l'arbre couvrant du jeu de données quelle que soit la dimension du plongement. Enfin, [Abin et Vu \(2020\)](#) reprend l'idée de chemin fort utilisée dans ASC (Section 4.4.4) pour calculer des *chaînes de densité* à partir desquelles les contraintes sont sélectionnées, ciblant notamment les bouts des chaînes.

4.4.7 Discussion

Type de requête

L'état de l'art se focalise sur les requêtes donnant lieu à des contraintes ML/CL, ce qui est cohérent avec la prépondérance des méthodes de clustering sous contraintes utilisant ces contraintes. Quelques méthodes proposent d'autres types de requêtes (Qian et al., 2013 ; Xiong et al., 2015) tandis que d'autres comme ASC complètent les requêtes en les propageant à d'autres points.

Exploitation ou exploration ?

D'une manière analogue, la plupart des méthodes de clustering actif utilise des critères visant à obtenir des contraintes informatives, aux frontières entre les clusters connus. SASC et QUFL sont des exemples de méthodes qui cherchent explicitement à obtenir des contraintes qui ne soient pas concentrées sur la même région (en s'écartant des contraintes déjà obtenues pour SASC, via le problème d'emplacement d'installation pour QUFL). Les méthodes utilisant le tandem Explore-Consolidate sont un peu plus hybrides : Explore vise à découvrir rapidement les différents voisinages correspondant aux clusters finaux, en supposant que le nombre total est connu de l'utilisateur, puis Consolidate améliore la connaissance des frontières entre les clusters. NPU possède uniquement un critère informatif mais peut à tout moment découvrir un nouveau voisinage.

Nombre de requêtes et complexité

Le nombre de requêtes est systématiquement mesuré, avec de gros écarts entre les méthodes, de quelques dizaines à des milliers de requêtes. Le design des requêtes semble avoir un grand impact sur cet aspect : les méthodes utilisant des ML/CL nécessitent souvent des centaines de requêtes au minimum pour avoir un impact significatif sur la qualité, alors que les requêtes de comparaisons relatives obtiennent des résultats en quelques dizaines de requêtes, même si ces dernières nécessitent plus d'effort de la part de l'expert(e). Cela pose la question du compromis entre le nombre et la complexité des requêtes.

Comme nous l'avons vu, les méthodes actuelles ont développé de nombreuses stratégies pour utiliser les informations du jeu de données, les contraintes passées ou un étiquetage partiel pour trouver de nouvelles contraintes. Cependant, aucune ne cherche à extrapoler ces informations pour produire une approximation concrète de ce que cherche l'utilisateur. Il nous semble que même imparfaite, une telle approximation peut se révéler utile en comparaison avec une partition existante ou même le jeu de données seul pour poser des requêtes. En outre, la complexité des méthodes les plus efficaces est trop élevée pour fonctionner sur de grands jeux de données en temps réel alors que cette condition est nécessaire lorsqu'on interagit avec un humain. La méthode de clustering actif REACT que nous proposons dans le chapitre 7 tente de répondre à ces besoins.

Approche	Contraintes	Catégorie	Stratégie	Voisinages
FFQS Basu et al. (2004)	ML/CL	agnostique	exploitation	X
MMFFQS Mallapragada et al. (2008)	ML/CL	agnostique	exploitation	X
NPU Xiong et al. (2014)	ML/CL	composite	exploitation	X
Atwa et Li (2014) Atwa et Emam (2019)	ML/CL	composite	exploitation	X
Cai et al. (2016)	ML/CL	agnostique	exploitation	X
AIPC (Zhong et al., 2019)	ML/CL	agnostique	exploitation	
ALPCS (Chen et Jin, 2020)	ML/CL	agnostique	exploitation	X
Active-HACC (Biswas et Jacobs, 2014)	ML/CL	composite	exploitation	
ALNO (Qian et al., 2013)	ordonnancement	agnostique	exploitation	
ASC Vu et al. (2010, 2012)	ML/CL	agnostique	exploitation	
SASC (Abin et Beigy, 2014)	ML/CL	agnostique	exploration	
QUFL (Abin, 2016)	ML/CL	agnostique	exploration	
RWACS (Abin, 2018)	ML/CL	agnostique	exploration	
QECT (Abin, 2019)	ML/CL	agnostique	exploitation	
Abin et Vu (2020)	ML/CL	agnostique	exploitation	
Xiong et al. (2015)	triplet	agnostique	exploitation	

TABLE 4.2 – Comparaison des approches présentées dans la Section 4.4.

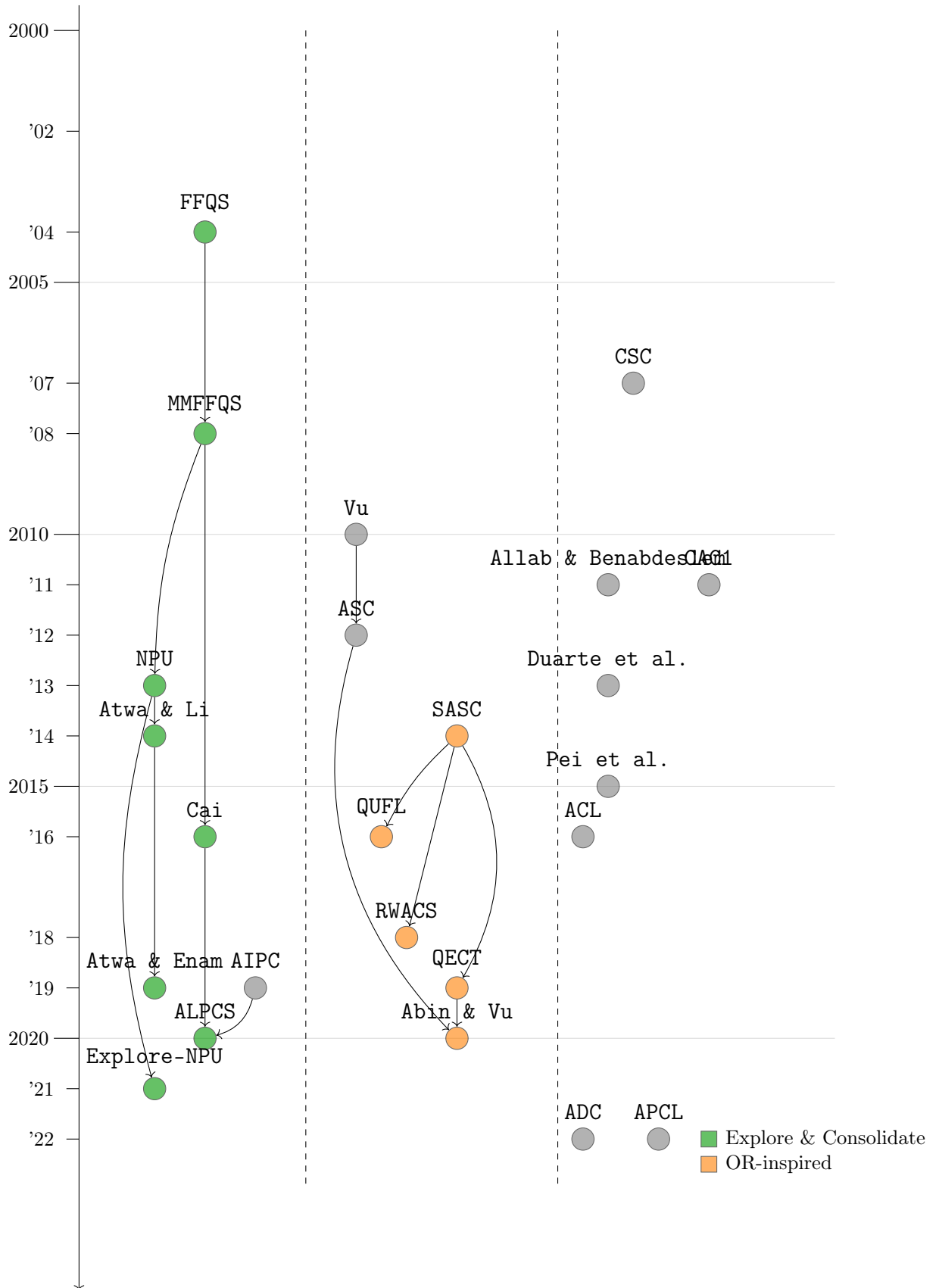


FIGURE 4.6 – Chronologie des algorithmes de clustering actif présentés.

Deuxième partie

Modification interactive de clustering

Dans cette partie, nous présentons les travaux effectués durant cette thèse.

Dans le chapitre 5, nous présentons le système de clustering sous contraintes incrémentales CoActiVe dans lequel s'insèrent les différentes contributions.

Dans le chapitre 6, nous expliquons le fonctionnement du modèle en programmation par contraintes CP-MWCM utilisé à pour réaliser la modification minimale pondérée d'une partition, en tenant compte des contraintes utilisateur.

Nous détaillons ensuite le fonctionnement de l'algorithme de clustering actif REACT construisant une approximation de la partition désirée par l'expert en se basant sur ses retours passés, afin de la confronter à la partition actuelle pour identifier des contraintes intéressantes (Chapitre 7).

Enfin, nous nous intéressons aux circonstances pouvant entraîner l'apparition de conflits entre les contraintes et présentons une mesure d'adéquation des contraintes à un modèle d'apprentissage supervisé, employée pour calculer le MCS d'adéquation minimale contenant les contraintes les plus incongrues pour éviter l'insatisfiabilité de CP-MWCM (Chapitre 8).

Chapitre 5

Un cadre de clustering sous contraintes adapté à l'utilisateur

Comme le clustering est par essence un processus exploratoire, toute partition trouvée dans un cadre interactif est sujette à des retours utilisateur basées sur des connaissances du domaine. La partition devrait donc être modifiée pour être en accord avec les retours, cependant les modifications apportées ne doivent pas donner une partition trop différente de la précédente pour ne pas désorienter l'expert(e). Il doit également être possible de suggérer des contraintes à valider ou rejeter pour aider l'expert(e) à comprendre les résultats produits tout en lui permettant de donner ses propres retours.

Comme le montre l'état de l'art réalisé dans la section précédente, les méthodes développées dans le domaine du clustering sous contraintes incrémentales se sont focalisées sur les aspects post-hoc (visualisation de la partition, interprétation des résultats) plutôt que sur le processus en lui-même. De plus, ces méthodes sont surtout testées sur des banques d'images, notamment de visages humains. On peut se demander si les résultats obtenus dans un tel cadre expérimental sont généralisables à des domaines d'application où les données sont plus abstraites et où le niveau d'expertise requis est plus élevé ; autrement dit, à des applications où l'interaction est primordiale pour donner des résultats satisfaisants. De plus, le maintien de la continuité entre les résultats, qui nous semble être une condition nécessaire pour assurer la lisibilité du processus pour l'utilisateur, est un aspect qui a été mentionné dans quelques travaux comme [Davidson et al. \(2007\)](#), mais n'a jamais été au centre de ces derniers.

L'ensemble de nos travaux s'insèrent dans un cadre de clustering sous contraintes incrémentales intégrant le clustering actif appelé CoActive¹ que nous avons mis au point. Ce cadre est schématisé dans la Figure 5.1.

Le but est d'aider un utilisateur à affiner une partition quelconque via sa connaissance du domaine et/ou les renseignements obtenus pendant le processus. Les données **ne changent pas** ; la démarche est incrémentale en fonction des contraintes. Les modalités d'application de ce cadre à des flux de données n'ont pas été étudiées dans notre travail de recherche.

Le processus est une boucle lancée par une partition initiale calculée par un algorithme de clustering quelconque ou obtenue par tout autre moyen. CoActive montre cette partition à l'utilisateur pour avoir un avis général (résultat satisfaisant ou pas). Le cas échéant,

1. Le nom initial donné dans [Beauchamp et al. \(2023\)](#) est IAC.

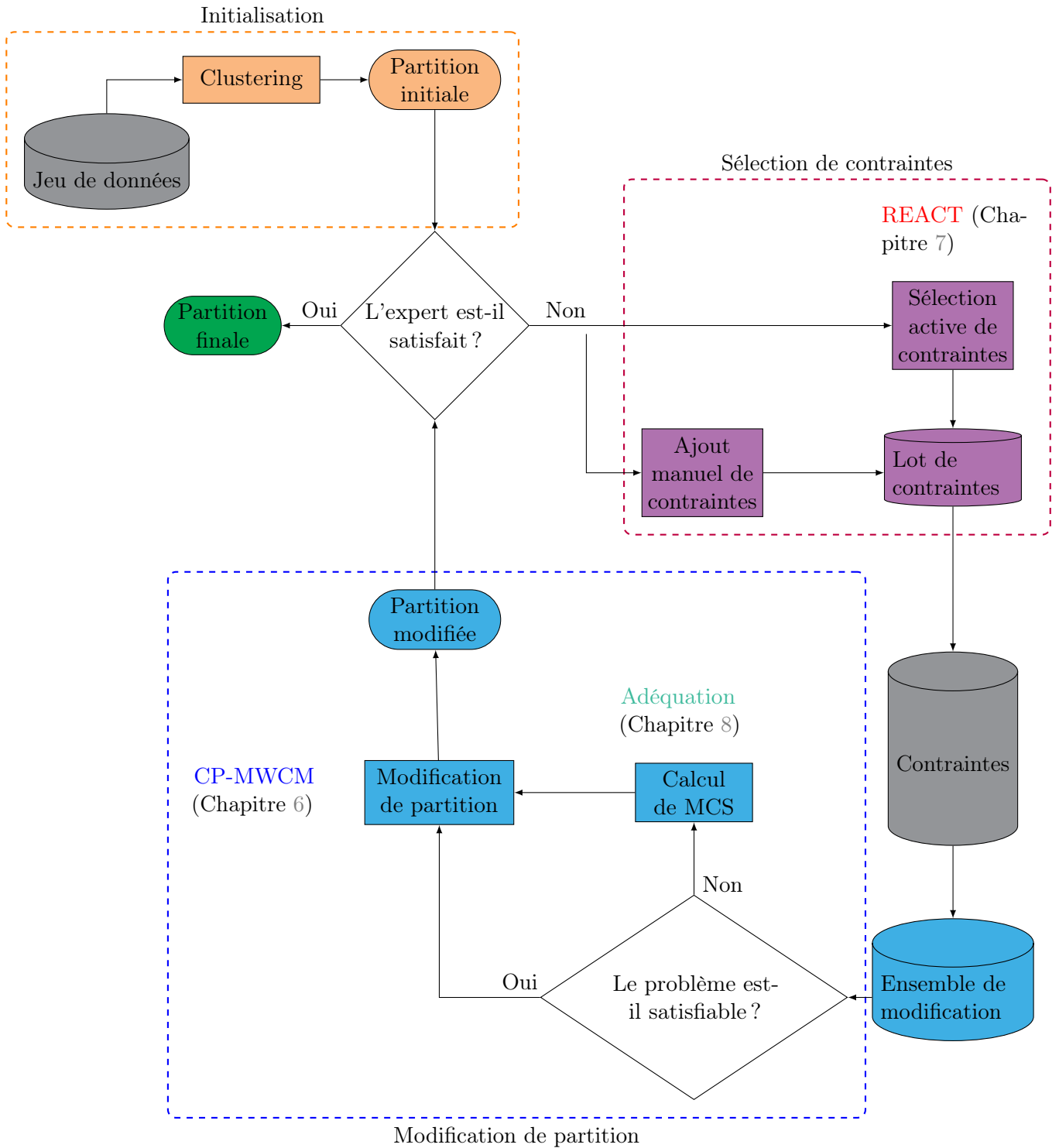


FIGURE 5.1 – Schéma de CoActive.

il ou elle peut fournir des retours plus spécifiques en donnant directement un lot de contraintes et/ou en les obtenant en passant par une méthode de clustering actif. On construit ensuite l'ensemble de contraintes utilisé pour modifier la partition en combinant le lot de contraintes obtenu dans d'anciennes itérations. L'étape de modification met à jour

la partition d'après cet ensemble de contraintes, en pouvant généraliser les changement à des points non contraints proches des points contraints. La sortie est une nouvelle partition satisfaisant les contraintes autant que possible tout en préservant la stabilité, c'est-à-dire en restant similaire à la partition antérieure. On demande alors à nouveau un avis général sur le résultat. Ce processus est répété jusqu'à ce que l'utilisateur juge le résultat satisfaisant. Un exemple sur un jeu de données synthétique dans la Figure 5.2 montre l'évolution d'une partition au cours du temps avec CoActiVe.

CoActiVe fonctionne donc en s'appuyant sur deux sources distinctes de contraintes et les traite comme des parties d'un processus dialectique convergent. D'une part, l'expert(e) indique ses objectifs et sa connaissance du domaine par ses contraintes. D'autre part, le clustering actif base ses propositions sur une analyse de la structure des données et de la partition. La boucle de CoActiVe tend à aligner les attentes de l'expert(e) et l'inférence automatisée au cours du temps : l'utilisateur(e) précise ses préférences grâce aux résultats produits par la modification, tandis que le clustering actif met à jour ses suggestions en fonction des retours utilisateur connus. Par cette dialectique humain-machine, les différences initiales entre les sources disparaissent, laissant place à un résultat final mélangeant connaissances expert et raisonnement sur les données.

Algorithme 25 CoActiVe

Entrée: Jeu de données $\mathcal{X}$, budget de requêtes par itération b

Sortie: Partition affinée $\mathcal{P}$

```

1:  $t \leftarrow 0$ 
2:  $\mathcal{C} \leftarrow \emptyset$ 
3:  $\mathcal{P} \leftarrow \text{clusteringInitial}(\mathcal{X})$ 
4:  $\mathcal{U}, sat \leftarrow \text{retourUtilisateur}(\mathcal{P})$  ▷ Réaction initiale
5: tant que  $sat == \text{Faux}$  faire
6:   si  $\mathcal{U} \neq \emptyset$  alors
7:      $L_t \leftarrow \mathcal{U}$  ▷ Contraintes données manuellement
8:   sinon
9:      $L_t \leftarrow \text{selectionActive}(\mathcal{X}, \mathcal{P}, b)$  ▷ Chapitre 7
10:   $\mathcal{C} \leftarrow \mathcal{C} \cup L_t$ 
11:   $\mathcal{C}_t \leftarrow \text{ensContraintes}(\mathcal{C})$ 
12:  si  $\text{UNSAT}(\mathcal{C}_t)$  alors
13:     $MCS \leftarrow \text{resoudreConflit}(\mathcal{C}_t)$  ▷ Chapitre 8
14:     $\mathcal{C}_t \leftarrow \mathcal{C}_t \setminus MCS$ 
15:   $\mathcal{P} \leftarrow \text{modifierClustering}(\mathcal{X}, \mathcal{P}, \mathcal{C}_t)$  ▷ Chapitre 6
16:   $\mathcal{U}, sat \leftarrow \text{retourUtilisateur}(\mathcal{P})$ 
17:   $t++$ 
18: retourner  $\mathcal{P}$ 

```

L'algorithme 25 donne la marche générale du système. Passons en revue les différentes étapes :

Initialisation de la partition *clusteringInitial* représente tout algorithme capable de produire une partition des données. Si le résultat est un dendrogramme, un niveau de celui-ci doit être choisi comme entrée du système ;

Réaction de l'expert(e) *retourUtilisateur* est l'étape d'interaction avec l'humain qui donne son avis global (sa satisfaction avec le résultat actuel, encodé dans la variable booléenne *sat*) et éventuellement à un niveau plus spécifique en donnant un ensemble de contraintes $\mathcal{U}$ pour améliorer la partition. Le système s'arrête si l'utilisateur considère le résultat comme satisfaisant ;

Sélection active de contraintes À défaut - ou en sus - des contraintes données manuellement, on peut avoir recours au clustering actif pour obtenir un lot de contraintes. La fonction *selectionActive* incarne toute méthode qui, prenant en compte le jeu de données et la partition, produit un ensemble de contraintes ;

Construction de l'ensemble de contraintes pour modification À partir du lot de contraintes obtenu et des contraintes des itérations antérieures, *ensContraintes* construit l'ensemble des contraintes utilisateur qui serviront à modifier la partition ;

Gestion des conflits Si toutes les contraintes ne peuvent pas être satisfaites en même temps, *resoudreConflit* détecte la source du problème et identifie un MCS à retirer. Les contraintes du MCS feront toujours partie de $\mathcal{C}$ mais ne seront plus sélectionnées pour construire un ensemble de modification $\mathcal{C}_t$;

Modification de la partition *modifierClustering* prend en entrée la partition existante et la modifie au besoin pour satisfaire l'ensemble de contraintes fourni. On fait réagir l'expert(e) sur ce nouveau résultat.

Pour un fonctionnement optimal, ces méthodes doivent être capables de respecter les contraintes évoquées dans la section précédente pour que le système soit utile, *i.e.* *selectionActive* doit tenir compte du temps nécessaire à la sélection des contraintes, et *modifierClustering* devrait pouvoir fournir une garantie sur la similarité entre les partitions pré- et post-modification.

Terminaison de CoActive . Comme CoActive est un système interactif, la terminaison dépend de l'utilisateur. En supposant qu'il ou elle mettra fin à l'algorithme si aucune nouvelle contrainte ne peut être ajoutée, nous allons démontrer que cette situation est forcément atteinte par CoActive. A chaque boucle de CoActive, de nouvelles contraintes sont ajoutées manuellement ou par le biais du clustering actif. Il est possible qu'aucune contrainte ne soit ajoutée dans le cas où l'utilisateur n'ajoute aucune contrainte manuellement et/ou donne une réponse indifférente aux requêtes lors de l'étape de clustering actif. La taille de $\mathcal{C}$ évolue donc de façon monotone croissante. Or l'espace des contraintes possibles est fini (cf. 3.1) et une même contrainte ne peut pas être ajoutée deux fois à $\mathcal{C}$ par définition d'un ensemble. De plus, aucune requête n'est posée deux fois sur le même point ou ensemble de points, de sorte que les réponses indifférentes aux requêtes participent à réduire l'espace des contraintes restantes. Il existe donc une itération $i \in \mathbb{N}$ à partir de laquelle $|\mathcal{C}|$ ne peut plus augmenter car l'espace des contraintes possibles aura été entièrement exploré. Par hypothèse, l'utilisateur termine CoActive à l'itération i . $\square$

Dans la section 6.1, nous présentons un modèle en PPC pour le problème de modification minimale pondérée d'une partition. Il peut être utilisé pour réaliser *modifierClustering*. Des alternatives présentées auparavant (Kuo et al., 2017 ; Nghiem et al., 2020) peuvent être utilisées en complément pour traiter les contraintes que notre modèle ne gère pas.

En ce qui concerne la sélection active, nous avons utilisé dans nos expériences des algorithmes basés sur les voisinages et nous en avons développé un nouveau détaillé dans

le chapitre 7. Ce choix s’explique par la synergie entre le fonctionnement des voisinages et l’incrémentalité : chaque itération apporte de nouvelles contraintes avec lesquels les voisinages sont renforcés, ce qui permet de trouver d’autres contraintes à la prochaine itération.

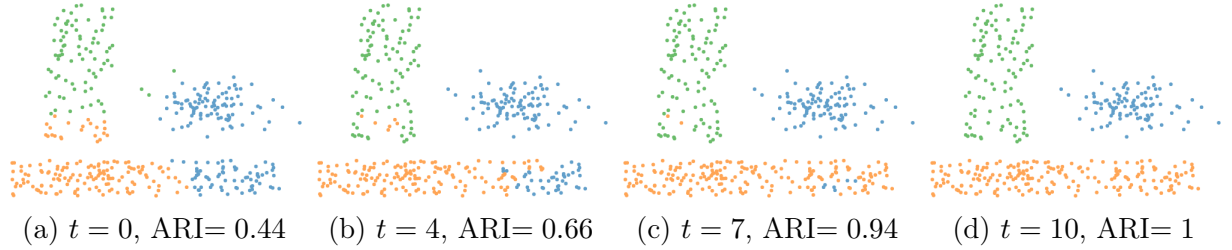


FIGURE 5.2 – Illustration de CoActive sur 10 itérations sur le jeu de données `1sun`, à partir d’une partition de KMEANS (Fig. 5.2a). Les figures successives montrent l’évolution de la partition après t itérations. L’ARI (cf. Définition 2.3) de la partition par rapport à la vérité terrain est reporté sur chaque figure.

Chapitre 6

Un modèle en PPC pour la modification minimale pondérée de partition

Table des matières

6.1	Modification minimale pondérée de partition (CP-MWCM) .	91
6.1.1	Fonction objectif et ancres	92
6.1.2	Généraliser les modifications	94
6.1.3	Formulation du modèle en PPC	95
6.1.4	Construction de l'ensemble de contraintes	96
6.1.5	Relâchement des contraintes	98
6.2	Expérimentations	99
6.2.1	Protocole expérimental	99
6.2.2	Paramétrage de CP-MWCM	100
6.2.3	Impact du nombre et du type de contraintes utilisateur sur le temps de calcul	101
6.2.4	Relâchement des contraintes	103
6.2.5	Comparaison avec l'état de l'art	104
6.2.6	Gestion des contraintes	106
6.2.7	Cas d'étude : coupes de bois	106

6.1 Modification minimale pondérée de partition (**CP-MWCM**)

Concrètement, la modification d'une partition existante consiste à changer l'affectation d'une partie des points soumis à des contraintes de façon à ce que ces dernières soit satisfaites, sans recalculer l'ensemble de la partition en entrée. On parle de **modification minimale** quand on cherche à trouver la meilleure partition modifiée selon un critère d'optimisation qui va limiter l'impact des modifications apportées.

Problème (Modification minimale pondérée de partition (MWCM)). Soit une partition $\mathcal{P}$ et un ensemble de contraintes utilisateur $\mathcal{C}$. On cherche à obtenir une partition modifiée $\mathcal{P}'$

satisfaisant $\mathcal{C}$ tout en étant similaire à $\mathcal{P}$. Nous pouvons formuler cela comme un problème d'optimisation :

$$\begin{aligned} \arg \min \quad & \text{wdiff}(\mathcal{P}, \mathcal{P}') \\ \text{s.t.} \quad & \mathcal{P}' \text{ satisfait } \mathcal{C} \end{aligned} \tag{6.1}$$

où $\text{wdiff}(\mathcal{P}, \mathcal{P}')$ est une fonction estimant la différence entre deux partitions comme suit :

$$\sum_{i=1}^n \mathbb{I}(\mathcal{P}[i] \neq \mathcal{P}'[i]) W_i \tag{6.2}$$

avec W_i le poids d'une modification et $\mathbb{I}$ la fonction indicatrice qui retourne 1 si l'expression en argument est vraie et 0 sinon.

Il s'agit d'une généralisation du problème de modification minimale de clustering présenté dans [Kuo et al. \(2017\)](#) où la fonction objectif prend en compte la qualité de chaque modification plutôt que d'estimer de façon globale la différence entre l'entrée et la sortie (en comptant le nombre de modifications ou en utilisant une mesure de validité externe).

Nous formulons le problème de modification minimale pondérée de clustering (MWCM¹) dans un problème d'optimisation sous contraintes. L'algorithme 26 donne un aperçu des étapes de pré- et post-traitement encadrant la résolution du COP avec la programmation par contraintes. Le reste du chapitre justifie et détaille les différentes étapes de cet algorithme que nous appelons CP-MWCM.

CALCULERREPRESENTANTS (Section 6.1.1) est une étape préparatoire qui sélectionne un sous-ensemble de points de chaque cluster pour le représenter. CALCULERPOINTSCOP (Section 6.1.2) produit le sous-ensemble des points soumis à des contraintes qui seront liés à des variables dans le COP. Ils peuvent éventuellement représenter des groupes des points de données. MATRICEDISTANCES peut alors, à l'aide des données précédentes calculer les distances de chaque point contraint au représentant le plus proche du cluster. POINTS-CONSTRAINTS donne les affectations de chaque point contraint dans la partition courante. RESOUDRECOP cherche les modifications à effectuer à l'aide du COP en section 6.1.3. Une fois trouvées, elles sont appliquées à la partition courante avec APPLIQUERMODIFICATIONS qui modifie les affectations de tous les points concernés.

6.1.1 Fonction objectif et ancres

Un exemple trivial de critère objectif est la minimisation du nombre de points réaffectés entre une partition obtenue à une itération t $\mathcal{P}_t$ et la partition modifiée à l'itération suivante $\mathcal{P}_{t+1}$ ([Kuo et al., 2017](#)) :

$$\arg \min \sum_{j=0}^N \mathbb{I}(\mathcal{P}_t[j] \neq \mathcal{P}_{t+1}[j]) \tag{6.3}$$

Cependant ce critère considère toutes les solutions ayant le même nombre de modifications comme étant équivalentes, sans considérer les modifications apportées aux clusters.

1. *Minimal Weighted Clustering Modification* en anglais.

Algorithme 26 CP-MWCM

Entrée: jeu de données $\mathcal{X}$, partition $\mathcal{P}_t$, ensemble de contraintes $\mathcal{C}_t$, taux de génération d'ancres α , taux de super-instances β , taux de satisfaction des contraintes δ

Sortie: partition modifiée $\mathcal{P}_{t+1}$

- 1: $ancres \leftarrow \text{CALCULERREPRESENTANTS}(\mathcal{X}, \mathcal{P}_t, \alpha)$ ▷ Section 6.1.1
 - 2: $X \leftarrow \text{CALCULERPOINTSCOP}(\mathcal{X}, \mathcal{P}_t, \mathcal{C}_t, \beta)$ ▷ Points utilisés dans le COP (Section 6.1.2)
 - 3: $\mathcal{D} \leftarrow \text{MATRICEDISTANCES}(X, ancres)$ ▷ Section 6.1.1
 - 4: $p \leftarrow \text{POINTSCONSTRAINTS}(X, \mathcal{P}_t)$ ▷ Appartenance des points contraints
 - 5: $mods \leftarrow \text{RESOUDRECOP}(\mathcal{D}, p, \mathcal{C}_t, \delta)$ ▷ Résout le COP en section 6.1.3
 - 6: $\mathcal{P}_{t+1} \leftarrow \text{APPLIQUERMODIFICATIONS}(mods, \mathcal{P}_t)$ ▷ Mise à jour de la partition et généralisation des modifications
 - 7: **retourner** $\mathcal{P}_{t+1}$
-

Intuitivement, on peut poser comme principe qu'une solution où deux points sont affectés à un cluster proche est meilleure - en termes de continuité des résultats et de facilité d'interprétation pour l'utilisateur - qu'une solution où un seul point est placé dans un cluster lointain. Pourtant l'équation 6.3 va privilégier la seconde solution.

Pour prendre en compte la structure des clusters, il est intéressant d'intégrer la notion de distance dans le critère. Par exemple en utilisant la distance au centre de chaque cluster :

$$\arg \min \sum_{j=0}^N \mathbb{I}(\mathcal{P}_t[j] \neq \mathcal{P}_{t+1}[j]) d(\mathcal{D}[j], \text{centre}(\mathcal{D}[j], \mathcal{P}_{t+1}[j])) \quad (6.4)$$

En formulant la modification minimale comme celle qui va causer la réaffectation d'un point au cluster dont le centre est le plus proche, on guide le processus d'optimisation de façon cohérente et les choix du système sont facilement explicables. Comme souvent lorsqu'on utilise des centroïdes, la limite de ce critère est la tendance à favoriser la formation de clusters sphériques. Un moyen d'outrepasser cette limite est l'utilisation d'ancres (Guilbert et al., 2022), des représentants locaux d'une sous-région d'un cluster. On calcule alors des distances par rapport à l'ancre du nouveau cluster qui est la plus proche d'un point modifié :

$$\arg \min \sum_{j=0}^N \mathbb{I}(\mathcal{P}_t[j] \neq \mathcal{P}_{t+1}[j]) d(\mathcal{D}[j], \text{closestAnchor}(\mathcal{D}[j], \mathcal{P}_{t+1}[j])) \quad (6.5)$$

où $\mathcal{D}$ est une matrice de dimensions $n \times k$ telle que $\mathcal{D}[i, c]$ est la distance du point i à l'ancre du cluster c la plus proche. L'équation 6.5 minimise donc les distances cumulées des réaffectations. C'est cette formule que nous utiliserons comme fonction objectif de notre modèle. Les ancres sont calculées en divisant chaque cluster la partition existante à l'aide du clustering hiérarchique *single-link*. Cet algorithme tend à produire des clusters sphériques, ou un énorme cluster contenant la majorité des points entouré de plusieurs clusters de quelques points. Dans les deux cas, les médoïdes de ces sous-clusters (qui deviennent les ancres) permettent de représenter les contours du cluster. Un paramètre α définit la proportion de points par cluster (en pourcentage) qui seront utilisés comme ancres. Un taux de 0% signifie que seuls les médoïdes des clusters sont utilisés, tandis qu'à un taux de 100%, tous les points sont des ancres (cf. Figure 6.1).

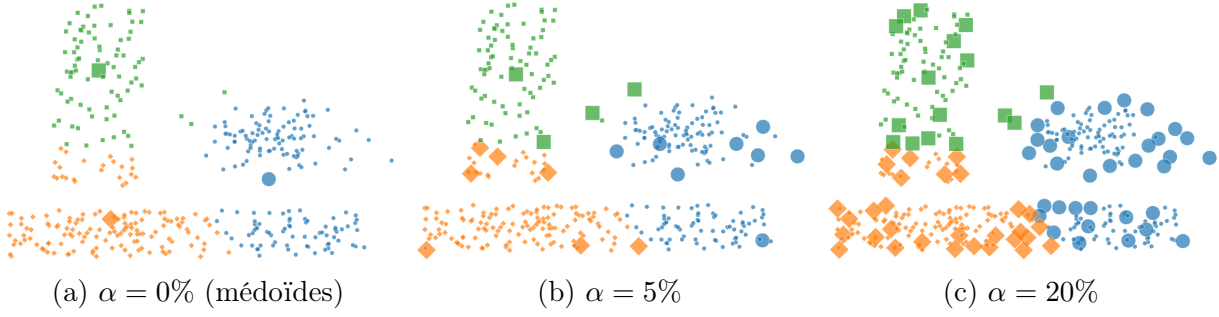


FIGURE 6.1 – Position des ancres pour différentes valeurs de α calculées à partir d’une partition de KMEANS sur le jeu de données `1sun`. Les ancres sont représentées avec une taille plus grande que les autres points.

Création de clusters. CP-MWCM permet à $\mathcal{P}_{t+1}$ d’avoir plus de clusters que $\mathcal{P}_t$, en élargissant le domaine des variables du COP à $[1, k']$ tel que $k' > k$. Cela ouvre la voie à des solutions affectant un point à un nouveau cluster si c’est la seule façon de parvenir à une solution. Par exemple, si on a $k = 2$ et que $\mathcal{C}_t$ contient trois contraintes $CL(i, j)$, $CL(j, l)$ et $CL(l, i)$, alors il est nécessaire de créer un nouveau cluster. Étant donné ses conditions d’apparition, il sera en général très petit. Afin d’éviter la création abusive de clusters, c’est-à-dire de former de nouveaux clusters simplement car cela mènerait à une solution optimale d’après la fonction objectif, la distance $\mathcal{D}[i, k']$ est choisie est une valeur plus grande que toutes les distances aux clusters originaux (Éq. 6.6) :

$$\mathcal{D}[i, k'] = \max(\{D[i, 1], \dots, D[i, k]\}) + 1, \quad \forall i \in \mathcal{X}, k < k' \quad (6.6)$$

6.1.2 Généraliser les modifications

Dans une application réelle, l’expert(e) va généralement réagir sur un nombre restreint de points lors d’une itération. Par conséquent, la modification minimale le sera tellement que la convergence du processus vers une partition satisfaisant l’utilisateur sera trop lente pour un cadre interactif, et ce d’autant plus que la taille du jeu de données est grande comparé au nombre de contraintes. D’où la nécessité d’exploiter efficacement le retour utilisateur pour généraliser² les modifications à des points non contraints mais pertinents pour améliorer la convergence et limiter le nombre de requêtes. Cette problématique a été soulevée dans Wagstaff (2007) ; elle est particulièrement important dans notre cadre incrémental. En outre, la généralisation doit être contrôlée pour que l’utilisateur puisse appréhender la portée des modifications supplémentaires générées par un tel mécanisme.

Nous faisons l’hypothèse que quand l’utilisateur sélectionne un point pour une contrainte, il ou elle ne se focalise souvent pas sur ce point en particulier mais sur la région alentour. En gardant cette intuition à l’esprit, la généralisation aux plus proches voisins ou à l’aide d’un rayon de proximité semble adéquate, mais ces méthodes manquent de prévisibilité : si un point se trouve dans le rayon de deux autres points contraints qui ont été réaffectés à des clusters différents, le résultat de la généralisation ne sera pas évident à comprendre.

2. La littérature du clustering actif utilise souvent le terme de *propagation* de contraintes (Lu et Peng, 2013) pour parler de ce type de mécanisme, mais nous préférons utiliser le terme de généralisation pour éviter la confusion avec la propagation de contraintes en PPC.

Nous proposons d'utiliser des super-instances, c'est-à-dire des points virtuels regroupant plusieurs points réels, pour généraliser les modifications. Notons que la généralisation ne signifie pas que de nouvelles contraintes sont générées, mais plutôt que les super-instances sont passées directement au modèle à la place des points réels. De cette manière, toute modification de l'appartenance à un cluster d'une super-instance revient à modifier l'appartenance de chaque point qui la compose. La portée de la généralisation est contrôlée comme suit : moins un cluster est divisé, plus l'impact d'une modification est important. La portée dépend donc du nombre de super-instances, déterminé par un taux β proportionnel à la taille du cluster. Ainsi, fixer β à par exemple 10% signifie que chaque cluster sera divisé en un nombre de super-instances égal à 10% de sa taille, 1 étant équivalent à ne pas généraliser du tout. Les super-instances sont calculées en divisant chaque cluster de la partition existante en plus petits groupes, chaque groupe représentant une super-instance. Nous avons constaté empiriquement que le clustering hiérarchique à liens complets (*complete-link*) est adéquat, malgré son utilisation de la mémoire qui le rend inadapté aux grands jeux de données. D'autres alternatives incluent l'algorithme OPTICS basé sur la densité comme dans [Mitchell \(2016\)](#), ou *Furthest Point First* (FPF) ([Gonzalez, 1985](#)). Cependant, les contraintes de l'utilisateur définies sur les points doivent être transférées aux super-instances. Cela peut entraîner des conflits potentiels. Pour éviter cet écueil, nous nous assurons que chaque super-instance ne contient pas plus d'un point de données contraint. Si ce n'est pas le cas, nous scindons la super-instance en utilisant les points contraints comme centres des nouvelles super-instances scindées.

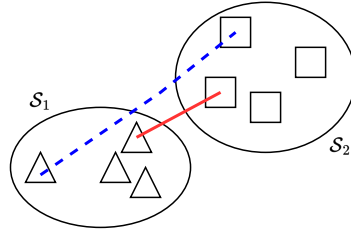


FIGURE 6.2 – Exemple de contraintes émergentes : une paire de points appartenant à deux super-instances S_1 et S_2 sont soumises à une contrainte *must-link* (en orange), tandis qu'une autre paire est soumise à une contrainte *cannot-link* (en bleu pointillés). Il se trouve alors que S_1 et S_2 sont en même temps soumis à une contrainte *must-link* et une contrainte *cannot-link*, ce qui rend le problème insoluble.

6.1.3 Formulation du modèle en PPC

Afin de bénéficier de la flexibilité des approches déclaratives, nous formulons un problème d'optimisation sous contraintes (COP) réalisant la modification minimale pondérée d'une partition en utilisant les contraintes définies en Section 2.3. Tout au long de cette section, le terme de *point* fera référence à un point du jeu de données ou à une super-instance sans distinction.

Variables et fonction objectif. Seuls les points qui sont soumis à des contraintes sont considérés dans le COP. La fonction POINTSCONSTRAINTS dans l'algorithme 26 produit le sous-ensemble des points contraints $X \subseteq \mathcal{X}$. Les variables du modèle sont :

- G_i de domaine $[1, k]$ pour chaque point contraint $i \in X$. $G_i = c$ signifie que le point i est affecté au cluster c dans la nouvelle partition ;
- Z_a de domaine $[0, 1]$ pour chaque contrainte a . $Z_a = 1$ si et seulement si a est satisfaite. Cela permet de relâcher le problème en permettant de ne pas satisfaire certaines contraintes et de contrôler le taux de satisfaction (cf. 6.1.5).

En utilisant l'équation 6.5, la fonction objectif est :

$$\arg \min \sum_{i \in X} \mathbb{I}(G_i \neq p_i) \mathcal{D}[i, G_i] \quad (6.7)$$

avec p_i l'affectation de i dans la partition courante $\mathcal{P}_t$ et $\mathcal{D}$ la matrice de distance entre les points contraintes et les représentants des clusters les plus proches (Section 6.1.1).

Une contrainte globale *element* relie pour chaque point son affectation à sa distance au représentant le plus proche de cette affectation via la matrice de distance :

$$\forall i \in \mathcal{X}, \text{element}(G_i, \mathcal{D}[i], \mathcal{D}[i, G_i]) \quad (6.8)$$

La contrainte 6.8 assure que le poids d'une modification dépend du cluster cible, de sorte que les meilleures modifications soient bien minimales.

Contraintes utilisateur Nous encodons dans le modèle les contraintes définies en Section 2.3.1, à l'aide de contraintes globales définies en PPC le cas échéant :

- Les contraintes ML/CL peuvent être exprimées par $G_i = G_j$ pour les ML et $G_i \neq G_j$ pour les CL.
- Les contraintes triplet s'écrivent à l'aide d'une implication :

$$G_a = G_n \implies G_a = G_p \quad (6.9)$$

- Les contraintes d'étendue limitée sont modélisées avec des contraintes de domaine fini ou des contraintes globales. Pour le cas spécifique où un groupe de point S doit être affecté aux clusters d'un ensemble prédéfini C , la contrainte est représentée avec une restriction de domaine :

$$G_i \in C, \forall i \in S \quad (6.10)$$

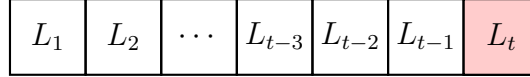
- Pour les contraintes d'étendue limitée génériques limitant à γ le nombre d'affectations différentes parmi les points de S la contrainte globale *atmost_nvalue* est utilisée (Bessière et al., 2006) :

$$\text{atmost_nvalue}(\gamma, \{G_i \mid i \in S\}) \quad (6.11)$$

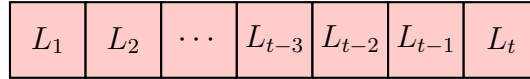
6.1.4 Construction de l'ensemble de contraintes

Soit L_t le lot de contraintes collecté à l'itération t . L'ensemble de contraintes $\mathcal{C}_t$ considéré à l'itération t dans l'algorithme 25 peut être construit suivant plusieurs stratégies de rétention des contraintes antérieures :

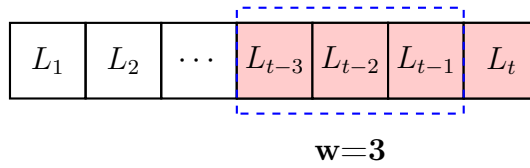
1. **Aucune rétention** : $\mathcal{C}_t = L_t$ l'ensemble de contraintes n'utilise que le dernier lot obtenu à l'itération actuelle. Les retours passés sont considérés comme encodés dans la partition courante, bien que rien ne le garantisse. Cette stratégie limite la taille du COP en termes de variables puisqu'elle est alors limitée au budget de requêtes. Elle évite tout conflit avec les retours plus anciens ; il faut noter qu'avec les stratégies de clustering actif basées sur les voisinages, il ne peut y avoir de conflit entre les contraintes issues de la partie active. Cependant, aucune information sur les choix passés n'est conservée hormis la partition elle-même. En particulier, les contraintes relâchées sont perdues étant données qu'elles n'ont pas contribué à une modification.



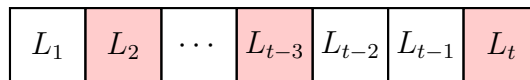
2. **Rétention totale** : $\mathcal{C}_t = \mathcal{C}_{t-1} \cup L_t = \bigcup_{i=1}^t L_i$ les contraintes sont ajoutées en continu à un ensemble de contraintes en croissance constante. Il est facile de revenir à une itération antérieure et de laisser l'utilisateur explorer différentes directions. Des conflits peuvent survenir si de nouvelles contraintes sont incompatibles avec d'autres plus anciennes ; la taille du COP augmente de façon monotone, ce qui pourrait rendre le passage à l'échelle difficile après un grand nombre d'itérations. Tous les lots de contraintes doivent être maintenus en mémoire pendant le processus.



3. **Least Recently Used (LRU)** : $\mathcal{C}_t = \bigcup_{k=1}^w L_{t-k} \cup L_t$ avec w le nombre de lots gardés. Un compromis simple avec les stratégies précédentes où seuls les w derniers lots sont retenus. Quand un nouveau lot est obtenu, le plus vieux lot est remplacé par le nouveau suivant le principe FIFO³. L'idée sous-jacente est que les retours anciens deviennent de moins en moins pertinents et peut être ignorées au bout d'un nombre arbitraire d'itérations. Dans le cas où l'expert(e) change d'avis sur la partition espérée, cette stratégie a l'avantage de faire disparaître les contradictions potentielles entre les deux visions concurrentes.



4. **Stratégie best-first** : $\mathcal{C}_t = \arg \max_{T \subseteq S, |T|=k} \sum_{x \in T} score(x) \cup L_t$ avec $score(L_i)$ un critère de qualité. Cette stratégie retient les meilleurs lots selon un critère donné. Parmi les critères pertinents pour CoActiVe, on peut citer le nombre de modifications induites ou la similarité entre les partitions successives. Cette stratégie peut être appliquée à un niveau plus fin en choisissant les contraintes de chaque lot individuellement.



3. First In, First Out

6.1.5 Relâchement des contraintes

Pendant l'utilisation de CoActiVe, des erreurs peuvent être introduites en essayant de modifier la partition, ce qui résulterait en conflits parmi les contraintes et à un problème surcontraint. Pour éviter cela, nous réifions les contraintes utilisateur pour avoir le contrôle sur leur satisfaction :

$$Z_i \iff c_i, \forall c_i \in \mathcal{C} \quad (6.12)$$

Les variables booléennes Z sont vraies si et seulement si la contrainte utilisateur correspondante est satisfaite. Un taux de satisfaction δ permet de fixer une borne inférieure⁴ au nombre de contraintes devant être satisfaites :

$$\sum_{c \in \mathcal{C}} Z_c \geq \delta \cdot |\mathcal{C}| \quad (6.13)$$

Comme le relâchement de contraintes est dépendant du COP et de son objectif, les premières contraintes à être relâchées sont celles qui induiraient la plus grande modification de la partition. L'introduction du relâchement de contraintes a cependant un impact sur le temps d'exécution en raison de la réification des contraintes avec les variables booléennes supplémentaires. En termes d'implémentation, CP-MWCM peut détecter automatiquement les conflits entre contraintes ML/CL et réduire le taux de satisfaction en conséquence. Par exemple, si $\delta = 100\%$ et $\mathcal{C} = \{ML(i, j), ML(j, k), CL(i, k)\}$, CP-MWCM va déduire $ML(i, k)$ et $CL(j, k)$ par fermeture transitive et identifier que $ML(i, k)$ et $CL(i, k)$ ne peuvent pas être satisfaites en même temps, tout comme $ML(j, k)$ et $CL(j, k)$. δ va alors être abaissé à 60%, permettant ainsi de relâcher deux contraintes sur cinq. La formule générale pour calculer ce nouveau taux est :

$$\delta_{conf} = 1 - \frac{n_{conf}}{|\mathcal{C}|} \quad (6.14)$$

avec n_{conf} le nombre de conflits ML/CL détectés.

Les conflits plus complexes ne sont pas pris en charge et nécessitent d'autres mécanismes (cf. Chapitre 8).

La contrainte 6.13 peut être modifiée pour inclure des préférence utilisateur en ajoutant un poids w_c à chaque contrainte c . Le seuil devient une limite de poids W_{lim} que l'on peut fixer de manière analogue comme une proportion du poids de l'ensemble de contraintes :

$$\sum_{a \in \mathcal{C}} w_a \times Z_a \geq W_{lim} \quad (6.15)$$

Complexité du modèle. Soit n_b le nombre de points impliqués dans des contraintes. Le modèle contient $n_b + |\mathcal{C}|$ variables – une variable par point contraint et une variable booléenne par contrainte. Dans le pire des cas où l'ensemble des points sont contraints, le nombre de variables devient $|\mathcal{X}| + |\mathcal{C}|$. Comme n_b contraintes *element* sont nécessaires pour suivre le poids des modifications, le nombre de contraintes du modèle est $n_b + |\mathcal{C}|$.

4. Cette borne peut facilement être adaptée pour donner une borne supérieure ou un taux exact de satisfaction de contraintes.

6.2 Expérimentations

Les expériences suivantes répondent aux **questions de recherche** (QR) pertinentes à propos de CP-MWCM :

1. **QR1** Quelle est l'influence des paramètres (centres ou ancrs pour calculer la matrice de distance, taille des super-instances) sur la performance de notre méthode ?
2. **QR2** Comment évolue le temps de calcul en fonction du nombre de contraintes et de la taille des données ?
3. **QR3** Comment se comporte le relâchement de contraintes de notre système par rapport à d'autres méthodes avec des contraintes *soft* ?
4. **QR4** Notre méthode tire-elle mieux avantage de la proposition de contraintes que l'état de l'art ?
5. **QR5** Comment se comporte notre méthode en termes de qualité de clustering, de similarité entre les partitions, et de temps de calcul, comparée à l'état de l'art ?

6.2.1 Protocole expérimental

CP-MWCM est implémenté en Python 3.11 avec la librairie `CPMpy`. Le solveur CP-SAT est utilisé pour résoudre le COP. Le choix de Python est motivé par la possibilité d'utiliser facilement des librairies d'apprentissage automatique comme `scikit-learn` en conjonction avec la PPC. Les implémentations d'algorithmes utilisés pour comparaison proviennent tous de la librairie `active-semi-supervised-clustering`⁵. Les expériences ont été réalisées sur un ordinateur comportant deux processeurs 48 coeurs Intel Xeon cadencés à 4 GHz et 64 Go de RAM sous Ubuntu 20.04.

Pour évaluer la performance des méthodes, nous utilisons trois métriques : l'ARI, l'AMI et le FMI (cf. Section 2.2.1). Nous comparons les partitions obtenues via les méthodes à la partition considérée comme vérité terrain pour évaluer la **qualité** du clustering; nous comparons deux partitions successives produites par la même méthode pour évaluer la **similarité** entre les résultats. Nous mesurons également le temps de calcul des partitions.

Nous utilisons des jeux de données de l'UCI utilisés dans de nombreux autres travaux en clustering sous contraintes, ainsi que de la suite FCPS (cf. table 2.3).

Nous utilisons comme méthodes de sélection, la sélection aléatoire et NPU ([Xiong et al., 2014](#)). Les requêtes produisent des contraintes ML/CL, en utilisant comme oracle la vérité terrain des jeux de données pour simuler les retours utilisateur : si deux points ont la même étiquette selon cette vérité terrain, une contrainte ML est produite. Dans le cas contraire, une contrainte CL est produite. Aucune réponse indifférente n'est générée.

L'expérience principale (répondant aux **QR4** et **QR5**) consiste à calculer une première partition avec KMEANS ([MacQueen, 1967](#)) en paramétrant le nombre de clusters K comme étant celui de la vérité terrain, puis à poser 10 requêtes à l'oracle à l'aide d'un algorithme de sélection de contraintes. Si besoin, la fermeture transitive de l'ensemble de contraintes est calculée (cf. Section 2.3). La partition est modifiée pour satisfaire les contraintes.

5. <https://github.com/datamole-ai/active-semi-supervised-clustering>

L'opération est répétée 10 fois.

Étant donnés les effets aléatoires dans la génération de contraintes, nous répétons les expériences 90 fois. Nous effectuons une comparaison bayésienne des méthodes telle que décrite dans Benavoli et al. (2017) pour les tâches de classification (cf. Section 2.5.1). Les résultats complets sont disponibles sur un dépôt public⁶.

6.2.2 Paramétrage de CP-MWCM

Afin de répondre à la QR1, nous évaluons les effets de différentes combinaisons de valeurs pour les paramètres de CP-MWCM : le taux de génération des ancres $\alpha \in \{0\%, 5\%, 20\%\}$ et le taux de génération des super-instances $\beta \in \{10\%, 30\%, 50\%, 100\%\}$, soit un total de 12 configurations. Pour rappel, $\alpha = 0\%$ signifie que seuls les médoïdes des clusters sont calculés, tandis que $\beta = 100\%$ marque l'absence de généralisation à l'aide des super-instances. Pour chaque configuration et chacune des trois métriques, nous réalisons une comparaison pair à pair bayésienne selon le Q-AUBC et le S-AUBC (cf. Section 2.2.2). On compte le nombre de victoires de chaque configuration.

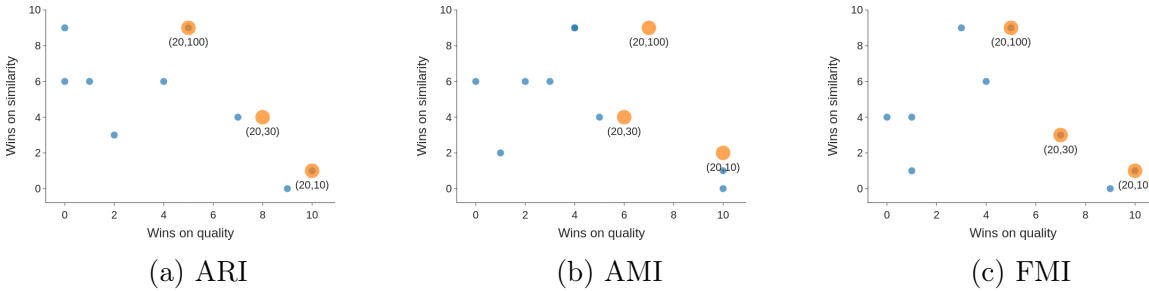


FIGURE 6.3 – Comparaison du nombre de victoires pour chaque configuration (α, β) avec un modèle hiérarchique bayésien, selon le Q-AUBC (horizontal) ou le S-AUBC (vertical). Les meilleures valeurs sont proches du haut à droite, avec les configurations sur le front de Pareto en orange.

D'après la Table 6.1, les configurations utilisant les ancres (i.e. $\alpha \neq 0$) donnent les plus hautes valeurs de Q-AUBC. Les meilleurs résultats sont obtenus avec $\alpha = 20\%$. Si on fixe la valeur de α , le Q-AUBC augmente au fur et à mesure que β diminue, la meilleure valeur étant 10%. Ce résultat implique qu'un grand⁷ rayon de généralisation ne semble pas avoir d'impact délétère sur la modification de clustering et par extension sa qualité. Les valeurs de S-AUBC ont un comportement opposé : $\beta = 10\%$ est le pire paramétrage pour la similarité. De ce point de vue, l'impact de α semble négligeable comparé à β . Il améliore légèrement la similarité quand β est faible, et sa contribution devient presque inexistante en l'absence de généralisation. Généraliser les modifications dégrade logiquement la similarité, mais pas considérablement. Un front de Pareto émerge de ces résultats pour chaque métrique (cf. figure 6.3) : une optimale en qualité (20%, 10%), une autre optimale en similarité (20%, 100%), et un compromis entre les deux (20%, 30%).

La figure 6.4 montre la distribution postérieure de la comparaison pair à pair dans un simplexe pour des configurations illustrant les analyses précédentes (cf. figure 2.6).

6. <https://github.com/aymericb213/IAC>

7. Relativement aux valeurs testées dans nos expériences.

(α, β)	Q-AUBC			S-AUBC		
	ARI	AMI	FMI	ARI	AMI	FMI
(0, 10)	9 (1)	10 (1)	9 (1)	0	0	0
(0, 30)	2 (0)	1 (0)	1 (0)	3 (1)	2 (0)	1 (1)
(0, 50)	0	0	0	6 (3)	6 (2)	4 (2)
(0, 100)	0	4 (1)	3	9 (9)	9 (9)	9 (9)
(5, 10)	10 (3)	10 (3)	10 (3)	1 (0)	1 (0)	1 (0)
(5, 30)	7 (3)	5 (3)	7 (3)	4 (2)	4 (1)	3 (1)
(5, 50)	1 (0)	2 (0)	1 (0)	6 (4)	6 (3)	4 (2)
(5, 100)	5 (2)	4 (0)	5 (1)	9 (9)	9 (9)	9 (9)
(20, 10)	10 (7)	10 (6)	10 (6)	1 (1)	2 (1)	1 (0)
(20, 30)	8 (3)	6 (3)	7 (3)	4 (2)	4 (2)	3 (1)
(20, 50)	4 (1)	3 (1)	4 (1)	6 (5)	6 (4)	6 (4)
(20, 100)	5 (2)	7 (3)	5 (1)	9 (9)	9 (9)	9 (9)

TABLE 6.1 – Nombre de victoires pour chaque configuration (α, β) avec un modèle hiérarchique bayésien. Les valeurs entre parenthèses indiquent le nombre de cas où la probabilité de la configuration gagnante est plus grande que 95%. Les valeurs de α et β sont exprimées en pourcentages.

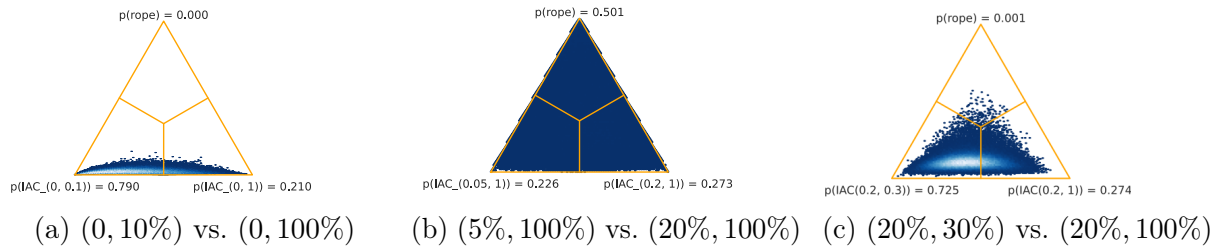


FIGURE 6.4 – Vue en simplexe de la comparaison bayésienne de deux configurations selon le Q-AUBC (*ARI*).

6.2.3 Impact du nombre et du type de contraintes utilisateur sur le temps de calcul

Nous examinons la [QR2](#) en étudiant deux points : l'efficacité d'exécution du modèle PPC et son expressivité. Nous considérons pour ces expériences trois tailles d'ensembles de contraintes (10, 100, 1000). Pour chaque cas, nous générons aléatoirement des ensembles de chacun des types de contraintes utilisables dans CP-MWCM. Nous calculons une partition initiale avec KMEANS, puis le modèle PPC fait une modification par ensemble de contraintes. Nous mesurons le temps de calcul processeur moyen sur 90 essais. Pour les contraintes ML/CL, la fermeture transitive n'est pas calculée pour s'assurer que le nombre de contraintes à traiter ne change pas. Les contraintes d'étendue limitée sont créées en tirant aléatoirement 10 points et en formant la plage de clusters (ou le nombre de clusters pour les d'étendue limitée génériques) à l'aide de la vérité terrain.

6.2. EXPÉRIMENTATIONS

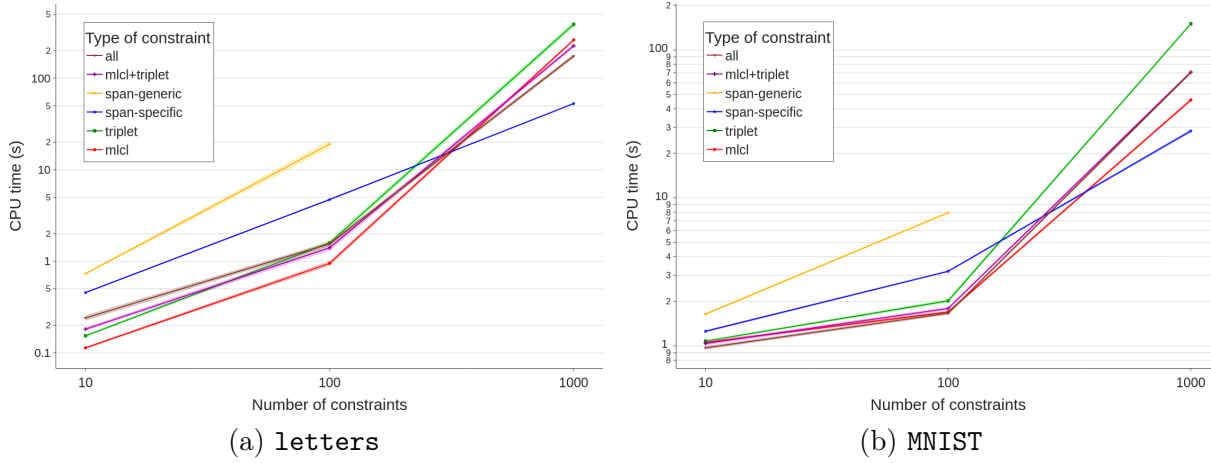


FIGURE 6.5 – Évolution du temps de calcul du modèle PPC pour **letters** et **MNIST** en fonction du nombre et du type de contraintes, avec les intervalles de confiance à 95% (échelle logarithmique).

Analyse du temps de calcul. La figure 6.5 montre les résultats obtenus pour les jeux de données **letters** et **MNIST** avec $\alpha = 0\%$ et $\beta = 100\%$, c'est-à-dire sans pré- ou post-traitement. Le modèle PPC peut traiter 10 contraintes ML/CL en moins de 0.05 secondes, tandis que pour les contraintes triplet et d'étendue limitée spécifiques le temps de calcul moyen est de 0.035 et 0.15 secondes respectivement. Avec 100 contraintes ML/CL, il faut en moyenne 0.35 secondes, là où 1.36 secondes sont nécessaires pour les contraintes triplet et d'étendue limitée. Cela paraît tout à fait raisonnable pour le contexte de CoActive. Avec 1000 contraintes, le temps de calcul augmente significativement et atteint 44 secondes pour les contraintes d'étendue limitée spécifiques, 67 secondes pour les ML/CL, et plus de 180 secondes pour les triplets; le système est tombé à court de mémoire avant que l'expérience puisse terminer pour les contraintes d'étendue limitée génériques. Ce dernier permet de délimiter les limites du passage à l'échelle de CP-MWCM. En pratique, le nombre de contraintes attendues à une quelconque itération se compte en dizaines plutôt qu'en centaines, et encore moins en milliers. L'augmentation de temps de calcul est plus marquée pour les contraintes complexes (triplet et étendue limitée) que pour les ML/CL. La table 6.2 montre que les ensembles de contraintes ML/CL générés pour les expériences contiennent en moyenne plus de CL que de ML (jusqu'à 97% sur **letters**, 90% sur **digits**, 81% sur **yeast**). Cela peut être attendu étant donné que la vérité terrain des jeux de données testés a un grand nombre de clusters. Cela réduit la probabilité que deux points tirés au hasard fassent partie du même cluster. Comme les contraintes CL sont plus difficiles à satisfaire que les ML, ces résultats signifient que les temps d'exécution ont été mesurés sur des cas relativement difficiles.

Combinaisons de types de contraintes. L'un des avantages de notre approche basée sur un paradigme déclaratif comme la PPC et sa capacité à traiter différents types de contraintes sans avoir à développer d'algorithme dédié. Afin de faire la démonstration de cette flexibilité, nous avons comparé deux compositions :

- **mlcl+triplet** : générer des ML/CL et des triplets en proportions égales, par exemple 50 ML/CL et 50 contraintes triplet pour le cas des 100 contraintes.

- **all** : même principe que **mlcl+triplet**, cependant une contrainte ML/CL est remplacée par une contrainte d'étendue limitée spécifique et un triplet substitué par une contrainte d'étendue limitée générique, utilisant ainsi tous les types de contraintes gérés par le modèle.

Nous constatons que les problèmes avec des ML/CL et des triplets sont résolus bien plus vite que ceux impliquant tous les types de contraintes. De façon surprenante, la combinaison ML/CL-triplet avec 1000 contraintes est même plus rapide que les cas avec 1000 triplets. L'ensemble de ces résultats montrent l'intérêt du clustering actif dans CoActive : il est important de sélectionner un petit nombre de contraintes qui ont un grand impact sur la modification pour avoir le meilleur compromis entre performance et rapidité.

Configuration	Taille	letters	% CL	digits	% CL	yeast	% CL
mlcl	10		97.0%	& 89.2%		78.4%	
mlcl	100		96.0%		90.0%		77.8%
mlcl	1000		96.3%		90.0%		81.0%
mlcl+triplet	10		49.7%		48.2%		40.5%
mlcl+triplet	100		49.8%		49.1%		47.4%
mlcl+triplet	1000		50.0%		49.7%		52.9%

TABLE 6.2 – Proportion moyenne de contraintes CL, groupée par jeu de données, configuration et taille.

6.2.4 Relâchement des contraintes

Nous répondons à la [QR3](#) en comparant CP-MWCM à d'autres méthodes pratiquant une satisfaction souple des contraintes. Nous avons choisi deux méthodes : PCK-MEANS ([Wagstaff et Cardie, 2000](#)) qui pénalise l'insatisfaction de contraintes ML/CL, et MPCK-MEANS ([Bilenko et al., 2004](#)) qui combine l'objectif de PCK-MEANS avec du *metric learning*.

Nous considérons deux situations :

- générer 950 contraintes aléatoirement en se basant sur la vérité terrain et ajouter 50 contraintes créant des conflits, pour forcer certaines contraintes à être relâchées pour résoudre le problème.
- générer 1000 contraintes aléatoirement sans conflit et fixer le taux de satisfaction δ de CP-MWCM au taux de satisfaction moyen de PCKMEANS et MPCKMEANS.

Nous utilisons deux paramétrages de CP-MWCM : CP-MWCM +Anchors ($\alpha = 20\%$, $\beta = 100\%$) et CP-MWCM +Anchors+SI ($\alpha = 20\%$, $\beta = 30\%$). La table 6.3 montre la performance des différentes approches sur le jeu de données **mk2**, en mesurant l'ARI, le temps d'exécution et le nombre de contraintes relâchées. Les résultats montrent qu'en présence de conflits, CP-MWCM viole moins de contraintes que les autres méthodes. En outre, le temps de calcul de CP-MWCM et de PCKMEANS sont comparables, alors que MPCKMEANS beaucoup plus coûteux en temps. En termes de qualité de clustering, CP-MWCM surpasse clairement les alternatives. Quand on impose un taux de satisfaction à CP-MWCM, PCKMEANS et MPCKMEANS atteignent une qualité comparable, mais la similarité reste faible. CP-MWCM arrive là encore à relâcher moins de contraintes.

6.2. EXPÉRIMENTATIONS

	Conflits seulement				$\delta = 94\%$			
	Qualité	Similarité	Temps	n_r	Qualité	Similarité	Temps	n_r
CoActiVe + Ancres	0.576	0.075	5.262	49.7	0.309	0.177	3.051	60.1
CoActiVe + Ancres+SI	0.760	0.024	4.868	49.83	0.393	0.108	2.972	60.1
PCK-MEANS	0.081	0.051	3.639	149.3	0.375	0.045	2.834	66.5
MPCK-MEANS	0.078	0.017	29.27	159.9	0.406	0.019	26.98	61.2

TABLE 6.3 – Étude comparative pour le clustering avec contraintes ML/CL avec relâchement sur le jeu de données **mk2**. Les métriques sont l’ARI avec la vérité terrain (Qualité), l’ARI avec KMEANS (Similarité), le temps de calcul et le nombre de contraintes relâchées dans la solution (n_r).

6.2.5 Comparaison avec l’état de l’art

Pour traiter les **QR4** et **QR5**, nous opérons une comparaison avec différentes méthodes de l’état de l’art. Nous avons modifié NPU pour l’utilisation en contexte incrémental (cf. Annexe B) et nous comparons CP-MWCM avec COPKMEANS (Wagstaff et al., 2001), PCKMEANS (Basu et al., 2004) et MPCKMEANS (Bilenko et al., 2004). Pour chaque algorithme testé, nous comparons la génération d’un lot de contraintes aléatoires et l’usage de NPU. En se basant sur les résultats de la section 6.2.2, nous choisissons la configuration de compromis entre qualité et similarité ($\alpha = 20\%$, $\beta = 30\%$) pour CP-MWCM.

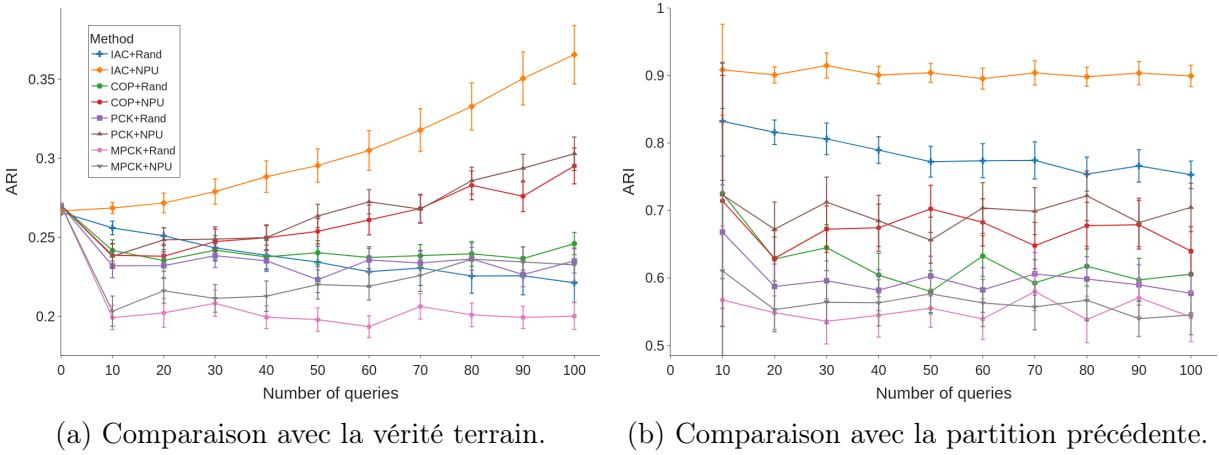


FIGURE 6.6 – ARI de la partition existante à chaque itération, comparée avec la vérité terrain (à gauche) ou la partition précédente partition (à droite), pour le jeu de données **glass**. Les valeurs sont les moyennes sur 90 essais avec intervalle de confiance à 95%. Les valeurs les plus hautes sont les meilleures.

Qualité et similarité. La figure 6.6 décrit l’évolution de l’ARI sur 10 itérations de CoActiVe pour le jeu de données **glass**. La combinaison CP-MWCM +NPU a tendance à produire des partitions continuellement meilleures au fur et à mesure des itérations, tout en gardant une similarité élevée. Elle produit la plus grande valeur de Q-AUBC et la plus grande valeur finale des métriques sur la plupart des jeux de données. En outre le comportement des algorithmes concurrents est plus erratique en termes de similarité.

Enfin, la sélection aléatoire de contrainte tend à produire les pires résultats parmi les configurations testées.

Nous validons ces observations par la comparaison bayésienne pour le Q-AUBC et le S-AUBC pour chaque métrique. La figure 6.7 montrent que CP-MWCM a une forte probabilité de surpasser les alternatives pour l'ARI ; les résultats obtenus pour l'AMI et le FMI sont proches. Il est intéressant de noter que l'usage de NPU impacte positivement la similarité (cf. figure 6.8). Cela permet de penser que l'usage du clustering actif peut apporter des bénéfices supplémentaires à l'amélioration de la qualité de la partition. Cela amène aussi à se demander si c'est l'utilisation des voisinages, maintenus au cours du processus, qui cause ce maintien de la similarité entre les partitions. L'inconvénient majeur par rapport à la sélection aléatoire reste le temps de calcul largement plus élevé, surpassant même le temps utilisé pour la modification. On peut en tout cas conclure à la lumière de ces résultats que CP-MWCM est bien la méthode la plus efficace pour fonctionner dans le cadre de CoActive, et que la modification minimale de partition est plus performante que le reclustering tant en termes de qualité que de similarité.

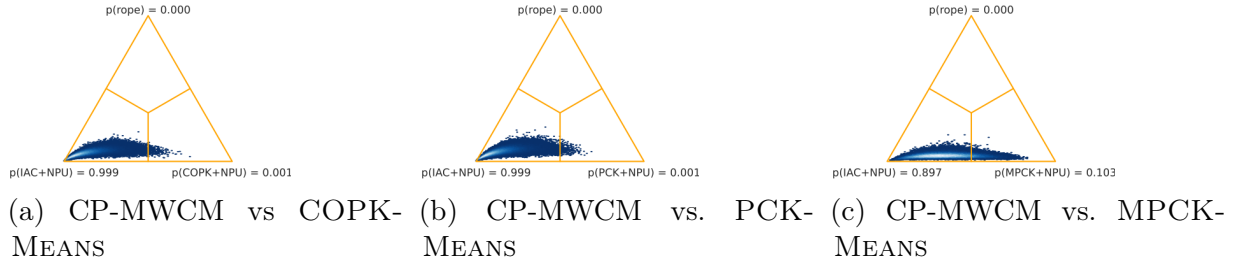


FIGURE 6.7 – Comparaison de CoActive avec CP-MWCM et NPU avec les alternatives sur le Q-AUBC (ARI).

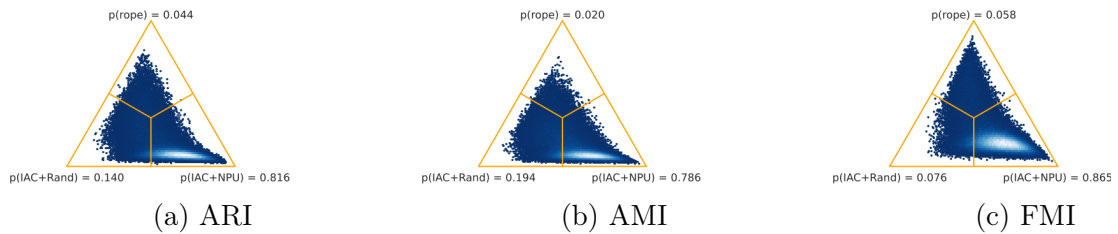


FIGURE 6.8 – Comparaison bayésienne de CoActive avec ou sans NPU pour le S-AUBC pour les trois métriques.

Temps de calcul. Dans la figure 6.9, le temps nécessaire à l'étape de modification (par reclustering ou MWCM) est noté pour chaque jeu de données. Le temps de calcul de CP-MWCM est comparable à celui des concurrents testés, quoiqu'il semble avoir un meilleur passage à l'échelle sur les plus grands jeux de données. Empiriquement, on constate que CP-MWCM est plus rapide d'un ordre de grandeur sur les jeux de données *yeast*, *statlog* et *letters* qui sont caractérisés par leur grand nombre de clusters. On peut également remarquer que MPCK-MEANS est beaucoup plus lent que les autres méthodes - probablement à cause du *metric learning* - mais que cette augmentation ne se traduit pas en une performance accrue par rapport à CP-MWCM. Le facteur limitant pour CP-MWCM

6.2. EXPÉRIMENTATIONS

se révèle être le calcul des ancres et des super-instances, qui ne permet pas de terminer à temps sur **Letters** et **MNIST**.

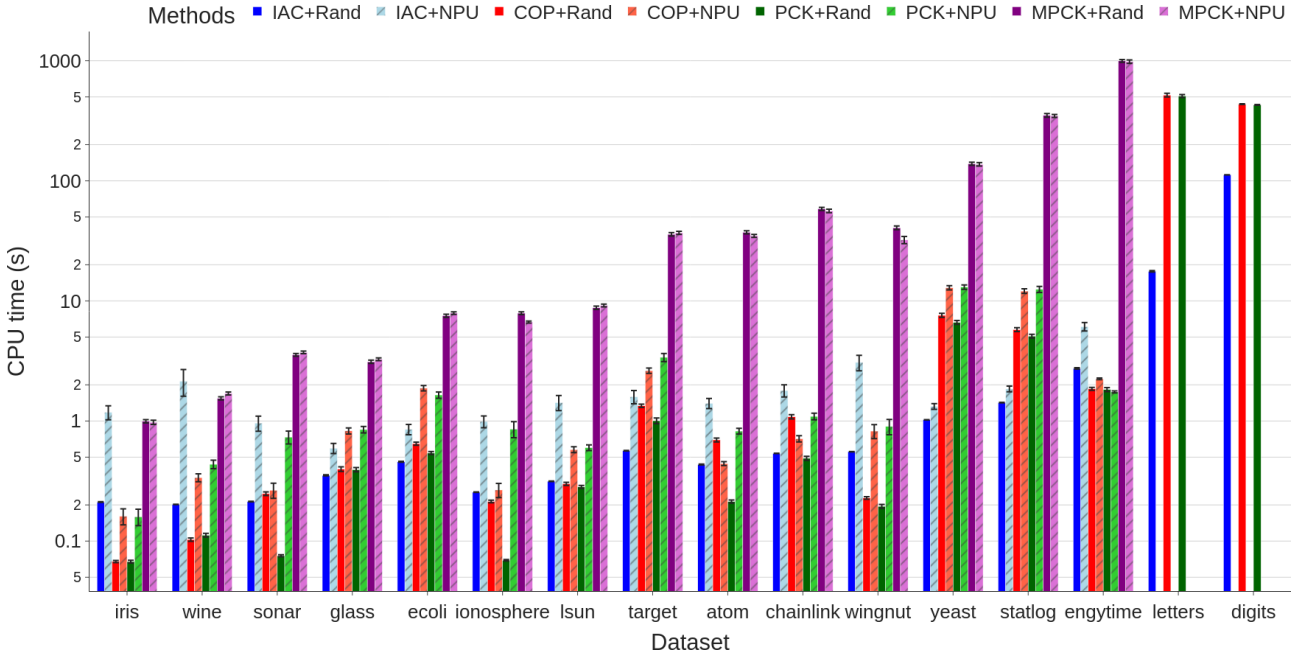


FIGURE 6.9 – Comparaison du temps de calcul des méthodes considérées (échelle logarithmique). Les méthodes utilisant NPU sont hachurées.

6.2.6 Gestion des contraintes

Nous avons comparé trois modalités de gestion des contraintes obtenues au cours du processus pour déterminer si cela avait un impact sur le temps de calcul et la qualité des résultats :

- donner toutes les contraintes en une seule fois
- n'utiliser que le lot de contraintes courant
- cumuler les lots de contraintes

Deux observations ressortent des expériences : premièrement, les modalités incrémentales sont plus rapides d'environ un ordre de grandeur par rapport à l'intégration de toutes les contraintes en une fois. Cela est probablement dû à la plus grande complexité du réseau de contraintes dans ce dernier cas. Deuxièmement, le cumul des contraintes au cours du temps n'impacte pas la qualité des partitions produites mais améliore leur similarité.

6.2.7 Cas d'étude : coupes de bois

Nous avons utilisé CP-MWCM pour résoudre le problème de clustering binaire présenté en Section 2.5.3. Nous nous sommes concentrés sur les deux plus grandes zones de coupe, car c'est sur celles-ci que sont posées les annotations à disposition. Pour rappel, le jeu de données a été séparé en deux clusters. Nous voulons placer l'ensemble des coupes de bois dans un seul cluster. En particulier, la plus grande zone de coupes de bois n'est quasiment pas présente dans le cluster qui leur est destiné. L'objectif est d'utiliser les annotations expert pour ramener cette zone dans le bon cluster.

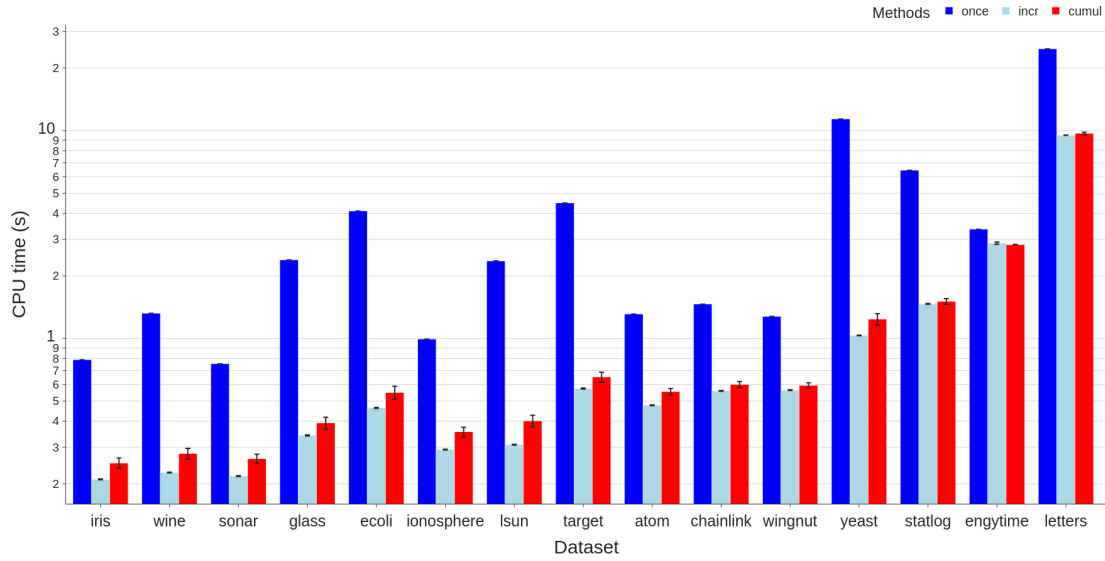


FIGURE 6.10 – Temps de calcul pour les trois modalités de gestion des contraintes : **once** pour la prise en compte de toutes les contraintes en une seule fois, **incr** pour le fonctionnement incrémental par défaut où seul le lot de contraintes courant est utilisé et **cumul** pour la stratégie de cumul des lots de contraintes au cours du temps (échelle logarithmique).

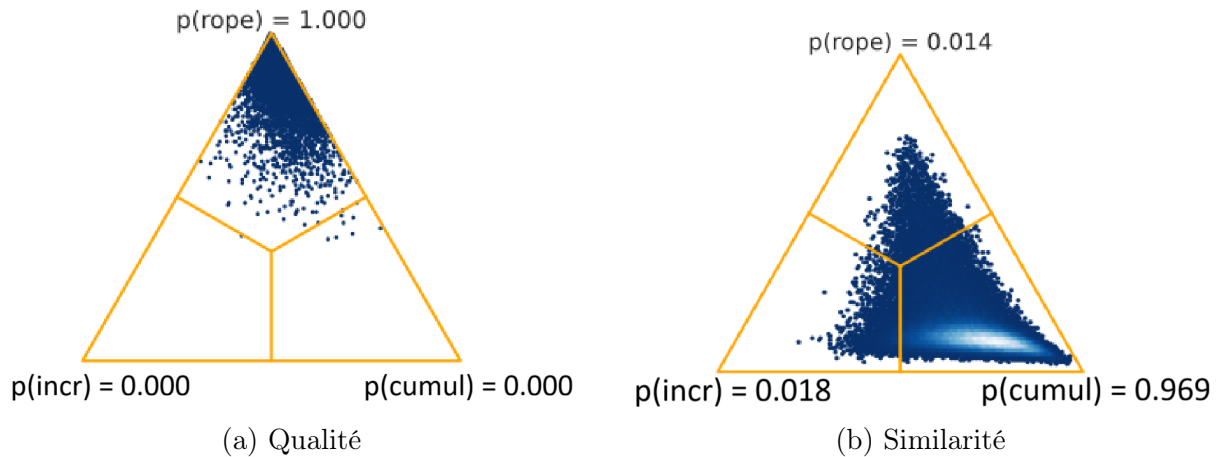


FIGURE 6.11 – Comparaison bayésienne des deux modalités incrémentales : cumul des contraintes au cours du temps ou intégration des seules contraintes de l'itération courante

Résultats. Le cluster modifié des coupes de bois est présenté dans la figure 6.12b. Nous pouvons observer le recouvrement d'une partie des coupes de bois, visible par l'élargissement des zones vertes : la zone de gauche est passée de 37 à 92 vrais positifs, couvrant maintenant près de la moitié de la zone. La zone de droite a gagné 10 vrais positifs. La modification a été effectuée en 22 secondes. La généralisation n'a pas été utilisée car le découpage du jeu de données entier en super-instances était trop coûteux en temps. On peut imaginer qu'une généralisation adaptée à l'application, c'est-à-dire centrée sur les zones annotées, aurait pu améliorer encore les résultats.

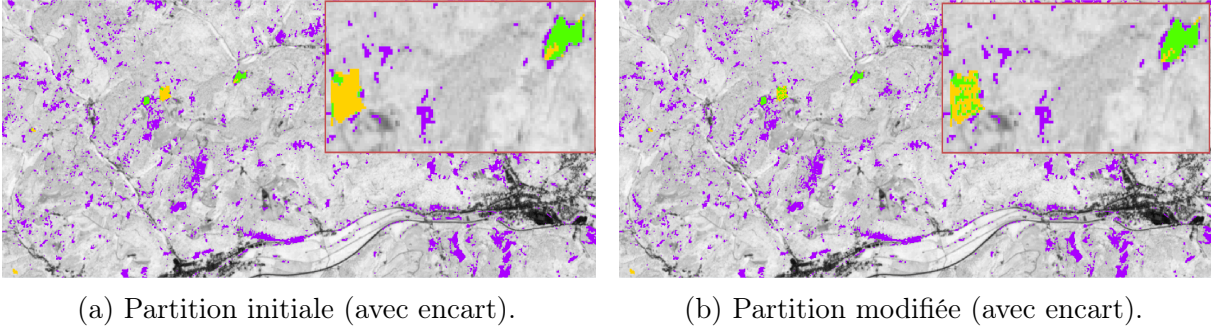


FIGURE 6.12 – Vues de la modification sur la zone spatiale ; les images montrent le cluster des "positifs", avant (6.12a) et après (6.12b) la modification. Les vrais positifs sont marqués en **vert**, les faux négatifs en **jaune** et les faux positifs en **violet**. Les pixels en noir et blanc ne font pas partie du cluster des positifs. Visionnage en couleurs conseillé.

À l'issue de ces expérimentations, nous avons pu vérifier que CP-MWCM arrive à maintenir la similarité entre les itérations successives. Nous avons aussi observé qu'elle exploite mieux les contraintes obtenues par le clustering actif et qu'elle les intègre plus rapidement que des algorithmes de clustering sous contraintes traditionnels.

Nous avons cependant remarqué que le facteur limitant pour CoActiVe était le temps requis pour la sélection active. Cela pose un dilemme au regard des algorithmes actuels : privilégier la rapidité ou la performance. Cette observation nous a orienté vers la conception d'une méthode de clustering actif qui trouve rapidement un lot de contraintes tout en maintenant une efficacité acceptable. Exploiter le retour utilisateur disponible au fil des itérations pour la sélection ainsi que pour pouvoir proposer une caractérisation des clusters attendus est une autre motivation.

Chapitre 7

Rendre le clustering actif utilisable en temps réel

Table des matières

7.1	Élicitation basée sur les règles pour le clustering sous contraintes incrémentales (REACT)	110
7.1.1	Extraction des règles	110
7.1.2	Génération de requêtes	111
7.2	Expérimentations	113
7.2.1	Protocole expérimental	114
7.2.2	Pertinence des aires de confiance	115
7.2.3	Efficacité individuelle des heuristiques	115
7.2.4	Comparaison avec d'autres approches basées sur les voisinages	117

Dans la seconde partie de la thèse, nous avons développé un algorithme de clustering actif fait pour CoActiVe, que nous avons appelé REACT (*Rule-based Elicitation for Active Constraint-incremental clusTering*) et qui se base sur un arbre de décision uniquement entraîné sur les points dont l'affectation a déjà été précisée par l'utilisateur (**des points certifiés**). L'arbre est idéalement appris sur des attributs de haut niveau, facilement interprétables par l'expert(e), constituant ce que nous appelons un **espace des connaissances** $\mathcal{K}$ sur les mêmes points que $\mathcal{X}$, mais $\mathcal{X}$ peut être employé comme espace des connaissances par défaut. Notre méthode extrapole l'apprentissage sur l'ensemble du jeu de données et confronte les prédictions de l'arbre à la partition existante pour trouver des requêtes pertinentes. L'idée principale derrière cette proposition est de se servir du cadre incrémental dans lequel elle s'inscrit pour "deviner" la partition espérée par l'utilisateur et la comparer au résultat actuel pour s'appuyer sur les conflits et les concordances entre les deux. Si aucune méthode dans la littérature en clustering actif n'a proposé ce concept à notre connaissance, le principe présente des similarités avec le domaine du *few-shot learning* (Song et al., 2023), où l'on cherche à apprendre un classifieur du mieux possible avec une très petite taille d'échantillons pour l'entraînement.

7.1 Élicitation basée sur les règles pour le clustering sous contraintes incrémentales (REACT)

Étant donné que nous avons déjà un moyen de modifier une partition avec CP-MWCM, nous cherchons à développer une méthode agnostique, prenant en compte le jeu de données ainsi que la partition courante. Elle vise à être une implémentation de *selectionActive* (cf. ligne 9 de l'algorithme 25) dans le fonctionnement général de CoActiVe. Nous avons remarqué que la façon dont les voisinages sont construits dans NPU est tout à fait adaptée au contexte incrémental de CoActiVe, c'est pourquoi nous avons cherché à les exploiter dans notre méthode. La sortie de l'algorithme doit être compatible avec les contraintes gérées par CP-MWCM.

La procédure de REACT est résumée dans l'algorithme 27. La première étape est l'initialisation des voisinages $\mathcal{N}$ (cf. section 4.3) qui seront nécessaires aux étapes suivantes. Nous nous basons sur la version incrémentale de NPU utilisée dans les expériences précédentes, que nous employons pour la première itération. Nous nous servons des valeurs d'entropie calculées par NPU lors de l'initialisation pour la sélection des points à l'aide d'heuristiques (Section 7.1.2.2). EXTRACTIONREGLES (Section 7.1.1) va généraliser les informations des voisinages sous forme de règles qui sont ensuite utilisées dans EXPLOITATIONCONNAISSANCES (Section 7.1.2) pour sélectionner des points candidats aux requêtes grâce à des heuristiques.

Algorithme 27 Rule-based Elicitation for Active Constraint-incremental clustering (REACT)

Entrée: Espace de clustering $\mathcal{X}$, espace des connaissances $\mathcal{K}$, partition $\mathcal{P}$, voisinages $\mathcal{N}$, budget de requêtes b

Sortie: lot de contraintes ML/CL L

- 1: **si** $\mathcal{N} = \{\}$ **alors**
 - 2: $L_{init}, \mathcal{N} \leftarrow \text{INITIALISER}(\mathcal{X}, \mathcal{P}, \mathcal{N}, b)$ ▷ Algorithme 30
 - 3: **retourner** L_{init}
 - 4: $\mathcal{R}, \mathcal{P}_{\mathcal{T}} \leftarrow \text{EXTRACTIONREGLES}(\mathcal{K}, \mathcal{N}, \mathcal{P})$ ▷ Algorithme 28
 - 5: $L, \mathcal{N} \leftarrow \text{EXPLOITATIONCONNAISSANCES}(\mathcal{X}, \mathcal{P}_{\mathcal{T}}, \mathcal{N}, \mathcal{R})$ ▷ Algorithme 29
 - 6: **retourner** L
-

7.1.1 Extraction des règles

Une règle R est notée (A_R, C_R) . A_R est l' **antécédent** de la règle, c'est-à-dire un ensemble de conditions sur les attributs de l'espace de connaissances $\mathcal{K}$. C_R est le **cluster conséquent**, c'est-à-dire le cluster où les points qui respectent A_R devraient se trouver d'après la règle. Soit en termes plus naturels : "les points qui présentent telles caractéristiques - ou suivent tel motif - ont tendance à appartenir à tel cluster."

Par l'emploi de l'ensemble de voisinages $\mathcal{N}$, REACT extrait des règles qui tentent de définir les clusters espérés d'après les *retours utilisateurs passés* représentés par $\mathcal{N}$, pas d'après la *partition existante*. Pour chaque voisinage $N \in \mathcal{N}$ (donc chaque cluster de la partition espérée, cf. Propriété 4.5), l'appartenance aux clusters dans la partition courante $\mathcal{P}$ est utilisée comme étiquetage pour entraîner un arbre de décision $\mathcal{T}$ (cf. Alg.

7.1. ÉLICITATION BASÉE SUR LES RÈGLES POUR LE CLUSTERING SOUS CONTRAINTES INCRÉMENTALES (REACT)

28). Chaque chemin de la racine à une feuille de $\mathcal{T}$ forme une règle, l'antécédent étant la séquence de noeuds traversés par le chemin, et la conséquence le cluster majoritaire dans la feuille.

Algorithme 28 EXTRACTIONRÈGLES

Entrée: Espace de connaissances $\mathcal{K}$, voisinages $\mathcal{N}$

Sortie: règles extraites $\mathcal{R}$, partition prédite $\mathcal{P}_{\mathcal{T}}$

- 1: $\mathcal{R} \leftarrow \{\}$
 - 2: $\mathcal{T} \leftarrow \text{ARBREDÉCISION}(\mathcal{K}, \mathcal{N})$ ▷ Apprentissage sur les voisinages
 - 3: **pour** chaque feuille $l \in \mathcal{T}$ **faire**
 - 4: $\mathcal{R} \leftarrow \mathcal{R} \cup \{(\text{chemin}(\mathcal{T}, l), \arg \max_{C \in \mathcal{P}} |\{x \in l \mid p_x = C\}|)\}$ ▷ Chemin de décision de la racine à l
 - 5: $\mathcal{P}_{\mathcal{T}} \leftarrow \text{prediction}(\mathcal{X}, \mathcal{T})$ ▷ Partition prédite d'après l'arbre de décision
 - 6: **retourner** $\mathcal{R}, \mathcal{P}_{\mathcal{T}}$
-

Une fois entraîné, $\mathcal{T}$ peut prédire à quel cluster espéré chaque point non certifié devrait être affecté en suivant les règles apprises. En confrontant cette prédiction à la partition courante $\mathcal{P}$, nous appelons **exemples** de l'une règle les points qui respectent l'antécédent de la règle et qui appartiennent au cluster conséquent de la règle dans $\mathcal{P}$. À l'inverse, les points qui respectent l'antécédent sans appartenir au cluster prédit dans $\mathcal{P}$ sont appelés les **contre-exemples** de la règle.

Apprendre l'arbre de décision sur $\mathcal{N}$ uniquement, sans utiliser le reste des données, permet d'éviter les biais associés aux algorithmes de clustering traditionnels qui peuvent exister dans la partition courante¹. Le plus grand intérêt de cette technique réside dans la capacité de confronter la partition existante à une autre partition qui tente de généraliser les retours utilisateur à l'ensemble du jeu de données.

7.1.2 Génération de requêtes

Les règles extraites sont ensuite utilisées dans deux buts complémentaires : (1) trouver de nouvelles contraintes informatives, cohérentes avec le retour utilisateur antérieur et non satisfaites par la partition existante ; (2) trouver des points "sûrs" que l'on peut considérer comme certifiés sans les présenter directement à l'utilisateur.

7.1.2.1 Aire de confiance

Comme les règles extrapolent des décisions utilisateur binaires à partir des points certifiés - peu nombreux par définition - elles sont susceptibles d'être en partie erronées notamment dans les premières itérations. Néanmoins, on peut raisonnablement supposer que les points proches des exemples certifiés d'une règle sont probablement aussi des exemples. REACT applique cette intuition en calculant des **aires de confiance** (AC) qui délimitent pour chaque règle la région où l'extrapolation est la plus fiable. Trois stratégies ont été étudiées pour les calculer :

1. **Enveloppe convexe** : le plus petit polygone convexe englobant tous les exemples certifiés d'une règle ;

1. Le biais résiduel le plus important est le nombre de classes à apprendre, qui dépend du nombre de clusters de $\mathcal{N}$.

7.1. ÉLICITATION BASÉE SUR LES RÈGLES POUR LE CLUSTERING SOUS CONTRAINTES INCRÉMENTALES (REACT)

Aire de confiance	Extrapolation	Complexité temporelle	Points requis
Enveloppe convexe	petite	$O(n \log n_{hull})$	$m + 1$
Hyper-rectangle	moyenne	$O(n^2 m)$	2
Rayon de proximité	grande	$O(n^2)$	2

TABLE 7.1 – Stratégies employées pour trouver des points candidats pour des requêtes.

2. **Hyper-rectangles** : le plus petit hyperrectangle de dimension m recouvrant tous les exemples certifiés ;
3. **Rayon de proximité** : une région définie comme l'union de sphères centrées sur chaque exemple certifié. Nous utilisons $\frac{d_{max}}{2}$ comme rayon des sphères, où d_{max} est la distance maximale entre deux exemples certifiés de la même règle, pour s'assurer que l'aire résultante soit continue.

Ces stratégies ont chacune leurs avantages et inconvénients, résumés dans la table 7.1. L'enveloppe convexe est la plus restreinte mais son calcul est le plus complexe, notamment dans des données avec beaucoup d'attributs. En effet, il faut $m + 1$ points pour calculer une enveloppe convexe de dimension m , ce qui la rend quasiment impossible à construire avant d'avoir réalisé de nombreuses itérations de CoActiVe. De plus, le fléau de la dimension implique que ces enveloppes ont une probabilité non négligeable d'être vides et donc peu exploitables. A l'inverse, le rayon de proximité fait l'extrapolation la plus large mais est très simple à calculer, tandis que l'hyper-rectangle fait le compromis entre les deux autres stratégies.

7.1.2.2 Heuristiques de requête

En utilisant le statut des points au regard des règles (exemple/contre-exemple) et leur position par rapport aux aires de confiance (dedans ou dehors), l'algorithme 29 sélectionne des candidats aux requêtes d'après les heuristiques suivantes :

1. **H1** : sélectionner les top- k exemples les plus incertains puis choisir **celui qui est le plus éloigné de l'aire de confiance**. Cet exemple va soit augmenter la taille de l'AC (si son statut d'exemple est certifié) ou induire des changements significatifs dans le prochain arbre de décision.
2. **H2** : le **contre-exemple hors de l'aire de confiance** le plus incertain, proche de la frontière de décision ;
3. **H3** : le **contre-exemple moyen dans l'aire de confiance**, qui est le plus susceptible de diviser la règle actuelle en deux s'il s'agit bien d'un contre-exemple ;
4. **H4** : l'**exemple dans l'aire de confiance** le plus proche d'un point certifié utilisé dans la construction de l'AC est ajouté au voisinage suivant la règle.

Ces heuristiques sont résumées dans la table 7.2 selon le cas d'application. La stratégie générale derrière ces heuristiques vise deux objectifs : (1) étendre l'AC afin de couvrir le coeur des clusters attendus par l'expert(e) ; (2) introduire des perturbations qui vont produire des arbres de décision divers au fil des itérations.

	Dans AC	Hors AC
Exemple	plus proche de l'AC	plus loin parmi les top- k incertains
Contre-exemple	contre-exemple moyen	contre-exemple le plus incertain

TABLE 7.2 – Tableau des heuristiques selon le statut du point et sa présence dans l'AC.

Une fois que tous les candidats ont été sélectionnés, une suite de requêtes est construite par traversée de loin en loin (*farthest-first traversal*) des candidats afin d'explorer des régions diverses de l'espace et d'exploiter les différentes règles extraites.

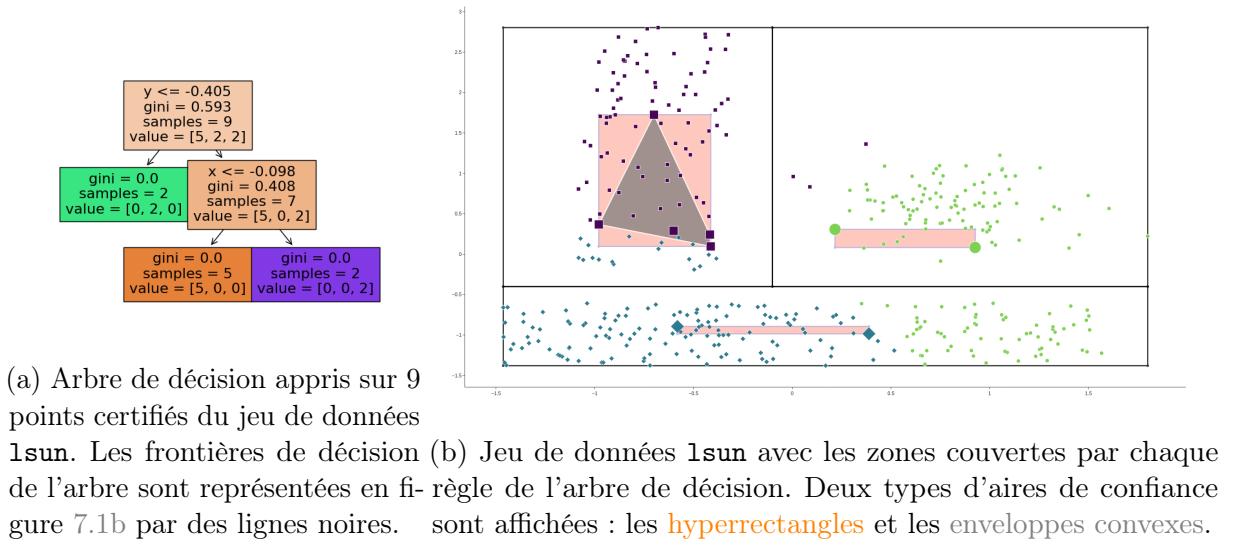


FIGURE 7.1 – Exemple illustratif du concept d'aires de confiance sur le jeu de données 1sun.

7.2 Expérimentations

Les expériences suivantes répondent aux **questions de recherche** (QR) pertinentes à propos de REACT :

1. **QR1** Les aires de confiance vérifient-elles l'hypothèse selon laquelle les points proches de points certifiés sont généralement aussi des exemples de règle ?
2. **QR2** Quelle heuristique de sélection de candidats est la plus efficace ?
3. **QR4** Comment se comporte REACT en termes de qualité de clustering, de similarité entre les partitions, et de temps de calcul, comparée à l'état de l'art ?
4. **QR3** Le temps d'exécution de REACT est-il suffisant pour le contexte de CoActive ?

Algorithme 29 KNOWLEDGEEXPLOITATION

Entrée: Espace de clustering $\mathcal{X}$, prédiction de l'arbre de décision $\mathcal{P}_T$, voisinages $\mathcal{N}$, règles $\mathcal{R}$, budget b

Sortie: lot de contraintes ML/CL L

```
1:  $\mathcal{Q} \leftarrow \{\}$ 
2:  $q \leftarrow 0$ 
3:  $\mathcal{A} \leftarrow \{\}$ 
4: pour chaque règle  $R = (A_R \implies C_R) \in \mathcal{R}$  faire
5:    $e \leftarrow \{x \in \mathcal{X} \mid x \in A_R \wedge x \in C_R\}$  ▷ Exemples
6:    $ce \leftarrow \{x \in \mathcal{X} \mid x \in A_R \wedge x \notin C_R\}$  ▷ Contre-exemples
7:    $\mathcal{H} \leftarrow \text{AIRECONF}(\mathcal{X}, \mathcal{N} \cap e)$  ▷ Calcul de l'aire de confiance
8:   Ajouter les candidats sélectionnés d'après H1, H2, H3 à  $\mathcal{Q}$ 
9:  $vus \leftarrow \{\}$ 
10: tant que  $q < b$  faire
11:    $x^* \leftarrow$  point de  $\mathcal{Q}$  le plus éloigné de  $vus$ 
12:    $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{x^*\}$ 
13:    $vus \leftarrow vus \cup \{x^*\}$ 
14:   pour chaque voisinage  $N$  de  $\mathcal{N}$  faire
15:      $A \leftarrow \text{requête}(x^*, y), y \in N$ 
16:      $q \leftarrow q + 1$ 
17:     si  $A = \text{"ML"}$  alors
18:        $N \leftarrow N \cup \{x^*\}$  ▷ Ajout à un voisinage existant
19:        $L \leftarrow L \cup \{ML(x^*, y)\}$ 
20:     sinon
21:        $L \leftarrow L \cup \{CL(x^*, y)\}$ 
22:   si aucun ML obtenu alors
23:      $\mathcal{N} \leftarrow \mathcal{N} \cup \{\{x^*\}\}$  ▷ Nouveau voisinage
24: retourner  $L$ 
```

7.2.1 Protocole expérimental

De manière analogue aux expérimentations du chapitre 6, nous initialisons CoActive avec une partition de KMEANS (MacQueen, 1967) avec K fixé au nombre de clusters de la vérité terrain de chaque jeu de données. Nous utilisons la vérité terrain pour simuler l'expert(e) et répondre aux requêtes. Pour les **QR2**, **QR3** et **QR4**, nous réalisons 10 itérations de CoActive avec CP-MWCM à l'étape de modification. Pour lisser les effets aléatoires à l'initialisation et dans la sélection de contraintes, nous répétons chaque expérience 10 fois. Les résultats complets sont disponibles sur un dépôt public².

Nous avons comparé REACT avec les algorithmes basés sur les voisinages décrits en section 4.4.1 : FFQS, MINMAX et NPU. Nous avons aussi comparé les performances de REACT en fonction de la stratégie utilisée pour les AC.

Pour les jeux de données non-mixtes, l'extraction et la modification de clustering sont réalisées dans le même espace ($\mathcal{X} = \mathcal{K}$). Pour les jeux de données mixtes, chaque vue est utilisée pour une étape séparée.

2. <https://github.com/aymericb213/REACT>

7.2.2 Pertinence des aires de confiance

Afin de déterminer dans quelle mesure les aires de confiance permettent d’extrapoler les retours utilisateur (**QR1**), nous avons comparé le taux moyen d’exemples *dans l’aire de confiance* qui devraient être groupés avec les points certifiés de leur règle à ce même taux pour les exemples *en dehors de l’AC*.

Jeu de données	Env. convexe		Hyper-rectangle		Rayon de proximité	
	$rate_{in}$	$rate_{out}$	$rate_{in}$	$rate_{out}$	$rate_{in}$	$rate_{out}$
yeast	-	-	50	16.6	<i>29.9</i>	15.4
iris	100	100	<i>25</i>	23.9	<i>24.7</i>	22.7
ionosphere	-	-	100	83.8	<i>85.4</i>	68.6
wine	-	-	100	98.3	<i>90</i>	50
glass	-	-	100	13.6	<i>50</i>	11.7
target	-	-	7.40	7.28	7.40	7.28
engyttime	96.1	94.6	<i>21.7</i>	21.6	<i>21.7</i>	20.6
aniso	100	51.3	<i>94.9</i>	53.8	<i>89.2</i>	50
blobs	97.7	100	66.3	<i>66.5</i>	<i>66.4</i>	66.3
circles	100	52.6	<i>98.7</i>	63.2	<i>88.9</i>	7.8
halfmoons	100	97.4	97.4	100	100	97.0
varied	100	100	49.0	<i>49.6</i>	<i>49.2</i>	0.5
uci_digits	-	-	50	17.9	<i>46.7</i>	16.4
treecut	-	-	25	19.7	<i>22.7</i>	13.9

TABLE 7.3 – Taux de ML entre un exemple d’une règle et un point certifié qui sont en accord avec la vérité terrain, dans l’aire de confiance ($rate_{in}$) et en dehors ($rate_{out}$), pour la **première** itération de CoActiVe avec REACT. Le meilleur score pour une AC donnée est en *italique*, le meilleur score global est en **gras**.

Les résultats en tables 7.3 et 7.4 montrent que, quelle soit la stratégie utilisée pour calculer l’AC, les points à l’intérieur ont une plus grande probabilité d’appartenir au voisinage de la règle que les points en dehors après l’initialisation (cf. table 7.3). Au fur et à mesure que l’AC s’étend, elle devient moins fiable comme l’atteste la table 7.4 montrant les résultats après 10 itérations. On vérifie également que l’enveloppe convexe est inefficace pour les jeux de données de grandes dimensions. L’hyper-rectangle semble être la technique qui reste la plus fiable tout en fonctionnant sur la plupart des jeux de données. Comme aucune stratégie ne s’est révélée dominante par rapport aux autres à l’issue de cette expérience, nous avons continué à tester toutes les stratégies dans les expérimentations suivantes.

7.2.3 Efficacité individuelle des heuristiques

Comme la performance de REACT dépend de la combinaison de plusieurs heuristiques, nous avons réalisé une étude par ablation en testant isolément chacune des heuristiques qui permettent d’obtenir des candidats (H1, H2, H3, cf. section 7.1.2). Pour chacune d’entre elles, nous avons compté le nombre de ML/CL générées (**QR2**).

7.2. EXPÉRIMENTATIONS

Jeu de données	Env. convexe		Hyper-rectangle		Rayon de proximité	
	$rate_{in}$	$rate_{out}$	$rate_{in}$	$rate_{out}$	$rate_{in}$	$rate_{out}$
yeast	-	-	21.5	7.8	0.9	0.1
iris	12.5	12.1	22.1	1.8	41.1	4.2
ionosphere	-	-	28.3	17.9	9.7	8.2
wine	-	-	41.6	41.1	17.0	5.5
glass	-	-	38.8	30.0	38.5	0
target	35.4	9.3	21.1	25.3	28.3	27.5
engytime	17.6	17.6	21.3	20.8	27.9	0.8
aniso	41.6	30.5	64.7	48.6	62.8	40.5
blobs	20.8	25.0	40.2	41.6	40.6	12.3
circles	37.5	29.0	65.7	51.6	21.6	0.6
halfmoons	17.1	19.7	14.9	17.7	48.5	33.3
varied	20.8	22.9	7.4	5.5	30.2	23.6
uci_digits	-	-	41.1	33.9	41.7	37.3
treecut	6.2	5.9	5.6	4.3	0.9	0.3

TABLE 7.4 – Taux de ML entre un exemple d’une règle et un point certifié qui sont en accord avec la vérité terrain, dans l’aire de confiance ($rate_{in}$) et en dehors ($rate_{out}$), pour la **dixième** itération de CoActiVe avec REACT. Le meilleur score pour une AC donnée est en *italique*, le meilleur score global est en **gras**.

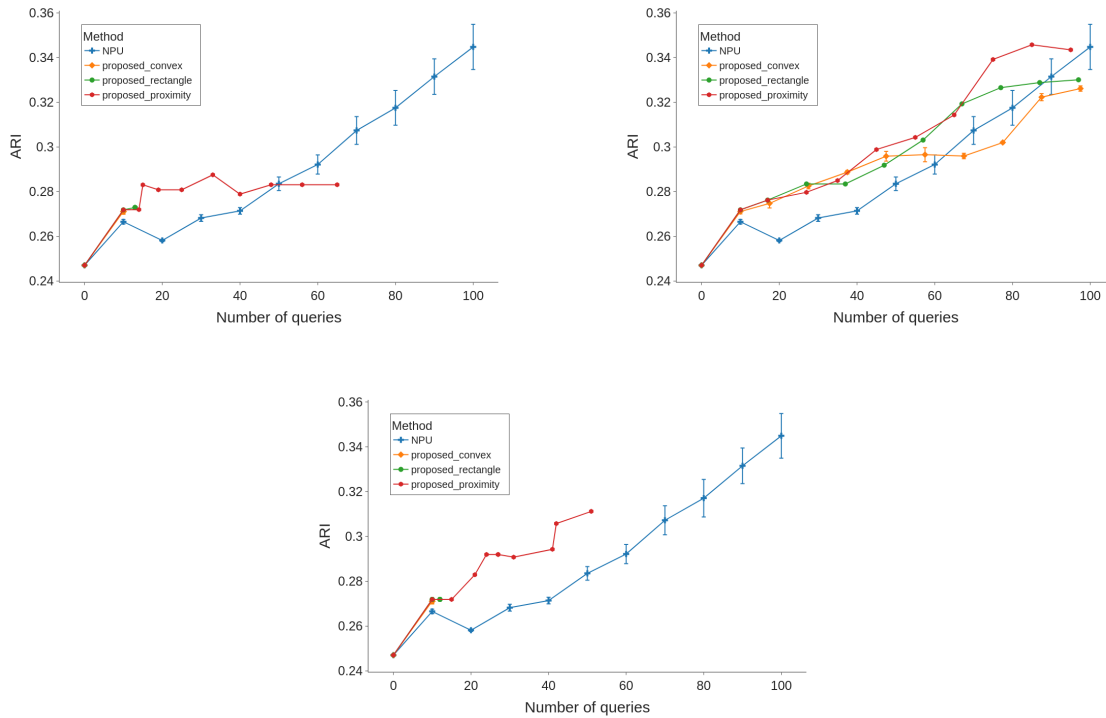


FIGURE 7.2 – ARI des heuristiques isolées sur le jeu de données **glass** pour chaque AC. Les résultats obtenus avec NPU sont pris comme référence.

Jeu de données	H1			H2			H3		
	ML	CL	Ratio	ML	CL	Ratio	ML	CL	Ratio
iris	15	0	∞	12	8	1.5	1	0	∞
wine	2	0	∞	23	9	2.55	0	0	-
glass	0	2	0	12	25	0.48	0	0	-
ionosphere	0	0	-	23	5	4.6	0	0	-
yeast	1	0	∞	22	25	0.88	0	0	-
letters	0	0	-	2	41	0.05	0	0	-
target	12	0	∞	11	16	0.69	10	0	∞
engytme	25	1	25	17	13	1.31	4	5	0.8
aniso	15	2	7.5	20	2	10	5	0	∞
varied	9	1	9	13	9	1.44	2	0	∞
circles	18	0	∞	14	4	3.5	9	0	∞
blobs	18	1	18	1	0	∞	0	0	-
halfmoons	18	0	∞	17	3	5.67	13	0	∞
uci_digits	0	0	-	4	34	0.12	0	0	-
treecut	13	0	∞	15	40	0.38	0	0	-

TABLE 7.5 – Nombre de contraintes *must-link* et *cannot-link* générées en utilisant uniquement l’heuristique H1, H2 ou H3 sur les jeux de données testés. Le ratio $\frac{ML}{CL}$ montre combien de contraintes ML ont été générées en plus par rapport aux contraintes CL.

A partir de la table 7.5, nous pouvons conclure que l’heuristique qui génère le plus de requêtes est H2, c’est-à-dire l’heuristique sur les contre-exemples hors de l’aire de confiance. L’heuristique H1 (exemples hors de l’AC) fournit surtout des contraintes ML, ce qui la rend plus efficace pour étendre l’AC que pour perturber les frontières de décision. Enfin, l’heuristique H3 sur les contre-exemples dans l’AC donne très peu de contraintes ; ce cas en particulier a peu d’occurrences dans les expériences réalisées. D’après les résultats de la section 7.2.2, cela signifie que trouver des exemples dans l’AC qui ne font en réalité pas partie du voisinage associé à la règle serait plus fructueux.

7.2.4 Comparaison avec d’autres approches basées sur les voisinages

Nous avons comparé REACT à d’autres méthodes de clustering actif basées sur les voisinages (Section 4.4.1). Pour rappel, FFQS fonctionne en deux phases, l’une d’exploration qui vise à découvrir tous les voisinages avant de les consolider par des requêtes sur des points choisis aléatoirement. MINMAX remplace la sélection aléatoire de FFQS par un critère de similarité, pour choisir le point dont la plus grande similarité avec un quelconque voisinage est la plus faible. NPU n’a pas de phase d’exploration et choisit les points en fonction de l’entropie des probabilités d’appartenance aux voisinages et de l’espérance du nombre de requêtes à faire pour déterminer le voisinage d’un point.

7.2.4.1 Qualité

Pour répondre à la **QR3**, nous avons comparé l'évolution de l'ARI de la partition sur 10 itérations de CoActive en faisant varier l'algorithme utilisé pour la sélection de contraintes.

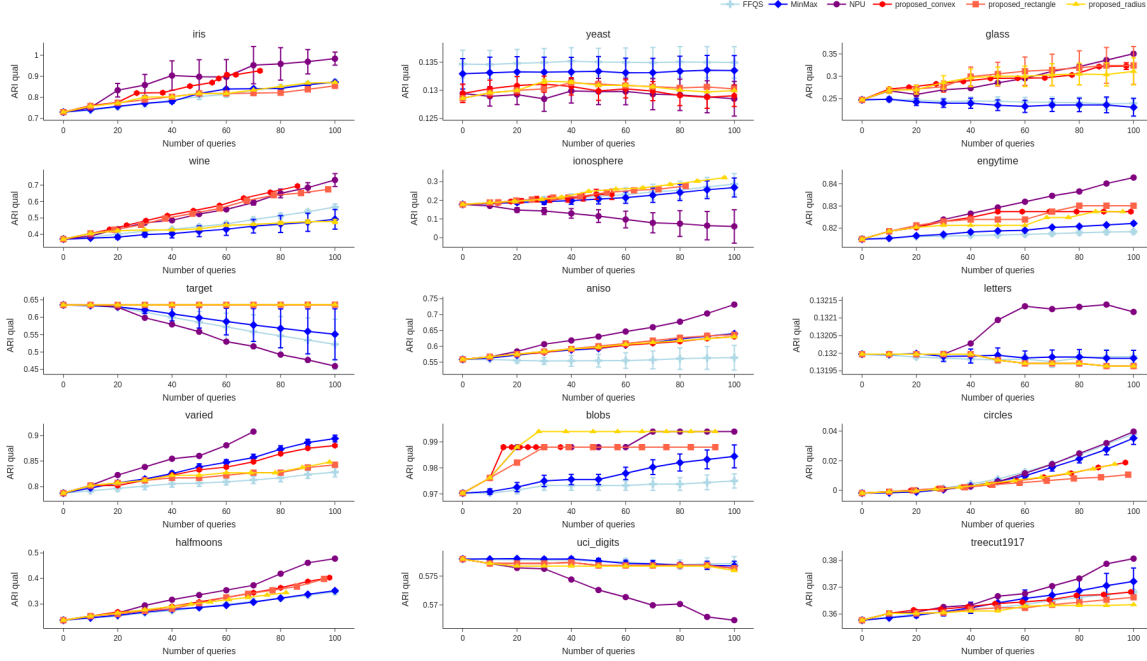


FIGURE 7.3 – ARI pour les jeux de données testés.

D'après la figure 7.3, REACT a une performance variable selon les jeux de données testés : il surpasse FFQS et MINMAX sur la plupart des jeux de données mais NPU reste globalement le meilleur algorithme en termes de qualité, exceptés certains cas (*ionosphere*, *target*, *uci_digits*) où on observe une baisse drastique par rapport à la partition de départ.

Encore une fois, aucune stratégie de calcul d'AC ne ressort particulièrement. Les performances sont généralement similaires, et dans les cas particuliers où une stratégie domine les autres, ce n'est jamais la même. Cela indique que le choix de la stratégie pour le calcul des aires de confiance n'est pas trivial. Il peut être guidé par le jeu de données et la stratégie pourrait être modifiée d'une itération à l'autre de CoActive.

7.2.4.2 Temps de calcul

Afin de savoir si REACT construit un lot de contraintes plus rapidement que les compétiteurs (**QR4**), nous avons mesuré le temps d'exécution moyen sur les 10 itérations de l'expérience précédente.

Nos expériences montrent que REACT est plus rapide d'environ un ordre de grandeur par rapport aux autres méthodes basées sur les voisinages (cf. figure 7.4). Il faut notamment remarquer que la majorité du temps de calcul enregistré vient de l'initialisation basée sur NPU ; l'usage des heuristiques dans les itérations suivantes est beaucoup plus rapide.

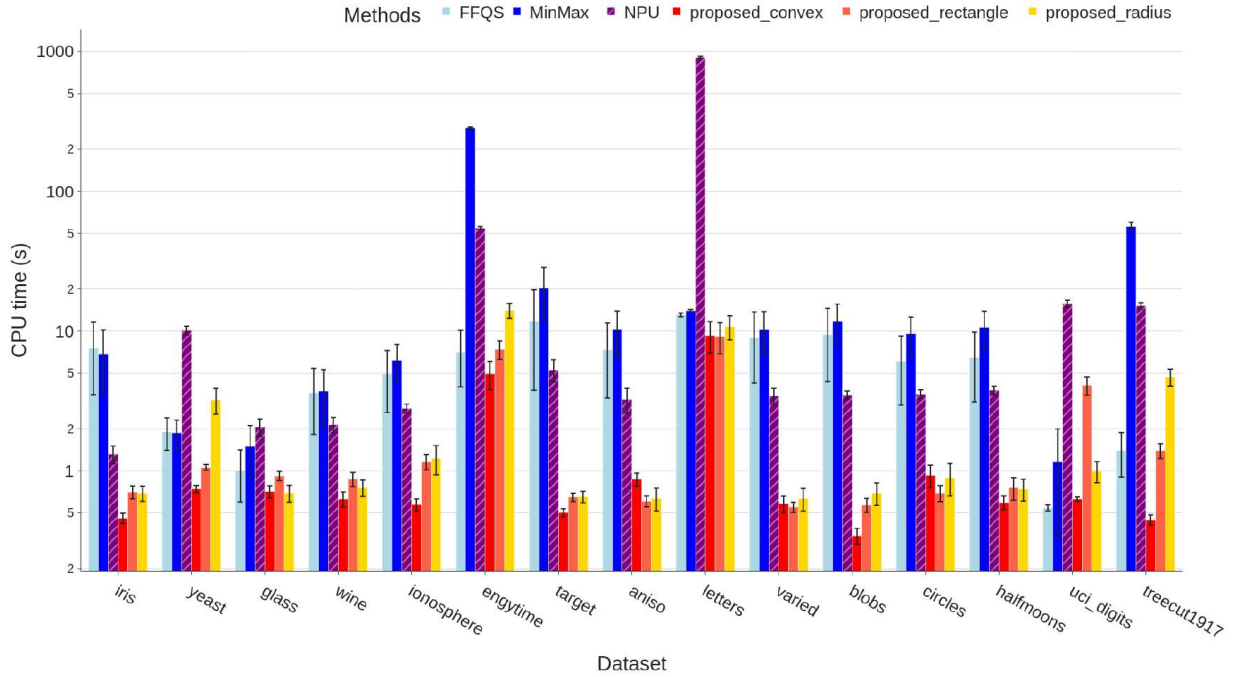


FIGURE 7.4 – Temps de calcul des méthodes comparées sur les jeux de données testés en secondes (échelle logarithmique).

Pour prendre un exemple concret, l'initialisation prend 15 minutes sur le jeu de données **Letters** alors que les étapes d'apprentissage de l'arbre, de collecte des candidats puis de sélection des contraintes sont résolues en 11 secondes par itération environ - soit une accélération d'un facteur 80 - et cela peu importe la stratégie employée pour les AC. Quand on compare les stratégies de calcul d'AC, les résultats tendent à montrer que la stratégie la plus rapide est celle de l'enveloppe convexe, bien qu'elle ait la plus grande complexité théorique. Cela peut s'expliquer par la faible taille des enveloppes pour les jeux de données bi- ou tridimensionnels. Pour les autres, le nombre de points requis par règle étant trop élevé pour construire la moindre enveloppe en 10 itérations de CoActiVe, le temps d'exécution mesuré est principalement celui de l'apprentissage de l'arbre de décision.

A la sortie de ces expérimentations, on peut conclure que REACT peut être utilisé en temps réel sur de plus grands jeux de données que les algorithmes de l'état de l'art utilisant les voisinages. L'étape la plus demandeuse en temps est l'initialisation, basée sur NPU. Les étapes suivantes sont calculées en une dizaine de secondes au pire, ce qui est acceptable dans un contexte interactif. Les contraintes sélectionnées permettent d'améliorer la partition courante de façon similaire aux autres. De plus REACT exploite les voisinages de façon plus approfondie en y associant des règles de décision tentant de décrire les clusters espérés, et qui deviennent de plus en plus précise au fur et à mesure

que les voisinages grossissent.

CoActiVe est maintenant capable de maintenir la similarité entre les résultats grâce à CP-MWCM et d'obtenir des contraintes en temps raisonnable grâce à REACT. Il reste à explorer un aspect crucial de ce système interactif : la gestion des erreurs.

Chapitre 8

Expliquer les conflits et proposer la meilleure solution

Table des matières

8.1	Adéquation des contraintes à un modèle	122
8.2	Modèle PPC pour choisir le MCS d'adéquation minimale . .	124
8.3	Expérimentations	125
8.3.1	Protocole expérimental	126
8.3.2	Impact du modèle sur l'adéquation	127
8.3.3	Impact du retrait des contraintes sur le clustering	128
8.3.4	Passage à l'échelle du modèle	129
8.3.5	Cas d'étude : coupes de bois	130

Comme toute application centrée sur l'interaction avec l'humain, il est primordial d'être préparé à l'arrivée d'erreurs dans les retours utilisateur pour s'assurer de la viabilité de CoActiVe dans des applications réelles. En l'occurrence, les erreurs prennent la forme de conflits dans les contraintes données, produisant une situation où toutes les contraintes ne pourront pas être satisfaites en même temps.

Plusieurs cas peuvent mener à l'apparition de ces conflits :

- l'utilisation simultanée des deux sources de contraintes (ajout manuel et clustering actif) ;
- les partitions successives peuvent provoquer un déclic mettant au jour une autre direction que celle envisagée au début et provoquer une modification de la partition espérée par l'expert(e), amenant des conflits entre les contraintes venant de la vision initiale et celles données après le déclic ;
- pour une même partition espérée, des incohérences entre le dernier lot obtenu et les contraintes précédentes gardées par la stratégie de rétention peuvent apparaître (Section 6.1.4) ;
- la fermeture transitive peut mettre au jour des conflits entre des contraintes apparemment cohérentes.

Pour éviter un blocage de CoActiVe, il est nécessaire d'y intégrer un mécanisme de résolution de conflits. Comme nous l'avons vu en Section 6.1.5, CP-MWCM peut relâcher une partie des contraintes en cas de conflit mais la relaxation est guidée par la fonction objectif, de telle sorte que les contraintes induisant la plus grande modification seront

relâchées en premier. Or il n'est pas toujours souhaitable de dépendre de cet objectif pour la résolution de conflits. Nous voulons être capables d'expliquer pourquoi une contrainte a été volontairement ignorée en utilisant les retours utilisateurs passés.

C'est pourquoi nous nous sommes orientés vers le calcul d'un sous-ensemble de correction minimal (MCS, cf. Définition 2.19) à présenter à l'utilisateur.

8.1 Adéquation des contraintes à un modèle

Il existe souvent de multiples façons de résoudre un 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. C'est un critère simple, mais il arrive souvent que de nombreux MCS aient la même cardinalité minimale. Or nous ne souhaitons présenter qu'un seul MCS à l'utilisateur. Il est également possible de pondérer les contraintes et chercher le MCS minimisant le poids des contraintes qui le constituent.

C'est dans cette optique que nous présentons une mesure d'adéquation estimant à quel point l'affectation des points d'une contrainte dans un modèle entraîné sur les retours utilisateur passés (étiquetage partiel du jeu de données, contraintes déjà fournies...) est en accord avec la définition de la contrainte. L'intuition 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 faire le parallèle avec la mesure d'informativité présentée dans [Davidson et al. \(2006\)](#) : un ensemble de contraintes informatif 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 de 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. Ce modèle tente ensuite de généraliser aux points qui ne sont liés à aucune contrainte, représentant ainsi une approximation de la partition espérée par l'utilisateur, d'après les informations connues. On reprend ici le principe utilisé dans REACT (Section 7.1.1). 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 espérée.

Définition 8.1 (Adéquation de contrainte à un modèle). Soit $\mathcal{M}$ un modèle d'apprentissage supervisé appris sur un sous-ensemble $\mathcal{X}_c \subseteq \mathcal{X}$ contenant les points impliqués dans des contraintes et une contrainte c . L'adéquation de c à $\mathcal{M}$ est une fonction $ADQ_{\mathcal{M}}(c) : \mathcal{X}^{\text{arité}(c)} \rightarrow [0, 1]$ modélisant la confiance dans $c \models \mathcal{M}$ tel que $ADQ_{\mathcal{M}}(c) = 1$ si et seulement si c est parfaitement satisfaite dans $\mathcal{M}$.

Une contrainte c est *parfaitement satisfaite* dans un modèle $\mathcal{M}$ s'il n'est pas possible de mieux satisfaire c qu'avec la disposition courante de ses points. Pour donner l'exemple d'un arbre de décision, une contrainte ML est parfaitement satisfaite si les deux points de la contrainte sont dans la même classe et dans la même feuille. Cette situation correspond à une insatisfaction parfaite pour une contrainte CL. Un arbre qui place les deux points d'une contrainte ML dans la même classe, mais pas la même feuille aura une adéquation haute, mais pas maximale.

De même, une contrainte triplet aura une forte adéquation si le modèle classe le point de référence et le point positif ensemble tout en classant le point négatif dans un cluster éloigné ; dans le cas des contraintes d'étendue limitée, 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 $\text{SIM}(x_i, x_j) : \mathcal{X}^2 \rightarrow [0, 1]$ dépendant du fonctionnement de $\mathcal{M}$:

$$ADQ_{\mathcal{M}}(ML(x_i, x_j)) = \text{SIM}(x_i, x_j) \quad (8.1)$$

$$ADQ_{\mathcal{M}}(CL(x_i, x_j)) = 1 - ADQ_{\mathcal{M}}(ML(x_i, x_j)) \quad (8.2)$$

$$ADQ_{\mathcal{M}}(\text{Triplet}(x_{ref}, x_+, x_-)) = 0.5 * (1 + \frac{ap - an}{\max(ap, an)}) \quad (8.3)$$

où $ap = ADQ_{\mathcal{M}}(ML(x_{ref}, x_+))$ et $an = ADQ_{\mathcal{M}}(ML(x_{ref}, x_-))$. Cette mesure donne 0.5 si le point de référence x_{ref} est à égale distance de x_+ et de x_- . Elle augmente à mesure que x_+ se rapproche et/ou x_- 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. L'évolution de cette fonction est simulée dans la figure 8.1.

$$ADQ_{\mathcal{M}}(\text{Span}(S, \gamma)) = \begin{cases} 0 & \text{si } Sp(S) > \gamma \\ \binom{|S|}{2}^{-1} \sum_{x_i, x_j \in S} ADQ_{\mathcal{M}}(ML(x_i, x_j)) & \text{sinon} \end{cases} \quad (8.4)$$

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 notre travail peuvent toutes être exprimées en fonction de contraintes ML/CL, 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. 8.5). 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

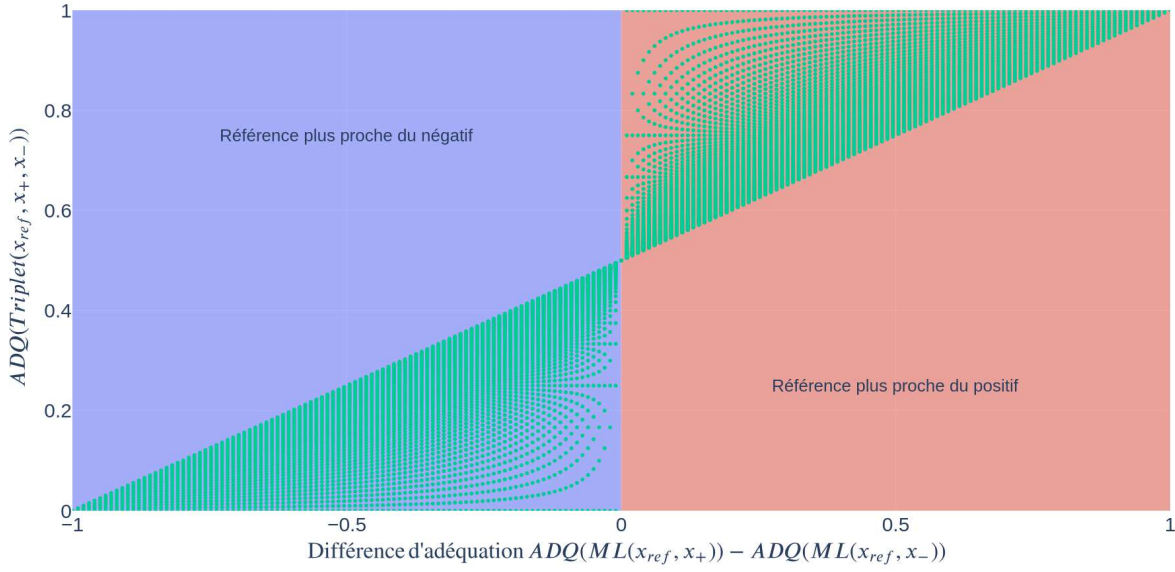


FIGURE 8.1 – Évolution de $ADQ_{\mathcal{M}}(\text{Triplet}(x_{ref}, x_+, x_-))$ en fonction de la différence $ap - an$ pour 10000 paires différentes de valeurs (ap, an) .

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 élevé 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.

$$LCA(x, y) = \frac{\text{profondeur}(x \cap y)}{\max(\text{profondeur}(x), \text{profondeur}(y))} \quad (8.5)$$

8.2 Modèle PPC pour choisir le MCS d'adéquation minimale

Puisque la mesure d'adéquation estime à quel point une contrainte est en accord avec les retours utilisateur passés, nous voulons retirer les contraintes ayant l'adéquation la plus faible pour résoudre les conflits.

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 reprenant les variables et contraintes décrites pour CP-MWCM (Section 6.1.3). Plus particulièrement, la construction du MCS revient à donner une assignation aux variables binaires Z réifiant les contraintes utilisateur. Dans une solution du COP, on obtient donc un MCS $\{c_i \mid Z_i = 0\}$ et un MSS $\{c_i \mid Z_i = 1\}$. Une fonction objectif pondérant les contraintes avec leur adéquation au modèle appris sur les retours utilisateurs oriente le solveur vers le MCS contenant les contraintes les moins adéquates :

$$\arg \min \sum_{i=0}^{|C|} \max(ADQ(c), \epsilon) \times \mathbb{I}(Z_i == 0) \quad (8.6)$$

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 COP à une itération t vient d'au moins une contrainte appartenant au lot L_t qui vient d'être obtenu. Le problème peut donc être restreint à l'ensemble des contraintes contenant au moins une des variables des contraintes de L_t :

$$E = \{c \in \mathcal{C}_t \mid \exists l \in L_t \text{ s.t. } \text{portée}(c) \cap \text{portée}(l) \neq \emptyset\} \quad (8.7)$$

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

$$\arg \min \quad \sum_{i=1}^{|E|} w_i (1 - Z_i) \quad (8.8)$$

$$\text{s.t.} \quad \sum_{j=1}^{|L_t|} (1 - Z_j) \geq 1 \quad (8.9)$$

$$Z_i \iff c_i, \forall c_i \in E \quad (8.10)$$

La contrainte 8.9 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.

Ce modèle peut également être utilisé pour tester la satisfiabilité d'un ensemble de contraintes : si la solution produit un MCS vide, il n'y a pas de conflit.

8.3 Expérimentations

Nous avons cherché à répondre à plusieurs **questions de recherche** concernant la mesure d'adéquation et le calcul de MCS optimal avec des expérimentations :

- **QR1** : quelle est l'influence du modèle sur l'adéquation ?
- **QR2** : comment se comporte l'adéquation en comparaison à d'autres mesures pour évaluer les contraintes ?
- **QR3** : quel est l'impact des contraintes retirées sur la performance de clustering ?
- **QR4** : le modèle peut-il être utilisé préventivement pour s'assurer que le COP est satisfiable et fournir directement une correction si ce n'est pas le cas, ou seulement en secours, après avoir échoué à modifier la partition ?
- **QR5** : quelle est l'utilité du calcul de MCS d'adéquation minimal dans une application réelle ?

8.3.1 Protocole expérimental

Comme CP-MWCM, le modèle PPC pour calculer le MCS d'adéquation minimale a été implémenté en Python 3.13 avec CPMpy et en utilisant le solveur CP-SAT. Nous fixons ϵ à 10^{-4} comme valeur minimale¹. Les expériences ont été menées sur un processeur 8 coeurs Intel i7 à 2.80 GHz et avec 32 Go de RAM sous Ubuntu 24.04.

La table 8.1 donne les paramètres des modèles testés et la définition de SIM pour ceux-ci. Pour le modèle bayésien, la définition de SIM est :

$$\text{SIM}_{NB}(x_i, x_j) = \text{sim}_{\cos}(p_i, p_j) = \frac{\sum_{y \in \mathcal{Y}} P(y|x_i)P(y|x_j)}{\sqrt{\sum_{y \in \mathcal{Y}} P(y|x_i)^2} \sqrt{\sum_{y \in \mathcal{Y}} P(y|x_j)^2}} \quad (8.11)$$

avec $\mathcal{Y}$ l'ensemble des classes apprises par le classifieur. Pour le SVM, SIM est une mesure de similarité normalisée basée sur la fonction de décision f du SVM :

$$\text{SIM}_{SVM}(x_i, x_j) = \frac{1}{1 + \exp\left(-\frac{f(x_i) + f(x_j)}{2\rho}\right)} \quad (8.12)$$

avec ρ la marge du SVM. Le classifieur des k plus proches voisins compare x_i et x_j en calculant le nombre de voisins en commun, c'est-à-dire en calculant l'indice de Jaccard entre les groupes de voisins :

$$\text{SIM}_{KNN}(x_i, x_j) = \frac{|\mathcal{N}(x_i) \cap \mathcal{N}(x_j)|}{|\mathcal{N}(x_i) \cup \mathcal{N}(x_j)|} \quad (8.13)$$

avec $\mathcal{N}(x_i)$ l'ensemble des voisins de x_i . Enfin, la similarité dans une forêt aléatoire est calculée par vote :

$$\text{SIM}_{RF}(x_i, x_j) = \frac{1}{2} \left(\frac{\sum_{T \in RF} T(x_i) = T(x_j)}{|RF|} + \frac{\sum_{T \in RF} \text{leaf}(x_i) = \text{leaf}(x_j)}{|RF|} \right) \quad (8.14)$$

Le premier terme compte le nombre d'arbres plaçant x_i et x_j dans la même classe ; le second terme fait la même chose au niveau des feuilles.

Dans les expériences, le calcul de MCS est effectué *post-hoc*, quand la modification a échoué car le problème est insatisfiable. La modification de partition est effectuée avec CP-MWCM. Le modèle supervisé utilisé pour le calcul de l'adéquation est entraîné au début d'une nouvelle itération de CoActive, avant de récupérer le nouveau lot de contraintes. Nous faisons deux itérations : la première avec des contraintes issues de la vérité terrain des jeux de données ; la seconde génère un lot de contraintes conçu pour entrer en conflit avec le premier lot, nécessitant ainsi un calcul de MCS. Les résultats complets sont disponibles sur un dépôt public².

1. Depuis la version de 9.24 de CPMpy, des valeurs réelles peuvent être données directement dans la fonction objectif, évitant les habituelles transformations en entiers. Cependant, nous avons observé empiriquement que le solveur CP-SAT commence à donner de fausses solutions quand $\epsilon < 10^{-4}$. Cela peut être dû à un problème de représentation des nombres flottants.

2. <https://github.com/aymericb213/Minimally-adequate-MCS>

8.3. EXPÉRIMENTATIONS

Modèle	Paramètres	SIM
Arbre de décision	indice de Gini	LCA (Éq. 8.5)
Naive Bayes	Modèle gaussien <i>a priori</i>	similarité cosinus des probabilités <i>a posteriori</i> (Éq. 8.11)
SVM	noyau RBF	similarité basée sur les marges (Éq. 8.12)
KNN	5 plus proches voisins	indice de Jaccard (Éq. 8.13)
Forêt aléatoire	10 arbres 100 arbres	taux d'accord entre les arbres (Éq. 8.14)

TABLE 8.1 – Modèles supervisés testés, leurs paramètres, et la fonction utilisée pour SIM. Les paramètres non mentionnés utilisent la valeur par défaut de *scikit-learn*.

Nous avons testé différents cas représentant les différentes combinaisons de contraintes possibles parmi celles définies en Section 2.3.1 :

- un ensemble de contraintes ML/CL contenant un ou plusieurs cycles contenant une unique contrainte CL (Soenen et al., 2021) ;
- un ou plusieurs triplets en conflit avec des contraintes CL posées sur les mêmes points concernés par le triplet ;
- une contrainte d'étendue limitée 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 d'étendue limitée.

8.3.2 Impact du modèle sur l'adéquation

Nous comparons les valeurs d'adéquations obtenues avec les modèles de la table 8.1 (QR1, QR2).

Model	ML	CL	Triplet	Etendue limitée
DT	0.51 ± 0.03	0.79 ± 0.03	0.58 ± 0.01	0.21 ± 0.04
NB	0.77 ± 0.03	0.70 ± 0.03	0.60 ± 0.01	0.28 ± 0.04
SVM	0.75 ± 0.03	0.22 ± 0.03	0.49 ± 0.01	0.37 ± 0.05
KNN	0.03 ± 0.01	0.99 ± 0.01	0.01 ± 0.01	0.01 ± 0.01
RF10	0.14 ± 0.01	0.70 ± 0.03	0.12 ± 0.07	0.08 ± 0.01
RF100	0.01 ± 0.01	0.98 ± 0.01	0.09 ± 0.01	0.01 ± 0.01
Inf.	0.43 ± 0.03	0.20 ± 0.03	0.12 ± 0.01	0.43 ± 0.07

TABLE 8.2 – Adéquation et informativité pour différents modèles et types de contraintes, avec intervalle de confiance à 95%.

La table 8.2 montre plusieurs variations dans les valeurs moyennes en fonction des contraintes et des modèles. Les contraintes ML/CL ont tendance à avoir une plus grande adéquation, quel que soit le modèle. Cela montre que le choix du modèle n'est pas trivial : deux modèles différents peuvent mener à deux MCS distincts. En outre, certains modèles peuvent être plus adaptés à certains types de contraintes, comme ici les SVM pour les triplets.

La Figure 8.2 montre à quel point l'utilisation de différents modèles et définitions de SIM impactent le temps de calcul. En particulier, le SVM et la forêt aléatoire avec

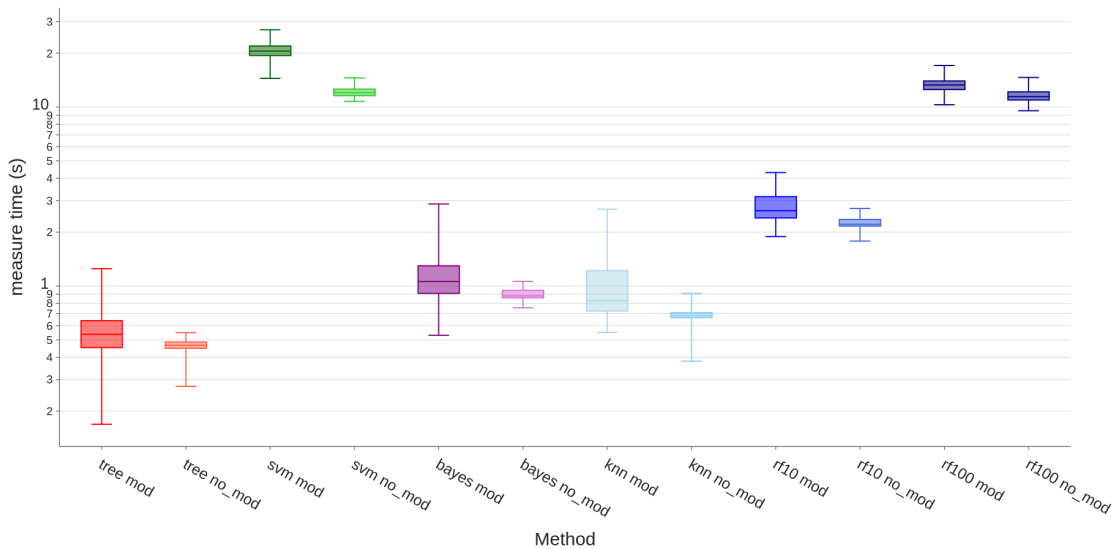


FIGURE 8.2 – Temps nécessaire pour mesurer l’adéquation de 100 contraintes de chaque type en utilisant divers modèles sur le jeu de données *yeast* (échelle logarithmique). La présence de *mod* dans le nom d’un modèle signifie que le calcul a été effectué après modification de la partition avec les contraintes.

100 arbres semblent être les modèles les plus lents, tandis qu’un simple arbre de décision termine les mesures en moins d’une seconde. La variance est plus faible quand les mesures sont faites après modification, quand toutes les contraintes sont satisfaites.

8.3.3 Impact du retrait des contraintes sur le clustering

Afin d’estimer comment le choix des contraintes à enlever impacte la partition obtenue (QR3), nous réalisons une modification avec CP-MWCM une fois le MCS retiré et comparons l’augmentation d’ARI par rapport à la vérité terrain selon la façon dont le MCS a été calculé. Nous comparons le MCS d’adéquation minimale avec deux références : le MCS de cardinalité minimale (c.à.d pas de pondération) et le MCS d’informativité minimale, en utilisant la mesure d’informativité de Davidson et al. (2006). Nous mesurons l’informativité par rapport à la partition courante de CoActive.

La table 8.3 montre que le choix du critère d’optimisation pour le MCS ne semble pas avoir un impact significatif sur la qualité de la partition modifiée : chaque critère mène à une augmentation de l’ARI grâce à CP-MWCM, et les différences sont très faibles.

Il semble donc que la différence entre les critères soit plutôt qualitative : minimiser la cardinalité peut donner des MCS différents à chaque exécution ou un MCS particulier parmi ceux de même cardinalité, dont l’obtention est déterminée par l’ordre de parcours des contraintes. L’informativité est une mesure binaire par rapport à un algorithme donné. Là encore, parmi plusieurs contraintes d’informativité nulle, plusieurs MCS sont possibles et le MCS obtenu va dépendre du parcours des contraintes. Enfin, l’adéquation donne une granularité plus fine en pouvant attribuer un score distinct à chaque contrainte et se base sur les retours utilisateurs passés.

Jeu de données	ARI_0	ARI_{ADQ}	ARI_{min}	ARI_{inf}
iris	0.730	0.741	0.738	0.752
wine	0.371	0.374	0.369	0.372
sonar	0.06	0.1	0.08	0.11
glass	0.270	0.292	0.293	0.311
ecoli	0.441	0.472	0.471	0.468
ionosphere	0.173	0.180	0.175	0.183
yeast	0.148	0.151	0.149	0.151
statlog	0.413	0.414	0.413	0.416

TABLE 8.3 – Évolution de l’ARI d’une partition initiale (ARI_0) après retrait d’un MCS minimisant (1) l’adéquation (ARI_{ADQ}), (2) la cardinalité (ARI_{min}) ou (3) l’informativité ARI_{inf} .

8.3.4 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 optimale soit trouvée en temps réel (QR4). 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 8.3 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 sous contraintes incrémentales, le temps d’exécution est de l’ordre de la seconde.

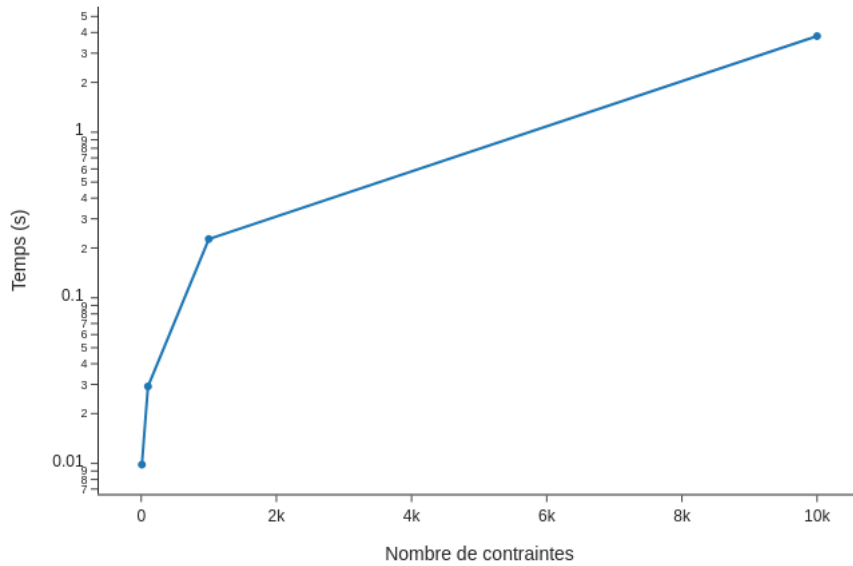


FIGURE 8.3 – Évolution du temps de calcul de MCS pour différentes tailles d’ensembles de contraintes (échelle logarithmique).

Ces résultats montrent non seulement que ce modèle est 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 $\mathcal{C}_t$ et produire directement un moyen de résoudre le conflit le cas échéant.

8.3.5 Cas d'étude : coupes de bois

Nous réalisons un test sur la série temporelle d'images satellite présentée en Section 2.5.3 (QR5).

Pour rappel, on dispose de 179 contraintes ML/CL issues de l'expertise humaine (Figure 2.8b). 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).

Nous testons deux cas :

- **Ajouter le bruit aux annotations.** Des conflits sont créés entre les contraintes des annotateurs et les contraintes bruitées. L'expérience consiste à observer quelles contraintes sont relâchées pour résoudre les conflits, entre l'annotation et le bruit.
- **Remplacer les annotations par du bruit.** Dans ce cas, la modification de partition est effectuée avec le bruit, ce qui va impacter l'apprentissage. Il n'y a pas de conflit ; on mesure l'adéquation des contraintes pour déterminer si les contraintes bruitées ont tendance à avoir une adéquation plus faible que les annotations.

Résultats. Le COP résout le problème en environ 0.02 secondes quel que soit le niveau de bruit. En variant la proportion de bruit de 5% à 95% du nombre d'annotations (cf. Table 8.4), on constate que plus de 75% du bruit est enlevé en mesurant l'adéquation à l'aide d'un arbre de décision. Dans le second cas, la différence entre annotations et bruit est marquée pour de faibles proportions de bruit et reste significative jusqu'à 90% de bruit remplaçant les annotations.

Taux de bruit (%)	5	10	15	20	25	30	35	40	45	50
Ajout (%)	100	78	74	83	71	76	75	79	73	81
Remplacement (pp)	62	49	40	38	32	31	28	26	34	32
Taux de bruit (%)	55	60	65	70	75	80	85	90	95	
Ajout (%)	75	83	80	78	76	76	76	77	77	
Remplacement (pp)	24	27	32	22	28	27	23	22	2	

TABLE 8.4 – Pourcentage de bruit corrigé (Ajout) ou différence entre l'adéquation moyenne des annotations et l'adéquation moyenne du bruit en points de pourcentage (Remplacement).

Ces expérimentations montrent l'intérêt que peut avoir la mesure d'adéquation dans notre cadre. CoActiVe peut maintenant gérer tout type de conflits avec une mesure basée sur l'historique des retours utilisateur et ne dépend plus de la fonction objectif de CP-MWCM comme c'était le cas avec la mécanique de relâchement de contraintes interne au modèle. On peut 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.

Chapitre 9

Conclusion

9.1 Contributions

Le but de cette thèse était d’étudier le clustering sous contraintes dans un contexte incrémental et interactif, c’est-à-dire quand un utilisateur est impliqué dans la construction du résultat en donnant régulièrement des indications sur ses souhaits sous forme de contraintes. Cela nous a amenés à formuler un cadre pour ce que nous appelons le clustering sous contraintes incrémentales, reposant sur une boucle entre la modification d’une partition des données et la réaction de l’utilisateur qui se traduit par l’ajout de nouvelles contraintes. Cela a donné lieu au système CoActiVe qui a été amélioré tout au long de la thèse avec nos contributions.

Le premier aspect sur lequel nous nous sommes concentrés est la similarité entre les résultats. C’est un problème peu étudié dans le clustering interactif. Pourtant, il nous semble primordial de donner les moyens à l’utilisateur de retracer son chemin depuis le début du processus de façon cohérente, et surtout de garantir que deux résultats successifs ne soient pas fondamentalement différents pour qu’un tel système de clustering puisse être utilisé dans des applications réelles. Nous avons donc cherché à concevoir un algorithme modifiant une partition existante pour satisfaire un ensemble de contraintes tout en minimisant les modifications apportées. Nous voulions également être capables de gérer des contraintes de nature diverse sans concevoir un algorithme ou une stratégie pour chaque contrainte. La flexibilité et l’expressivité de la programmation par contraintes nous ont paru être des atouts majeurs pour résoudre ce problème. De plus, les travaux antérieurs sur ce problème précis faisaient également usage d’approches déclaratives pour les mêmes raisons (Kuo et al., 2017 ; Nghiem et al., 2020). Nous avons repris l’objectif de Kuo et al. (2017) consistant à compter les points réaffectés entre l’entrée et la sortie du problème et nous avons ajouté un système de pondération par la distance au cluster d’arrivée des points réaffectés. Ainsi, un point est modifié **lorsque c’est nécessaire** pour satisfaire une contrainte ; il est alors affecté au **cluster le plus proche** permettant la satisfaction de la contrainte. Nous avons utilisé une méthode pour identifier un sous-ensemble de représentants d’un cluster (Guilbert et al., 2022) pour éviter les biais liés à l’utilisation de centroïdes ou de médoïdes dans le calcul de la distance au cluster. Les premières expérimentations ont montré que le modèle CP-MWCM remplissait l’objectif de maintenir la similarité entre l’entrée et la sortie, mais que cela rendait les modifications imperceptibles notamment sur de grands jeux

de données. Cela nous a poussés à ajouter un mécanisme de généralisation des modifications aux alentours des points réaffectés, avec des paramètres pour contrôler l’amplitude de l’effet.

Après avoir testé le modèle dans la boucle d’interaction, avec un algorithme de clustering actif fournissant des contraintes en se basant sur une partition déjà connue des données, nous avons fait une autre découverte : le facteur limitant du système en termes de temps n’est pas la modification, mais la sélection. Sur des jeux de données de grande taille, NPU (Xiong et al., 2014) prend plusieurs minutes pour faire chaque requête à l’utilisateur. D’autres algorithmes basés sur les voisinages sont plus rapides mais moins performants, et le principe des voisinages se peuplant au fur et à mesure des requêtes se marie bien avec le cadre incrémental. Nous avons donc étudié comment développer un algorithme de clustering actif qui exploite les spécificités de notre cadre de travail, en étant plus rapide que NPU tout en maintenant des performances similaires. Le résultat est l’algorithme REACT qui généralise les retours utilisateurs encodés par les voisinages avec un arbre de décision pour donner une prédiction du résultat désiré par l’expert(e). Bien qu’approximative étant donné que l’entraînement est réalisé sur très peu de points, l’intérêt vient de la confrontation entre cette prédiction et la partition existante pour sélectionner de nouvelles contraintes. Les expériences montrent que REACT est plus rapide d’un ordre de grandeur en moyenne que les autres méthodes basées sur les voisinages. De plus, l’arbre de décision permet de donner des règles décrivant les clusters d’après les retours utilisateur.

Enfin, nous avons examiné la gestion des conflits dans le système CoActiVe. Au début de nos travaux, les contraintes des itérations précédentes étaient considérées comme encodées implicitement dans la partition de l’itération courante. Cette stratégie simple comporte certaines limites. Par exemple, il n’était pas possible de garantir la satisfaction d’une contrainte que l’expert(e) juge très importante à chaque itération. Nous avons modifié CoActiVe pour garder en mémoire tous les lots de contraintes et être en mesure de récupérer une partie des contraintes pour les réinjecter dans la modification et vérifier qu’elles sont toujours satisfaites. Ce changement a introduit la possibilité de conflits dans nos expérimentations. Pour autant, des conflits étaient déjà possibles en application réelle via les deux sources concurrentes de contraintes (ajout manuel et clustering actif) ou un éventuel changement d’objectif de l’expert(e) pendant le processus. Nous avons alors étudié la résolution de conflits dans les ensembles de contraintes et notamment les ensembles de correction minimaux contenant les contraintes à enlever pour résoudre les conflits. Nous avons conclu qu’il était possible de calculer un ensemble optimal à présenter à l’utilisateur à partir de CP-MWCM, mais qu’il manquait une mesure capable de justifier la décision de retrait d’une contrainte avec les retours utilisateur. Nous avons donc proposé l’adéquation de contrainte à un modèle d’apprentissage, qui cible les contraintes qui semblent être incongrues par rapport aux retours utilisateurs. Cette mesure reprend l’idée de REACT d’entraîner un modèle sur le retour utilisateur et la généralise à tout modèle d’apprentissage supervisé. Nos expérimentations montrent que le calcul de l’ensemble de correction d’adéquation minimale est suffisamment rapide pour être effectué à chaque itération avant la modification, pour vérifier la satisfaisabilité du problème et proposer directement une correction le cas échéant.

9.2 Perspectives d'amélioration

Plusieurs axes d'amélioration de CoActive et de ses composantes peuvent faciliter son application dans des cas réels :

Clustering actif. REACT se base sur NPU pour l'initialisation ; c'est la phase la plus chronophage de l'algorithme. Remplacer cette phase par une méthode plus simple permettrait d'accentuer l'avantage de REACT en temps d'exécution, mais il faut également évaluer l'impact sur les performances. L'initialisation a souvent une importance capitale dans le fonctionnement d'un algorithme, comme le montre l'exemple de KMEANS.

En outre, un des objectifs initiaux de la thèse était de développer une méthode de clustering actif produisant des contraintes basées sur des connaissances thématiques du domaine d'application. Plusieurs approches ont été testées en ce sens avant de développer REACT mais ces pistes se sont montrées infructueuses. Notre contribution dans ce domaine produit des contraintes ML/CL comme le font la plupart des algorithmes de clustering actif. Une méthode permettant de proposer à l'utilisateur de valider des règles extraites par REACT- et donc d'évaluer quelles règles doivent être proposées et quand - est une piste à court terme pour la proposition de contraintes de haut niveau à partir des travaux effectués. La validation d'une règle pourrait par exemple servir à "verrouiller" l'affectation des points qui la suivent (Codén et al., 2017), ce qui revient à élaguer l'espace de recherche du clustering actif.

Une autre piste intéressante consiste à utiliser des approches symboliques d'apprentissage de contraintes. En particulier, l'acquisition de contraintes appliquée à la résolution interactive de problèmes fait l'objet de plusieurs publications récentes (Bessière et al., 2023 ; Menguy et al., 2023 ; Tsouros et al., 2023). C'est un domaine qui partage des similitudes avec le clustering actif : les deux nécessitent de poser des requêtes à un humain et visent à limiter le nombre de requêtes posées tout en exploitant au mieux les réponses. Dans le cas général, les requêtes posées en acquisition de contraintes portent sur l'ensemble des variables d'un CSP et demandent à l'utilisateur si une instanciation globale de ces variables est une solution du CSP. Les travaux récents proposent de poser des requêtes partielles sur un sous-ensemble des variables d'un CSP pour élaguer l'espace des solutions possibles. Transposées dans notre cadre, les requêtes globales peuvent être reliées au retour utilisateur général sur la partition, tandis que les requêtes partielles correspondent à celles faites par le clustering actif. On pourrait donc imaginer une approche d'acquisition de contraintes englobant la boucle de CoActive et informant les étapes de modification et de sélection via l'espace de candidats restants (ici, les partitions possibles). En outre, ces approches peuvent apprendre tout type de contraintes dans un langage donné, ce qui peut éviter de devoir développer un algorithme pour chaque type de contraintes ou de dépendre de l'état de l'art largement concentré sur les contraintes ML/CL. D'autres approches comme la programmation logique inductive (Cropper et Dumančić, 2022) peuvent aussi se révéler utiles dans ce but.

Explicabilité du système. En tant que processus en interaction directe avec un humain, être capable d'expliquer les résultats obtenus et les décisions prises pour conserver l'implication et la confiance de l'utilisateur dans le système. Nous avons pris cela en compte dans le développement de REACT et dans la gestion des conflits entre contraintes, mais

cet effort peut être largement approfondi. Des travaux récents sur l'explicabilité d'arbres de décision (Audemard et al., 2022) et la construction d'explications de modèles sous forme de règles avec la PPC (Koriche et al., 2024) pourraient être intégrés à REACT pour renforcer l'interprétabilité de la méthode.

Par ailleurs, l'explicabilité conversationnelle (Galitsky, 2020 ; Zhang et al., 2025) permet à l'utilisateur de poser des questions sur les aspects qui l'intéressent ou de demander des précisions. Intégrer cette approche à la présentation des résultats pourrait donner à l'utilisateur la capacité de proposer des contraintes sur les explications, qui peuvent être difficiles à produire *ex nihilo* mais que le dialogue avec le système aiderait à concrétiser. De plus, l'accès à une série de résultats obtenus séquentiellement pourrait produire des explications plus faciles à appréhender pour un utilisateur sans expertise en intelligence artificielle. Ces explications pourraient par exemple se baser sur les changements d'affectation réalisés au cours du processus : "ce point était dans le cluster 1, mais a été réaffecté au cluster 3 lors de la cinquième itération pour satisfaire cette contrainte *must-link*".

Scénarios d'utilisation et interfaces humain-machine. CoActiVe n'a pas d'interface graphique dédiée très développée. Les partitions obtenues à chaque étape peuvent être visualisées avec les contraintes, mais les requêtes sont posées en ligne de commande ou fournies dans des fichiers texte. Développer une véritable interface dédiée est une condition nécessaire pour que le système soit utilisable pour des applications réelles.

Il serait également intéressant de (a) mettre en place un protocole expérimental pour évaluer l'expérience de l'utilisateur de CoActiVe (une fois franchie la barrière de l'interface) et/ou (b) modéliser des scénarios d'utilisation réalistes comme un utilisateur qui change d'avis sur la partition voulue au cours du clustering. De telles expérimentations pourraient être menées de concert avec des expert(e)s dans des domaines complexes et aboutir à des applications pratiques.

Annexe A

Implémentation de CoActiVe

CoActiVe est implémenté en Python 3.11. L'implémentation est construite pour être modulaire afin de respecter le caractère générique du framework.

Une classe abstraite donne l'implémentation de base des cinq éléments constitutifs du processus : initialisation avec un premier clustering, obtention du lot de contraintes par clustering actif, construction de l'ensemble de contrainte utilisé pour l'itération, modification de partition et résolution de conflit.

Deux classes héritent de cette classe abstraite : une version en ligne de commande faite pour les expériences, et une version reliée à une interface web et pensée pour une application dans un cadre réel.

Les implémentations sont adaptées à un cadre multi-vues et peuvent donc gérer plusieurs partitions d'un ou plusieurs jeux de données et recevoir des contraintes sur un ou plusieurs jeux de données. De par la nature incrémentale du processus, les données successives sont stockées au fur et à mesure. Les structures de données pour les partitions et les contraintes sont donc des tenseurs à trois dimensions : une pour les éléments des partitions, une pour les partitions de la même itération, et une pour les itérations elles-mêmes. De plus, un historique des changements apportés aux partitions est stocké sous la forme d'une liste de dictionnaires, chaque dictionnaire contenant les modifications apportées aux différentes partitions ; les modifications sont elles aussi représentées par des dictionnaires dont les clés sont les points modifiés et la valeur un tuple contenant le cluster de départ et le cluster d'arrivée.

Annexe B

NPU incrémental

L'algorithme ci-après est une version agnostique de NPU (Xiong et al., 2014) pour utilisation dans CoActive. La modification de l'algorithme consiste principalement à éviter l'étape de clustering sous contraintes et à simplement renvoyer un ensemble de contraintes à partir d'une partition donnée, qui sera dans notre contexte fournie par la boucle incrémentale.

Algorithme 30 NPU incrémental

Entrée: jeu de données $\mathcal{X}$, partition $\mathcal{P}$, budget de requêtes b

Sortie: ensemble de contraintes $\mathcal{C}$

```
1:  $\mathcal{C} \leftarrow \emptyset$ 
2:  $\mathcal{N} \leftarrow N_1 \mid N_1 = \{random(\mathcal{X})\}$ 
3:  $l \leftarrow 1$ 
4:  $x^* \leftarrow \text{MOSTINFORMATIVE}(\mathcal{X}, \mathcal{P}, \mathcal{N})$  ▷ Cf. Algorithme 21
5: pour chaque  $N_i \in \mathcal{N}$  par ordre décroissant de  $P(x^* \in N_i)$  faire
6:   Query  $x^*$  against any  $x_i \in N_i$  to the oracle
7:   si  $(x^*, x_i, ML)$  alors
8:      $\mathcal{C} \leftarrow (x^*, x_i, ML)$ 
9:      $N_i = N_i \cup x^*$ 
10:   break
11: sinon
12:    $\mathcal{C} \leftarrow (x^*, x_i, CL)$ 
13: si  $|\{c \in \mathcal{C} \mid c[2] = ML\}| = 0$  alors
14:    $l++$ 
15:    $N_l = x^*$ 
16:    $\mathcal{N} \leftarrow \mathcal{N} \cup N_l$ 
17: retourner  $\mathcal{C}$ 
```

Bibliographie

- Abin, A. et V.-V. Vu (2020). A Density-based Approach for Querying Informative Constraints for Clustering. *Expert Systems with Applications* 161, 113690.
- Abin, A. A. (2016). Querying Beneficial Constraints Before Clustering Using Facility Location Analysis. *IEEE Transactions on Cybernetics PP*, 1–12.
- Abin, A. A. (2018). A Random Walk Approach to Query Informative Constraints for Clustering. *IEEE Transactions on Cybernetics* 48(8), 2272–2283.
- Abin, A. A. (2019). Querying informative constraints for data clustering : An embedding approach. *Appl. Soft Comput.* 80, 31–41.
- Abin, A. A. et H. Beigy (2014). Active selection of clustering constraints : A sequential approach. *Pattern Recognition* 47(3), 1443–1458.
- Ackerman, M., S. Ben-David, et D. Loker (2010). Towards property-based classification of clustering paradigms. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, et A. Culotta (Eds.), *Advances in Neural Information Processing Systems 23 : 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada*, pp. 10–18. Curran Associates, Inc.
- Aggarwal, C. C. (Ed.) (2014). *Data Classification : Algorithms and Applications*. New York : Chapman and Hall/CRC.
- Andrieu, C., N. de Freitas, A. Doucet, et M. I. Jordan (2003). An Introduction to MCMC for Machine Learning. *Machine Learning* 50(1), 5–43.
- Angluin, D. (1988). Queries and Concept Learning. *Machine Learning* 2(4), 319–342.
- Apt, K. R. (2003). *Principles of constraint programming*. Cambridge University Press.
- Atwa, W. et M. Emam (2019). Improving Semi-Supervised Clustering Algorithms with Active Query Selection. *Advances in Systems Science and Applications* 19(4), 25–44.
- Atwa, W. et K. Li (2014). Active query selection for constraint-based clustering algorithms. In H. Decker, L. Lhotská, S. Link, M. Spies, et R. R. Wagner (Eds.), *Database and Expert Systems Applications - 25th International Conference, DEXA 2014, Munich, Germany, September 1-4, 2014. Proceedings, Part I*, Volume 8644 of *Lecture Notes in Computer Science*, pp. 438–445. Springer.

- Audemard, G., S. Bellart, L. Bounia, F. Koriche, J. Lagniez, et P. Marquis (2022). On the explanatory power of boolean decision trees. *Data Knowl. Eng.* 142, 102088.
en
- Awasthi, P., M.-F. Balcan, et K. Voevodski (2015). Local algorithms for interactive clustering.
- Bae, J., T. Helldin, M. Riveiro, S. Nowaczyk, M.-R. Bouguelia, et G. Falkman (2020). Interactive Clustering : A Comprehensive Review. *ACM Computing Surveys* 53(1), 1 :1–1 :39.
- Bailey, J. et P. J. Stuckey (2005). Discovery of Minimal Unsatisfiable Subsets of Constraints Using Hitting Set Dualization. In M. V. Hermenegildo et D. Cabeza (Eds.), *Practical Aspects of Declarative Languages*, Berlin, Heidelberg, pp. 174–186. Springer.
- Basu, S., A. Banerjee, et R. J. Mooney (2004). Active Semi-Supervision for Pairwise Constrained Clustering. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, ICDM'04, pp. 333–344. Society for Industrial and Applied Mathematics.
- Basu, S., M. Bilenko, et R. J. Mooney (2006). Probabilistic Semi-Supervised Clustering with Constraints. In B. Schölkopf, O. Chapelle, et A. Zien (Eds.), *Semi-Supervised Learning*, pp. 73–102. MIT Press.
- Basu, S., I. Davidson, et K. Wagstaff (2008). *Constrained Clustering : Advances in Algorithms, Theory, and Applications*. CRC Press.
- Basu, S., D. Fisher, S. M. Drucker, et H. Lu (2010). Assisting users with clustering tasks by combining metric learning and classification. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, Atlanta, Georgia, pp. 394–400. AAAI Press.
- Baum, E. et K. Lang (1992). Query learning can work poorly when a human oracle is used. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pp. 335–340.
- Beauchamp, A., T.-B.-H. Dao, S. Loudni, et C. Vrain (2022). Intégration incrémentale de contraintes pour le clustering avec la programmation par contraintes. In *Journées Francophones de Programmation Par Contraintes (Evènement affilié à PFIA 2022)*, Saint-Étienne, France.
- Beauchamp, A., T.-B.-H. Dao, S. Loudni, et C. Vrain (2023). Incremental Constrained Clustering by Minimal Weighted Modification. In R. H. C. Yap (Ed.), *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, Volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Toronto, Canada, pp. 10 :1–10 :22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- Beauchamp, A., T.-B.-H. Dao, S. Loudni, et C. Vrain (2024). Rule-Based Constraint Elicitation For Active Constraint-Incremental Clustering. In *2024 IEEE 36th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 798–805.

- Beauchamp, A., T.-B.-H. Dao, S. Loudni, et C. Vrain (2025). Ensemble de correction minimal basé sur l'adéquation des contraintes en clustering. In *Journées Francophones de Programmation Par Contraintes (Évènement affilié à PFIA 2025)*, Dijon, France.
- Benavoli, A., G. Corani, J. Demšar, et M. Zaffalon (2017). Time for a change : A tutorial for comparing multiple classifiers through Bayesian analysis.
- Bessiere, C., C. Carbonnel, A. Dries, E. Hebrard, G. Katsirelos, N. Lazaar, N. Narodytska, C. Quimper, K. Stergiou, D. C. Tsouros, et T. Walsh (2023). Learning constraints through partial queries. *Artif. Intell.* 319, 103896.
en
- Bessière, C., E. Hébrard, B. Hnich, Z. Kiziltan, et T. Walsh (2006). Filtering Algorithms for the NValue Constraint. *Constraints* 11(4), 271–293.
- Bilenko, M., S. Basu, et R. J. Mooney (2004). Integrating constraints and metric learning in semi-supervised clustering. In *ICML*, pp. 11–18.
- Biswas, A. et D. Jacobs (2014). Active Image Clustering with Pairwise Constraints from Humans. *International Journal of Computer Vision* 108(1), 133–147.
- Bogaerts, B., E. Gamba, J. Claes, et T. Guns (2020). Step-Wise Explanations of Constraint Satisfaction Problems. In *Frontiers in Artificial Intelligence and Applications*. IOS Press.
- Brinker, K. (2003). Incorporating diversity in active learning with support vector machines. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pp. 59–66. AAAI Press.
- Cai, L., T. Yu, T. He, L. Chen, et M. Lin (2016). Active Learning Method for Constraint-Based Clustering Algorithms. In B. Cui, N. Zhang, J. Xu, X. Lian, et D. Liu (Eds.), *Web-Age Information Management*, Lecture Notes in Computer Science, Cham, pp. 319–329. Springer International Publishing.
- Chen, D. et Y. Jin (2020). An Active Learning Algorithm Based on Shannon Entropy for Constraint-Based Clustering. *IEEE Access* 8, 171447–171456.
- Coden, A., M. Danilevsky, D. Gruhl, L. Kato, et M. Nagarajan (2017). A method to accelerate human in the loop clustering. In N. V. Chawla et W. Wang (Eds.), *Proceedings of the 2017 SIAM International Conference on Data Mining, Houston, Texas, USA, April 27-29, 2017*, pp. 237–245. SIAM.
- Cohn, D., L. Atlas, et R. Ladner (1994). Improving generalization with active learning. *Machine Learning* 15(2), 201–221.
- Cohn, D., R. Caruana, et A. McCallum (2001). Semi-Supervised Clustering with User Feedback. 4.
- Corani, G., A. Benavoli, J. Demšar, F. Mangili, et M. Zaffalon (2017). Statistical comparison of classifiers through Bayesian hierarchical modelling. *Machine Learning* 106(11), 1817–1837.

- Cornuéjols, A., C. Wemmert, P. Gañçarski, et Y. Bennani (2018). Collaborative clustering : Why, When, What and How. *Information Fusion* 39, 81–95.
- Craenendonck, T. V., S. Dumancic, E. V. Wolputte, et H. Blockeel (2018). COBRAS : interactive clustering with pairwise queries. In W. Duivesteijn, A. Siebes, et A. Ukkonen (Eds.), *Advances in Intelligent Data Analysis XVII - 17th International Symposium, IDA 2018, 's-Hertogenbosch, The Netherlands, October 24-26, 2018, Proceedings*, Volume 11191 of *Lecture Notes in Computer Science*, pp. 353–366. Springer.
- Cropper, A. et S. Dumančić (2022). Inductive logic programming at 30 : A new introduction.
- Culotta, A. et A. McCallum (2005). Reducing labeling effort for structured prediction tasks. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2, AAAI'05*, pp. 746–751. AAAI Press.
- en
- Dao, T.-B.-H., K.-C. Duong, et C. Vrain (2017). Constrained clustering by constraint programming. *Artificial Intelligence* 244, 70–94.
- Dao, T.-B.-H. et C. Vrain (2024). A review on declarative approaches for constrained clustering. *International Journal of Approximate Reasoning* 171, 109135.
- Dasgupta, S. et V. Ng (2009). *Single Data, Multiple Clusterings*.
- en
- Davidson, I. et S. S. Ravi (2005). Agglomerative Hierarchical Clustering with Constraints : Theoretical and Empirical Results. In *PKDD*, pp. 59–70.
- Davidson, I. et S. S. Ravi (2007). The complexity of non-hierarchical clustering with instance and cluster level constraints. *Data Mining and Knowledge Discovery* 14(1), 25–61.
- Davidson, I., S. S. Ravi, et M. Ester (2007). Efficient incremental constrained clustering. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Jose California USA, pp. 240–249. ACM.
- Davidson, I., S. S. Ravi, et L. Shamis (2010). A SAT-based framework for efficient constrained clustering. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2010, April 29 - May 1, 2010, Columbus, Ohio, USA*, pp. 94–105. SIAM.
- Davidson, I., K. L. Wagstaff, et S. Basu (2006). Measuring Constraint-Set Utility for Partitional Clustering Algorithms. In *Knowledge Discovery in Databases : PKDD 2006*, Lecture Notes in Computer Science, Berlin, Heidelberg, Germany, pp. 115–126. Springer.
- Ester, M., H.-P. Kriegel, J. Sander, et X. Xu (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, KDD'96*, Portland, Oregon, pp. 226–231. AAAI Press.

- Fowlkes, E. B. et C. L. Mallows (1983). A Method for Comparing Two Hierarchical Clusterings. *Journal of the American Statistical Association* 78(383), 553–569.
en
- Gagolewski, M. (2022). A Framework for Benchmarking Clustering Algorithms.
- Galitsky, B. (2020). Conversational Explainability. In *Artificial Intelligence for Customer Relationship Management*, Human–Computer Interaction Series (HCIS), pp. 415–445. Springer International Publishing.
- Gamba, E., B. Bogaerts, et T. Guns (2023). Efficiently explaining CSPs with Unsatisfiable Subset Optimization. *J. Artif. Intell. Res.* 78, 709–746.
- Gan, G., C. Ma, et J. Wu (2020). *Data Clustering : Theory, Algorithms, and Applications, Second Edition*. Mathematics in Industry. Society for Industrial and Applied Mathematics.
- Gançarski, P., T. Dao, B. Crémilleux, G. Forestier, et T. Lampert (2020). Constrained clustering : Current and new trends. In P. Marquis, O. Papini, et H. Prade (Eds.), *A Guided Tour of Artificial Intelligence Research : Volume II : AI Algorithms*, pp. 447–484. Springer.
- Gelman, A., J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, et D. B. Rubin (2013). *Bayesian Data Analysis, Third Edition*. CRC Press.
en
- Gonzalez, T. F. (1985). Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science* 38, 293–306.
- González-Almagro, G., D. Peralta, E. D. Poorter, J. R. Cano, et S. García (2025). Semi-supervised constrained clustering : an in-depth overview, ranked taxonomy and future research directions. *Artif. Intell. Rev.* 58(5), 157.
- Guilbert, M., C. Vrain, T.-B.-H. Dao, et M. C. P. de Souto (2022). Anchored Constrained Clustering Ensemble. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8.
- Guo, X., L. Gao, X. Liu, et J. Yin (2017). Improved deep embedded clustering with local structure preservation. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence, IJCAI’17*, Melbourne, Australia, pp. 1753–1759. AAAI Press.
- Harary, F. (1953). On the notion of balance of a signed graph. *Michigan Mathematical Journal* 2, 143–146.
en
- Hubert, L. et P. Arabie (1985). Comparing partitions. *Journ. of Class.* 2(1), 193–218.
- Junker, U. (2004). QUICKXPLAIN : preferred explanations and relaxations for over-constrained problems. In D. L. McGuinness et G. Ferguson (Eds.), *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, pp. 167–172. AAAI Press / The MIT Press.

- Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica* 4(4), 373–395.
- Klein, D., S. D. Kamvar, et C. D. Manning (2002). From Instance-level Constraints to Space-Level Constraints : Making the Most of Prior Knowledge in Data Clustering. In *Proceedings of the Nineteenth International Conference on Machine Learning, ICML '02*, San Francisco, CA, USA, pp. 307–314. Morgan Kaufmann Publishers Inc.
- Koriche, F., J. Lagniez, S. Mengel, et C. Tran (2024). Learning model agnostic explanations via constraint programming. In A. Bifet, J. Davis, T. Krilavicius, M. Kull, E. Ntoutsis, et I. Zliobaite (Eds.), *Machine Learning and Knowledge Discovery in Databases. Research Track - European Conference, ECML PKDD 2024, Vilnius, Lithuania, September 9-13, 2024, Proceedings, Part IV*, Volume 14944 of *Lecture Notes in Computer Science*, pp. 437–453. Springer.
- Kruskal, J. B. (1956). On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society* 7(1), 48–50.
- Kullback, S. et R. A. Leibler (1951). On Information and Sufficiency. *The Annals of Mathematical Statistics* 22(1), 79–86.
- Kumar, P. et A. Gupta (2020). Active Learning Query Strategies for Classification, Regression, and Clustering : A Survey. *Journal of Computer Science and Technology* 35(4), 913–945.
- Kuo, C., S. S. Ravi, T. Dao, C. Vrain, et I. Davidson (2017). A framework for minimal clustering modification via constraint programming. In S. Singh et S. Markovitch (Eds.), *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pp. 1389–1395. AAAI Press.
- Lafabregue, B. (2021). *Clustering et apprentissage profond sous contraintes pour l'analyse de séries temporelles - Application à l'analyse temporelle incrémentale en télédétection*. Ph. D. thesis, Université de Haute-Alsace.
- Lafabregue, B., P. Gançarski, J. Weber, et G. Forestier (2022). Incremental constrained clustering with application to remote sensing images time series. In *2022 IEEE International Conference on Data Mining Workshops (ICDMW)*.
- Lai, H. P., M. Visani, A. Boucher, et J.-M. Ogier (2014). A new interactive semi-supervised clustering model for large image database indexing. *Pattern Recognition Letters* 37, 94–106.
- Lampert, T., B. Lafabregue, T.-B.-H. Dao, N. Serrette, C. Vrain, et P. Gancarski (2019). Constrained Distance-Based Clustering for Satellite Image Time-Series. *IEEE Journ. of Sel. Top. in Appl. Earth Obs. and Remote Sens.* 12(11), 4606–4621.
- Lewis, D. D. et W. A. Gale (1994). A sequential algorithm for training text classifiers.
- Li, D., Z. Wang, Y. Chen, R. Jiang, W. Ding, et M. Okumura (2024). A Survey on Deep Active Learning : Recent Advances and New Frontiers.

- Liffiton, M. H. et A. Malik (2013). Enumerating Infeasibility : Finding Multiple MUSes Quickly. In C. Gomes et M. Sellmann (Eds.), *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Volume 7874, pp. 160–175. Berlin, Heidelberg : Springer Berlin Heidelberg.
- Liffiton, M. H. et K. A. Sakallah (2008). Algorithms for Computing Minimal Unsatisfiable Subsets of Constraints. *Journal of Automated Reasoning* 40(1), 1–33.
- Lu, Z. et Y. Peng (2013). Exhaustive and Efficient Constraint Propagation : A Graph-Based Learning Approach and Its Applications. *International Journal of Computer Vision* 103(3), 306–325.
- MacKay, D. J. C. (1992). Information-Based Objective Functions for Active Data Selection. *Neural Computation* 4(4), 590–604.
- MacQueen, J. (1967). Some Methods For Classification And Analysis Of Multivariate Observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, Volume 1, Oakland, CA, USA, pp. 281–297.
- Mallapragada, P. K., R. Jin, et A. K. Jain (2008). Active query selection for semi-supervised clustering. In *2008 19th International Conference on Pattern Recognition*, Tampa, FL, USA, pp. 1–4. IEEE.
English (US)
- Mehlhorn, K., B. Newell, P. Todd, M. Lee, K. Morgan, V. Braithwaite, D. Hausmann, K. Fiedler, et C. Gonzalez (2015). Unpacking the exploration-exploitation tradeoff : A synthesis of human and animal literatures. *Decision* 2(3), 191–215.
- Melville, P. et R. J. Mooney (2004). Diverse ensembles for active learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, New York, NY, USA, pp. 74. Association for Computing Machinery.
- Menguy, G., S. Bardin, N. Lazaar, et A. Gotlieb (2023). Active disjunctive constraint acquisition. In P. Marquis, T. C. Son, et G. Kern-Isberner (Eds.), *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023, Rhodes, Greece, September 2-8, 2023*, pp. 512–520.
- Mitchell, L. A. (2016). *INCREMENT - Interactive Cluster Refinement*. Ph. D. thesis, Brigham Young University.
- Nghiem, N., C. Vrain, et T. Dao (2022). 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*, pp. 174–190. Springer.
en
- Nghiem, N.-V.-D., C. Vrain, T.-B.-H. Dao, et I. Davidson (2020). Constrained Clustering via Post-processing. In *Discovery Science*, pp. 53–67.

- Nguyen, X. V., J. Epps, et J. Bailey (2009). Information theoretic measures for clusterings comparison : is a correction for chance necessary ? In A. P. Danyluk, L. Bottou, et M. L. Littman (Eds.), *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, Volume 382 of *ACM International Conference Proceeding Series*, pp. 1073–1080. ACM.
- Ouali, A., S. Loudni, Y. Lebbah, P. Boizumault, A. Zimmermann, et L. Loukil (2016). Efficiently finding conceptual clustering models with integer linear programming. In S. Kambhampati (Ed.), *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pp. 647–654. IJCAI/AAAI Press.
- Paparrizos, J. et L. Gravano (2016). K-Shape : Efficient and Accurate Clustering of Time Series. *SIGMOD Rec.* 45(1), 69–76.
- Prabakara Raj, S. R. et B. Ravindran (2013). Incremental Constrained Clustering : A Decision Theoretic Approach. In D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, J. Li, L. Cao, C. Wang, K. C. Tan, B. Liu, J. Pei, et V. S. Tseng (Eds.), *Trends and Applications in Knowledge Discovery and Data Mining*, Volume 7867, pp. 475–486. Berlin, Heidelberg : Springer Berlin Heidelberg.
- Qian, B., X. Wang, F. Wang, H. Li, J. Ye, et I. Davidson (2013). Active learning from relative queries. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI '13, Beijing, China*, pp. 1614–1620. AAAI Press.
- Régin, J.-C. (1994). A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)*, Volume 1 of *AAAI '94*, Seattle, Washington, USA, pp. 362–367. American Association for Artificial Intelligence.
- Rossi, F., P. van Beek, et T. Walsh (Eds.) (2006). *Handbook of Constraint Programming*, Volume 2 of *Foundations of Artificial Intelligence*. Elsevier.
- Rousseeuw, P. J. (1987). Silhouettes : A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics* 20, 53–65.
- Roy, N. et A. McCallum (2001). Toward Optimal Active Learning through Sampling Estimation of Error Reduction. In *Proceedings of the Eighteenth International Conference on Machine Learning, ICML '01, San Francisco, CA, USA*, pp. 441–448. Morgan Kaufmann Publishers Inc.
- Sakoe, H. et S. Chiba (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26(1), 43–49.
- Schultz, M. et T. Joachims (2003). Learning a distance metric from relative comparisons. In *Proceedings of the 17th International Conference on Neural Information Processing Systems, NIPS'03, Cambridge, MA, USA*, pp. 41–48. MIT Press.

- Settles, B. (2009). Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison.
- Settles, B., M. W. Craven, et S. Ray (2007). Multiple-instance active learning. In *Neural Information Processing Systems*.
- Seung, H. S., M. Oppen, et H. Sompolinsky (1992). Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory, COLT '92*, New York, NY, USA, pp. 287–294. Association for Computing Machinery.
- Shannon, C. E. (1948). A mathematical theory of communication. *Bell Systems Technical Journal* 27, 623–656.
- Soenen, J., S. Dumančić, T. Van Craenendonck, et H. Blockeel (2021). Tackling Noise in Active Semi-supervised Clustering. In F. Hutter, K. Kersting, J. Lijffijt, et I. Valera (Eds.), *Machine Learning and Knowledge Discovery in Databases*, Cham, pp. 121–136. Springer International Publishing.
- Song, Y., T. Wang, P. Cai, S. K. Mondal, et J. P. Sahoo (2023). A comprehensive survey of few-shot learning : Evolution, applications, challenges, and opportunities. *ACM Comput. Surv.* 55(13s), 271 :1–271 :40.
- Tax, D. M. et R. P. Duin (2004). Support Vector Data Description. *Machine Learning* 54(1), 45–66.
- Tsang, E. P. K. (1993). *Foundations of constraint satisfaction*. Computation in cognitive science. Academic Press.
- Tsouros, D. C., S. Berden, et T. Guns (2023). Guided Bottom-Up Interactive Constraint Acquisition. In *DROPS-IDN/Opus :19073*, Toronto, Canada. Schloss-Dagstuhl - Leibniz Zentrum für Informatik.
- Ultsch, A. et J. Lötsch (2020). The Fundamental Clustering and Projection Suite (FCPS) : A Dataset Collection to Test the Performance of Clustering and Data Projection Algorithms. *Data* 5(1), 13.
- Van Craenendonck, T., S. Dumančić, et H. Blockeel (2017). COBRA : A Fast and Simple Method for Active Clustering with Pairwise Constraints. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*.
- Van Craenendonck, T., W. Meert, S. Dumančić, et H. Blockeel (2018). COBRASTS : A New Approach to Semi-supervised Clustering of Time Series. In L. Soldatova, J. Vanschoren, G. Papadopoulos, et M. Ceci (Eds.), *Discovery Science*, Lecture Notes in Computer Science, Cham, pp. 179–193. Springer International Publishing.
- Von Luxburg, U., R. C. Williamson, et I. Guyon (2011). Clustering : Science or art ? In *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27, UTLW'11*, Washington, USA, pp. 65–79. JMLR.org.

- Vu, V.-V., N. Labroche, et B. Bouchon-Meunier (2010). An Efficient Active Constraint Selection Algorithm for Clustering. In *20th International Conference on Pattern Recognition, ICPR 2010, Istanbul, Turkey, 23-26 August 2010*, pp. 2969–2972.
- Vu, V.-V., N. Labroche, et B. Bouchon-Meunier (2012). Improving constrained clustering with active query selection. *Pattern Recognition* 45(4), 1749–1758.
- Wagstaff, K. et C. Cardie (2000). Clustering with instance-level constraints. In *Proceedings of the Seventeenth International Conference on Machine Learning, ICML '00*, San Francisco, CA, USA, pp. 1103–1110. Morgan Kaufmann Publishers Inc.
- Wagstaff, K., C. Cardie, S. Rogers, et S. Schrödl (2001). Constrained k-means clustering with background knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning, ICML '01*, San Francisco, CA, USA, pp. 577–584. Morgan Kaufmann Publishers Inc.
- en
- Wagstaff, K. L. (2007). Value, Cost, and Sharing : Open Issues in Constrained Clustering. In *Knowledge Discovery in Inductive Databases*, pp. 1–10.
- Wang, X. et I. Davidson (2010). Flexible constrained spectral clustering. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, KDD, pp. 563–572.
- Winston, W. L. et J. B. Goldberg (2004). *Operations research : applications and algorithms* (4th ed ed.). Australia ; Belmont, CA : Thomson Brooks/Cole.
- Wolsey, L., G. Cornuéjols, et G.-L. Nemhauser (1990). The uncapacitated facility location problem. *Discrete Location Theory*, 119–171.
- Wu, Y., I. Kozintsev, J. Bouguet, et C. Dulong (2006). Sampling strategies for active learning in personal photo retrieval. In *Proceedings of the 2006 IEEE International Conference on Multimedia and Expo, ICME 2006, July 9-12 2006, Toronto, Ontario, Canada*, pp. 529–532. IEEE Computer Society.
- Xiong, C., D. M. Johnson, et J. J. Corso (2016). Active Clustering with Model-Based Uncertainty Reduction. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39(1), 5–17.
- Xiong, S., J. Azimi, et X. Z. Fern (2014). Active Learning of Constraints for Semi-Supervised Clustering. *IEEE Transactions on Knowledge and Data Engineering* 26(1), 43–54.
- Xiong, S., Y. Pei, R. Rosales, et X. Z. Fern (2015). Active Learning from Relative Comparisons. *IEEE Transactions on Knowledge and Data Engineering* 27(12), 3166–3175.
- Xu, Z., R. Akella, et Y. Zhang (2007). Incorporating diversity and density in active learning for relevance feedback. In G. Amati, C. Carpineto, et G. Romano (Eds.), *Advances in Information Retrieval, 29th European Conference on IR Research, ECIR 2007, Rome*,

- Italy, April 2-5, 2007, Proceedings*, Volume 4425 of *Lecture Notes in Computer Science*, pp. 246–257. Springer.
- Yurtman, A., W. Meert, et H. Blockeel (2021). COBRAS+ : Reusing Previously Obtained Constraints in Active Semi-Supervised Clustering. In *33rd Benelux Conference on Artificial Intelligence and 30th Belgian-Dutch Conference on Machine Learning (BNAIC/BeneLearn 2021)*.
- Zhan, X., H. Liu, Q. Li, et A. B. Chan (2021). A comparative survey : Benchmarking for pool-based active learning. In Z. Zhou (Ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pp. 4679–4686. ijcai.org.
- Zhang, H., T. Zhan, S. Basu, et I. Davidson (2021). A Framework for Deep Constrained Clustering.
- Zhang, T., R. Ramakrishnan, et M. Livny (1996). BIRCH : An efficient data clustering method for very large databases. *SIGMOD Rec.* 25(2), 103–114.
- Zhang, T., M. Zhang, W. Y. Low, X. J. Yang, et B. A. Li (2025). Conversational Explanations : Discussing Explainable AI with Non-AI Experts. In *Proceedings of the 30th International Conference on Intelligent User Interfaces, IUI '25*, New York, NY, USA, pp. 409–424. Association for Computing Machinery.
- Zhong, G., X. Deng, et S. Xu (2019). Active informative pairwise constraint formulation algorithm for constraint-based clustering. *IEEE Access* 7, 81983–81993.

Clustering sous contraintes : intégration incrémentale et active des retours utilisateur

Résumé : Le clustering sous contraintes vise à exploiter les connaissances des experts lors de la tâche de clustering. Les algorithmes traditionnels supposent que toutes les contraintes pertinentes peuvent être fournies dès le départ, ce qui n'est pas réaliste. Dans cette thèse, nous proposons un cadre de clustering sous contraintes incrémentales, où l'utilisateur réagit aux résultats en ajoutant des contraintes progressivement jusqu'à obtenir satisfaction. Nous commençons par introduire un modèle en programmation par contraintes pour réaliser une modification minimale de partition pondérée par la distance des points modifiés à leur nouvelle affectation. Ce modèle assure la continuité des résultats tout au long du processus itératif, un aspect peu étudié dans le clustering interactif. Les résultats expérimentaux sur des jeux de données classiques montrent que, par rapport à l'état de l'art, notre approche exploite plus efficacement les contraintes pour se rapprocher d'une partition cible, tout en préservant la similarité entre les itérations. Nous proposons ensuite un algorithme de clustering actif adapté au cadre incrémental, fondé sur un modèle supervisé appris à partir des retours utilisateur pour suggérer de nouvelles contraintes. Notre algorithme affiche des performances comparables à l'état de l'art tout en étant généralement plus rapide et en apprenant progressivement la partition désirée. Enfin, nous explorons la gestion des conflits entre contraintes pouvant perturber le bon fonctionnement du système. Nous proposons une mesure d'adéquation de contrainte à un modèle d'apprentissage supervisé entraîné sur les retours utilisateur. Il est alors possible de déterminer les contraintes à relâcher en priorité en fonction de leur adéquation.

Mots clés : apprentissage non supervisé, clustering sous contraintes, clustering sous contraintes incrémentales, programmation par contraintes, clustering actif, ensemble de correction minimal, retours utilisateur

Constrained clustering : incremental and active feedback integration

Abstract : Constrained clustering aims to exploit expert knowledge during the clustering task. Traditional algorithms assume that all relevant constraints can be given *a priori*. In this thesis, we develop a framework for constraint-incremental clustering, where the user reacts to results and adds constraints accordingly until satisfaction. In the first part, we propose a constraint programming model for minimum weighted clustering modification that ensures the continuity of results during the iterative process, an overlooked aspect of interactive clustering. Experimental results on known datasets show that our model exploits constraints more effectively to get closer to a target partition than the state of the art, while preserving similarity between two iterations. In the second part, we propose an active clustering algorithm adapted to the incremental framework, which exploits past user feedback to propose new constraints. This algorithm achieves similar performance to the state of the art, while being faster on average and while learning an approximation of the desired partition. Lastly, we examine the case where conflicts arise between constraints, preventing the system from functioning properly. We propose a model-constraint adequacy measure that quantifies how well a constraint aligns with a supervised learning model trained on user feedback. It is then possible to determine which constraints should be relaxed first, based on their adequacy.

Keywords : unsupervised learning, constrained clustering, constraint-incremental clustering, constraint programming, active clustering, minimal correction subset, user feedback