# Clustering et apprentissage profond sous contraintes pour l'analyse de séries temporelles : application à l'analyse temporelle incrémentale en télédétection

Baptiste Lafabregue

# Université de Haute-Alsace

École doctorale n°269 : *Mathématiques, Sciences de l'Information et de l'Ingénieur*
Institut de Recherche en Informatique, Mathématiques, Automatique et Signal (EA 7499)

THÈSE

*présentée par*

**Baptiste Lafabregue**

soutenance prévue le 20 septembre 2021

pour obtenir le grade de : **Docteur de l'Université de Haute-Alsace**
Discipline: Informatique

# Clustering et apprentissage profond sous contraintes pour l'analyse de séries temporelles - Application à l'analyse temporelle incrémentale en télédétection.

Directeur de thèse : **Germain Forestier**
Co-directeur de thèse : **Pierre Gançarski**
Co-encadrant de thèse : **Jonathan Weber**

**Jury**

| | | |
|---|---|---|
| **Antoine Cornuéjols,** | Professeur des Universités à AgroParisTech | Rapporteur |
| **Dino Ienco,** | Chargé de recherches à l'INRAE | Rapporteur |
| **Thi-Bich-Hanh Dao,** | Maître de Conférences à l'Université d'Orléans | Examinateur |
| **Sébastien Lefèvre,** | Professeur des Universités à l'Université Bretagne Sud | Examinateur |

# Résumé

Les progrès dans les systèmes d'acquisition d'images satellitaires ont donné lieu à la génération d'une quantité d'images de télédétection sans précédent. Les satellites actuels réalisent désormais des captures d'images de la Terre avec une haute fréquence de revisite et une haute disponibilité. Ces séries d'images obtenues à différentes dates peuvent être considérées comme des séries temporelles. Leur analyse permet d'effectuer une observation continue de la Terre sur un large spectre avec des applications dans la cartographie agricole, la surveillance des catastrophes environnementales, etc. Cependant, ce phénomène ne se limite pas au domaine de la télédétection. On peut en effet observer une croissance similaire dans de nombreux domaines, tel que la médecine, le commerce de détail ou encore la finance. Que ce soit pour la télédétection ou les autres domaines, l'analyse de ces données fait face aux mêmes problématiques.

Une grande quantité de données n'est pas toujours accompagnée d'un étiquetage suffisant, ce qui empêche généralement une bonne application des méthodes supervisées. En effet l'étiquetage reste une tâche très chronophage, mais également complexe, car nécessitant une expertise sur les données analysées. Cette complexité est renforcée par la difficulté d'appréhender la dimension temporelle elle-même, surtout si elle est conjointe avec une dimension spatiale. A l'opposé, les méthodes non supervisées ne nécessitent pas de connaissances de l'expert mais donnent parfois des résultats médiocres ou éloignés des attentes de ce dernier.

Dans ce contexte, le clustering sous contraintes, qui est une classe d'algorithmes d'apprentissage semi-supervisé, est une alternative et offre un bon compromis en termes d'investissement pour l'expert. Toutefois, les méthodes de clustering sous contraintes sont sujettes à des limitations importantes sur la qualité des résultats obtenus. Nous montrons dans cette thèse que deux facteurs limites fortement l'impact des contraintes, la consistance, qui est la quantité d'information dans l'ensemble des contraintes que l'algorithme peut déterminer par ses propres biais, et la cohérence, qui est le degré d'accord entre les contraintes elles-mêmes.

Afin de répondre au problème de consistance, nous proposons une nouvelle méthode, I-Samarah, basée sur le clustering collaboratif et l'intégration des contraintes de manière incrémentale. Cependant, nous montrons également que le problème de cohérence reste un verrou scientifique important que nous proposons d'aborder de manière plus prospective avec des méthodes d'apprentissage de représentation basées sur l'apprentissage profond.

# Abstract

The advent of satellite imagery is generating an unprecedented amount of remote sensing images. Current satellites now achieve frequent revisits and high mission availability and provide series of images of the Earth captured at different dates that can be seen as time series. Their analysis allows continuous observation of the Earth on a wide spectrum with applications in agricultural mapping, environmental disaster monitoring, etc. However, this phenomenon is not limited to the field of remote sensing. Similar growth can be observed in many fields, such as medicine, retail, or finance. Whether it is for remote sensing or other domains, the analysis of these data faces the same issues.

A large amount of data does not always imply sufficient labeling, which generally prevents a good application of supervised methods. Indeed, labeling remains a very time-consuming task, but also a complex one, as it requires expertise on the analyzed data. This complexity is reinforced by the difficulty of understanding the temporal dimension itself, especially if it is combined with the spatial dimension. On the other hand, unsupervised methods do not require the expert's knowledge but sometimes give poor results or results that are far from the expert's expectations.

In this context, constrained clustering, which is a form of semi-supervised learning algorithms, is an alternative and offers a good compromise in terms of investment for the expert. However, constrained clustering methods are subject to important limitations on the quality of the obtained results. We show in this thesis that two factors strongly limit the impact of constraints, consistency, which is the amount of information in the set of constraints that the algorithm can determine by its own bias, and coherence, which is the degree of agreement between the constraints themselves.

In order to address the consistency problem, we propose a new method, I-Samarah, based on collaborative clustering and incremental integration of constraints. However, we also show that the consistency problem remains an important scientific challenge that we propose to address in a more prospective way with representation learning methods based on deep learning.

# Remerciements

Une thèse dure globalement trois ans. Trois ans pendant lesquelles on se lance sur un nouveau sujet, qu'on essaye d'aborder de manière scientifique en prenant du recule sur l'existant et en essayant de contribuer à l'état de la recherche en levant des questionnements ou verrous scientifiques.

Pour en avoir discuté avec mes collègues doctorants, cela nous mène irrémédiablement au fameux effet Dunning-Kruger. Dans l'idée de révolutionné la recherche, partant la fleur au fusil en haut de la *Montagne de la stupidité*, on tombe rapidement dans la *Vallée de l'humilité* quand on commence à vraiment comprendre les limites de l'état de l'art et qu'on essaye d'avoir un véritable questionnement/approche scientifique. La remonté vers le *Plateau de la consolidation* est quant à elle longue et fastidieuse.

Heureusement, une thèse ne se fait pas seul. C'est pour cela, qu'en avant propose je tiens à remercier tous ceux qui m'ont aidé de près ou de loin pendant ces trois années de thèse.

Je souhaite tout d'abord remercier mes encadrants de thèse, Germain Forestier, mon directeur de thèse, Pierre Gançarski, mon co-directeur de thèse, et Jonathan Weber, mon encadrant de thèse. Ils ont tous les trois étaient présents pour m'aider à mieux cerner mon sujet de recherche et à mieux comprendre ce qu'était vraiment le métier de chercheur, que ce soit à travers les retours sur mes travaux, et mes articles (et des fautes d'orthographe) et nos discussions (scientifiques ou non). Je remercie d'ailleurs particulièrement Pierre, qui s'est démené pour me trouver un financement de thèse.

Je remercie aussi Anne Puissant et les autres membres de l'ANR Times, pour les travaux qu'on a conduits ensemble et pour le demi-financement de ma thèse.

Sinon je souhaite également remercier mes collègues, que ce soit à Mulhouse avec notamment Gautier, Hassan, Maxime, Mounir, et Robin, ou à Strasbourg avec notamment Abdoul, Antoine, Bonan, Emmanuelle, Hussein, Jelica, Mihailo, Sarah et Tom, pour les longues discussions au RU, les nuits de l'info et les HIC fais ensemble.

Enfin je tiens à remercier particulièrement ma famille, mes parents et ma grand-mère, qui ont aussi était là pendant les moments de questionnements et surtout ma compagne, Amandine, qui était prêt à me revoir partir pour trois ans et toujours là pour me supporter.

# Table des matières

# Liste des figures

# Liste des tableaux

# Introduction

La prolifération croissante de données géospatiales, désormais produites en (quasi-)continu par des capteurs de plus en plus sophistiqués et nombreux, ouvre de nouveaux champs d'étude sur le suivi et l'analyse de l'évolution de notre environnement, comme par exemple le suivi agricole ou les changements des modes d'occupation et d'utilisation des sols. Cependant, l'accès aux données, bien qu'étant une condition nécessaire à l'analyse de ces phénomènes, n'est pas une condition suffisante. Il est donc nécessaire de proposer des approches adaptées pour accompagner les géographes dans l'analyse de ces données.

Par ailleurs, ce besoin ne se limite pas au domaine de la télédétection. On peut en effet observer une croissance similaire dans de nombreux domaines, tels que la médecine [Craddock et collab., 2012; Oliveira et collab., 2020], le commerce de détail [Abbasimehr et Shabani, 2019; Claeys et collab., 2020] ou encore la finance [Sezer et collab., 2020]. Que ce soit pour la télédétection ou ces autres domaines, l'analyse de ces données fait face aux mêmes problématiques.

Premièrement, l'utilisation d'approches classiques d'apprentissage supervisé requière des ensembles d'apprentissage étiquetés. Or, la création de ces données de référence, reposant largement sur l'annotation manuelle, reste un processus fastidieux et chronophage pour l'expert. De plus, même si des approches intermédiaires existent, reposant sur un petit nombre d'exemples [Forestier et Wemmert, 2016; Han et collab., 2018], elles font très souvent face à une absence ou une connaissance uniquement partielle des champs sémantiques (classes thématiques, nomenclatures, objets d'intérêt, ...) nécessaires pour qualifier correctement ces données, rendant de fait ces méthodes difficilement utilisables. Cette difficulté d'annotation est encore renforcée par la complexité de l'analyse de la dimension temporelle elle-même et la nouveauté de ce domaine d'étude pour l'expert.

Deuxièmement, même si les méthodes non-supervisées permettent de se passer de données de référence, elles peuvent aboutir à des solutions non pertinentes pour l'expert. Les clusters proposés peuvent être, par exemple, éloignés de classes thématiques connues par l'expert. En effet, même s'il n'existe pas une base de connaissances sous forme d'étiquettes, les experts thématiques ont souvent une connaissance informelle sur le sujet d'étude, que ce soit au travers d'intuitions ou de connaissances théoriques ou parcellaires. On parle souvent de *background-knowledge*, qu'on pourrait traduire par connaissance contextuelle. En télédétection cela peut prendre la forme d'une connaissance sur la phénologie de certaines plantes, sur le nombre d'espèces d'arbres dans une zone donnée ou encore sur la localisation de zones de croissance urbaine. Or les méthodes de clustering ne prennent pas en compte ces connaissances de l'expert et ne permettent pas à ce dernier d'agir sur le résultat obtenu.

Pour contourner ce problème, le clustering sous contraintes propose d'intégrer et de prendre en compte une partie de ces connaissances dans le processus de clustering. Cette connaissance est transmise sous la forme d'un ensemble de contraintes, le processus cherchant à obtenir un clustering qui satisfait au maximum ces contraintes tout en restant cohérent par rapport à la structure des données. De nombreuses méthodes existent déjà dans la littérature, mais se pose la question de leur applicabilité aux séries temporelles.

Les travaux effectués dans le cadre de cette thèse ont pour objectif de répondre à cette problématique à travers deux approches. D'une part, le clustering sous contraintes incrémental et d'autre part, l'apprentissage profond. Néanmoins pour répondre au besoin initial des géographes à l'origine de cette thèse, nous avons principalement validé nos approches dans le domaine de la télédétection.

Ce mémoire comporte cinq parties :

**Chapitre 1** La première partie introduit la problématique de l'analyse non supervisée de séries temporelles (section 1.1) et en quoi ce problème est en pratique difficile à aborder (section 1.2). Nous expliquons ensuite (section 1.3) en quoi une intégration, même partielle, des connaissance de l'expert peut faciliter cette analyse.

**Chapitre 2** Le second chapitre introduit l'utilisation de contraintes au sein du processus de clustering. Il présente les différents types de contraintes existantes (section 2.1) et les principales méthodes de clustering sous contraintes (sections 2.2 et 2.3). Ensuite, nous procédons à une évaluation des méthodes de clustering sous contraintes de l'état de l'art (section 2.4). Il en ressort deux verrous scientifiques principaux, que sont la consistance des méthodes et la cohérence des contraintes. Nous proposons alors une nouvelle approche incrémentale pour répondre à la problématique de consistance (section 2.5).

**Chapitre 3** Le troisième chapitre porte sur la validation de notre approche en télédétection. Après une présentation détaillée du contexte de la télédétection (section 3.1), ce chapitre expose les résultats obtenus dans ce domaine (sections 3.2 et 3.3). Enfin, nous commentons ces résultats (section 3.4).

**Chapitre 4** Ce chapitre, plus prospectif et consécutif à la mise en évidence des limites de notre approche, limites principalement liées à la représentation des données, tente de répondre à la problématique de cohérence de contraintes. Ces travaux consistent en l'adaptation de l'apprentissage profond au clustering de séries temporelles. Après avoir exposé l'intérêt d'utiliser ce type de méthodes dans notre contexte (section 4.1), nous présentons nos premières expériences sur des données de télédétection (section 4.2). Toutefois, ces premiers résultats ont mis en évidence certaines limitations des méthodes existantes (section 4.3).

**Chapitre 5** Ce dernier chapitre vise à pousser l'étude du chapitre précédent en menant une étude plus large sur les gains potentiels à utiliser de l'apprentissage de représentations par apprentissage profond pour le clustering de séries temporelles. Nous établissons d'abord un état de l'art des méthodes de clustering basées sur l'apprentissage profond (section 5.1). Puis nous présentons notre étude et les résultats obtenus (section 5.2), que nous complétons avec la proposition d'une méthode d'interprétabilité des résultats (section 5.3). Enfin, nous présentons nos conclusions sur cette étude (section 5.4).

# Chapitre 1

# Contexte

**Sommaire**

Avant d'aborder l'approche que nous avons développée, il est important de définir le contexte qui a motivé notre proposition. Pour cela, nous présenterons le clustering de séries temporelles et plus précisément les difficultés de traiter de telles données de manière non supervisée (section 1.1). Ensuite, nous exposerons en quoi la nature mal-posé du problème de clustering amplifie la difficulté d'une approche non-supervisée (section 1.2). Enfin, nous détaillerons l'angle par lequel nous nous proposons de l'aborder dans cette thèse (section 1.3).

## 1.1 Clustering et séries temporelles

Il existe un grand nombre de méthodes de clustering, qu'elles soient dites de regroupement (à plat ou hiérarchique) telles que KMEANS ou CHA, de formation de concepts tel que COBWEB ou à base de modèles de mélange tel qu'EM. Dans cette thèse, nous nous sommes uniquement intéressés aux méthodes de clustering et plus précisément à celles basées sur l'utilisation d'une distance/similarité. Néanmoins, nous pensons que la majorité de nos avancées devraient être aisées à adapter à ces autres approches.

### 1.1.1 Formalisation du problème

Soit $X$ un ensemble de $N$ objets :

$$X = \{x_1, ..., x_N\} \tag{1.1}$$

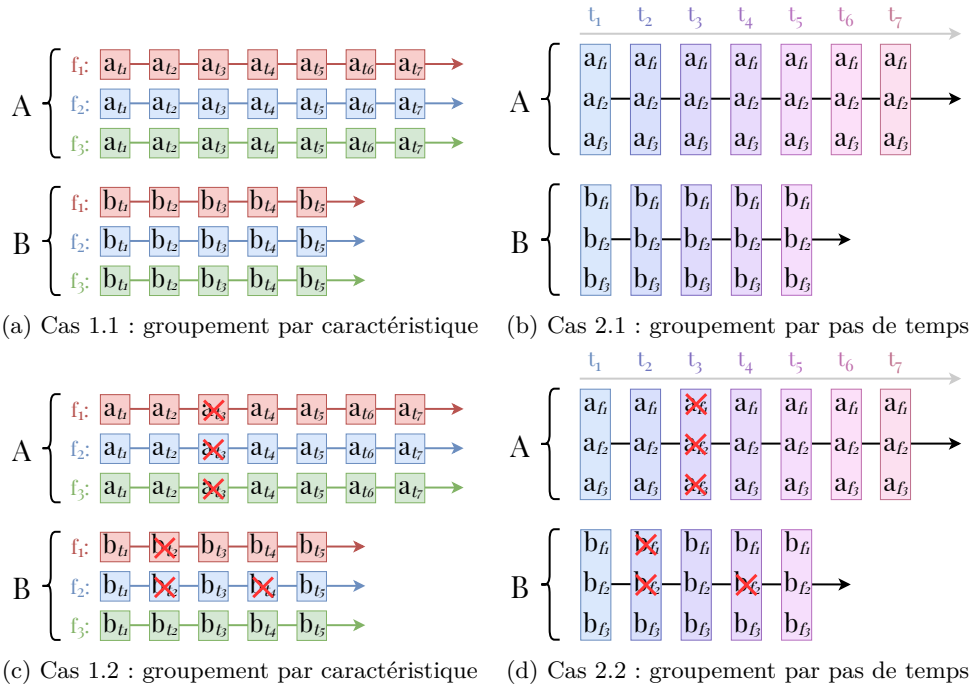
La tâche de clustering par regroupement consiste à trouver une partition de $X$, en $K$ clusters $c_1, \ldots, c_k$, qui à la fois maximise la similarité des objets appartenant à un même cluster et maximise la dissimilarité entre les objets de clusters différents.

Les séries temporelles sont des données spécifiques pour lesquelles chaque objet est vu comme une séquence ordonnée chronologiquement. Chaque élément de cette séquence dénote l'évolution de l'état de l'objet dans le temps. Cet objet peut avoir une ou plusieurs caractéristiques. Par exemple, pour suivre l'évolution de l'état d'un patient, le médecin peut uniquement suivre sa fréquence cardiaque, ou y ajouter sa pression artérielle ou encore son taux de sucre dans le sang. Dans la littérature on distingue souvent les séries temporelles univariées (une seule caractéristique) et multivariées (plus d'une caractéristique).

Dans le cas univarié, une série temporelle de longueur $T$ peut être définie comme une suite de valeurs réelles ou symboliques, représentée sous la forme d'un vecteur $x \in \mathbb{R}^T$ composé d'un ensemble de mesures synchrones. Dans le cas multivarié, cette série temporelle peut être considérée soit comme un ensemble de séries temporelles correspondant chacune à une mesure (a)synchrone d'une caractéristique différente, soit comme une suite de vecteurs représentant chacun l'état de l'objet à un pas de temps. La figure 1.1 illustre différentes représentations pour deux séries temporelles multivariées $A, B$. Le Cas 1.1 (figure 1.1a) illustre les séries par caractéristiques, c'est-à-dire que chaque caractéristique est indépendante des autres. Dans ce cas l'évolution de l'objet est représentée par $F$ séries temporelles indépendantes, $F$ étant le nombre de caractéristiques décrivant l'objet. Le Cas 2.1 (figure 1.1b) joint les caractéristiques dans un même vecteur et crée donc une relation entre elles par un regroupement par pas de temps. Dans ce cas, l'évolution de l'objet est représentée par une série temporelle où chaque pas de temps est décrit par $F$ caractéristiques. A noter qu'il est toujours possible de passer d'une représentation à une autre en créant des vecteurs (potentiellement incomplets) ou en décomposant les vecteurs en séries indépendantes.

En fonction de la représentation, le traitement des séries temporelles sera différent, notamment lorsqu'il s'agit de comparer deux séries. En effet, la distance entre $A$ et $B$ est calculée par :

$$D(A,B) = \begin{cases} \sum_{i=1}^{F} d_1(A_{f_i}, B_{f_i}), & (\text{ Cas 1.x}) \\ \sum_{i=1}^{T} d_2(A_{t_i}, B_{t_i}), & (\text{ Cas 2.x}) \end{cases} \tag{1.2}$$



(a) Cas 1.1 : groupement par caractéristique (b) Cas 2.1 : groupement par pas de temps

(c) Cas 1.2 : groupement par caractéristique (d) Cas 2.2 : groupement par pas de temps

FIGURE 1.1 – Différentes représentations des séries temporelles avec (a, b) ou sans valeurs manquantes (c, d).

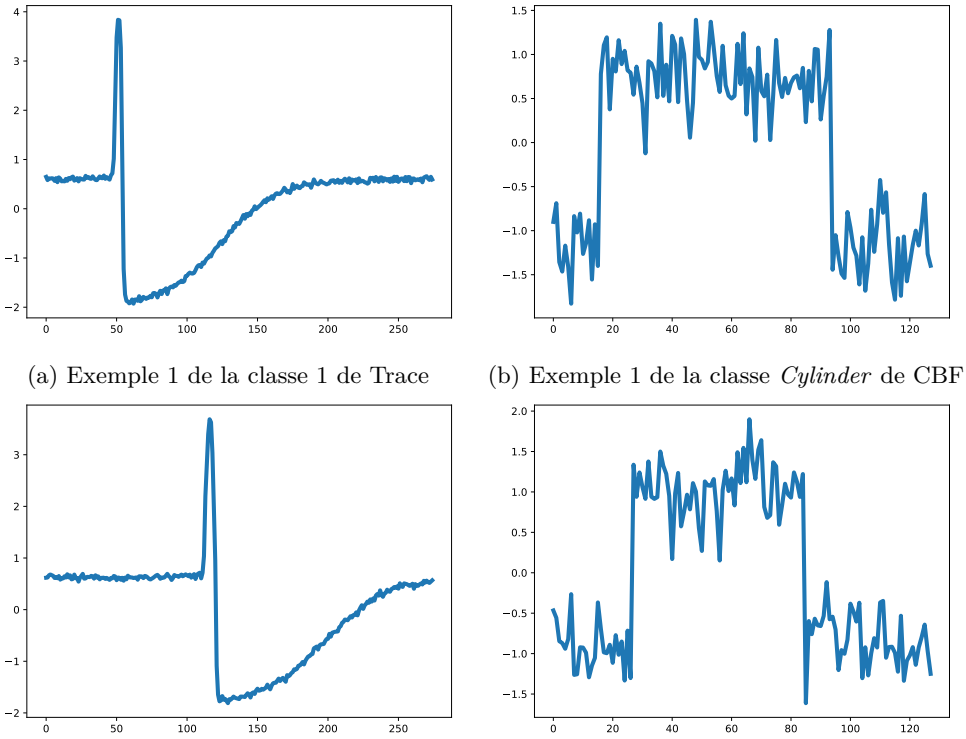
où $A_{f_i}$ représente une série temporelle des valeurs de la caractéristique $i$, $A_{t_i}$ un vecteur des différentes valeurs des caractéristiques au temps $t_i$, $d_1$ et $d_2$ des fonctions de distance. De ce fait, les deux distances peuvent aboutir à des résultats différents, $d_1$ se basant sur la comparaison de valeurs uniques tandis que $d_2$ se base sur la comparaison de vecteurs.

De plus cela pose également des problèmes sur la gestion des données manquantes. En effet, si on peut supposer que l'on dispose d'une fonction de distance $d$ qui gère des séries temporelles de longueurs différentes (par exemple par alignement), la représentation par séries indépendantes (Cas 1.2) peut toujours être traitée. Par contre, la représentation par série de vecteurs (Cas 2.2) ne peut adopter la même approche, car cela suppose que $d_2$ puisse gérer la comparaison de vecteurs dont certaines valeurs peuvent être manquantes, ce qui est rarement possible. Dans le Cas 2.2, il est donc généralement nécessaire d'effectuer un pré-traitement des données, soit par suppression des vecteurs aux pas de temps incriminés, soit par une imputation des valeurs manquantes (par exemple par interpolation).

Cependant, dans cette thèse, nous considérerons que les séries temporelles ne présentent pas de valeurs manquantes et nous adopterons la représentation du cas 2.1 (figure 1.1b). De plus, nous ferons référence aux séries temporelles comme étant par défaut multivariées et considérerons les évolutions d'une seule caractéristique comme un cas particulier où $F = 1$.

### 1.1.2 Dimension temporelle et similarité

Généralement, chaque classe d'un ensemble de séries temporelles correspond à une évolution d'un objet d'intérêt pour l'expert. Par exemple, dans le domaine médical, les électrocardiogrammes permettent de distinguer des populations souffrant de déformations ventriculaires, dans l'analyse de mouvements, l'évolution des coordonnées d'un bras permet d'identifier un geste particulier, ou dans notre cas de la télédétection, l'évolution de réponses spectrales permet d'identifier le type de culture d'une parcelle agricole. Selon le cas d'observation, l'évolution caractéristique d'une classe peut se produire plusieurs fois, être décalée dans le temps, ou/et être contractée/dilatée dans le temps (par exemples les champs de blé ne sont pas tous semés ou récoltés en même temps, les parcelles ne bénéficient pas du même ensoleillement, ...). L'évolution peut également être rapide (par exemple une augmentation soudaine



(a) Exemple 1 de la classe 1 de Trace

(b) Exemple 1 de la classe *Cylinder* de CBF

(c) Exemple 2 de la classe 1 de Trace

(d) Exemple 2 de la classe “Cylindre” de CBF

FIGURE 1.2 – Exemples de séries temporelles appartenant à la même classe mais avec un décalage (Figs. 1.2a and 1.2c) et un étirement (Figs. 1.2b and 1.2d)

du cours d'une action) ou étalée sur une longue période (par exemple le trafic routier quotidien). Les méthodes de clustering sont utilisées dans ce contexte pour créer des groupes contenant des séries temporelles représentant des évolutions semblables. Il se pose donc la question de l'évaluation de la similarité de deux séries temporelles

En clustering, et en analyse de données plus généralement, la méthode couramment utilisée est la distance euclidienne :

$$\begin{aligned} d_{eucl}(x_i, x_j) &= d_{eucl}(\begin{bmatrix} x_{i,1} \\ ... \\ x_{i,T} \end{bmatrix}, \begin{bmatrix} x_{j,1} \\ ... \\ x_{j,T} \end{bmatrix}) \\ &= \sqrt{(x_{i,1} - x_{j,1})^2 + ... + (x_{i,T} - x_{j,T})^2} \end{aligned} \tag{1.3}$$

Il découle de ce choix que chaque pas de temps est considéré comme une caractéristique indépendante des autres. Cela suppose un parfait alignement des deux séries temporelles dans le temps. Or nous avons vu précédemment que cette hypothèse est rarement présente dans les données réelles. De fait, une distorsion entre les deux séries temporelles (par exemple un décalage ou un étirement) peut résulter en une distance élevée dans l'espace euclidien [KEOGH et KASETTY, 2003]. Des exemples de séries temporelles pour lesquelles la distance euclidienne est très élevée alors que manifestement ces séries sont très similaires sont présentés dans la Fig. 1.2, où CBF est un ensemble de données synthétiques conçu pour distinguer trois formes, *Cylinder*, *Bell* et *Funnel*, et Trace est un ensemble de données synthétiques conçu pour simuler les pannes d'instrumentation dans une centrale nucléaire. Ces deux jeux de données sont extraits de l'archive de données UCR [DAU et collab., 2019].

Il est donc nécessaire que la méthode de clustering utilisée prenne en compte cette possibilité et soit capable de reconnaître des motifs dans les séries temporelles même s'ils sont décalés ou étirés dans le temps.

### 1.1.3 Une grande variété de mesure de similarité

Pour répondre au problème de mesure de (dis)similarité des séries temporelles, de nombreuses méthodes existent dans la littérature Aghabozorgi et collab. [2015]; Keogh et Kasetty [2003]; Laxman et Sastry [2006]. Cela s'explique en grande partie par la diversité des données et de leurs caractéristiques. Toutefois on peut distinguer trois éléments caractéristiques des méthodes de mesure de similarité pour les séries temporelles :

— Utilisation des données brutes ou projetées : la mesure de similarité peut prendre directement en compte la dimension temporelle de la donnée ou travailler sur une projection/représentation de ces données afin d'utiliser une méthode de distance standard ou pour simplifier l'utilisation d'une distance adaptée aux séries temporelles. Les méthodes de représentation permettent de projeter les données dans un nouvel espace, ce qui consiste souvent en une réduction de dimension et/ou en l'extraction de motifs/caractéristiques temporelles dans les données. Cette catégorie regroupe des méthodes qui ne sont pas forcément propres aux séries temporelles, telles que les méthodes basées sur des transformation, avec par exemple les Discrete Wavelet Transform (DWT) [Daubechies, 1992; Rhif et collab., 2019], ces méthodes consistant essentiellement en l'extraction de caractéristiques, notamment basée sur la fréquence. D'autres méthodes, comme Symbolic Aggregate ApproXimation (SAX) [Lin et collab., 2007], permettent de réduire les séries temporelles en une suite de segments plus simples à traiter. Enfin, il peut également être fait mention des méthodes basées sur l'extraction de modèles, tel que Hidden Markov Model [Panuccio et collab., 2002] ou sur l'utilisation de shapelets, avec par exemple la méthode Unsupervised Salient Subsequence Learning (USSL) [Zhang et collab., 2018].

— Utilisation de la série complète ou de sous-séries : habituellement l'ensemble des deux séries est utilisé lors du calcul de similarité. Cependant certaines méthodes proposent de répondre au problème de distorsion en extrayant une ou plusieurs sous-séquences similaires, qui servent alors de base de comparaison. Par exemple la méthode Longest Common Subsequence (LCSS) [Vlachos et collab., 2002] calcule la similarité en fonction de la longueur de la plus longue sous-séquence commune entre les deux séries. Plus largement, les méthodes basées sur les sous-séquences se focalisent sur la recherche de sous-séquences similaires. Par exemple la méthode DualMatch [Moon et collab., 2002] ou les travaux de Faloutsos et collab. [1994] consistent à découper les séries temporelles en plusieurs sous-séries disjointes, ce qui permet alors d'établir les séries temporelles les plus proches en utilisant des fenêtres glissantes (*sliding windows* en anglais).

— Une vision chronométrique ou chronologique : la vision chronométrique correspond à l'approche euclidienne, où le temps d'occurrence est pris en compte. Elle est donc souvent utilisée en conjonction avec une méthode de représentation (par exemple c'est le cas pour SAX [Lin et collab., 2007]). Dans la vision chronologique, l'ordre dans lequel les différentes variations des valeurs de la série s'enchaînent compte plus que leur temps précis d'occurrence. Les méthodes faisant le choix de cette vision proposent donc des mécanismes pour prendre en compte les potentiels distorsions temporelles entre deux séries. Les méthodes "élastiques" [Aref et collab., 2004], basées sur un recalage des séries sont les plus utilisées en pratique, avec notamment la mesure Dynamic Time Warpping (DTW) [Sakoe et Chiba, 1978]. D'autres méthodes se basent sur la corrélation entre les caractéristiques et sont spécifiques aux séries temporelles mutlivariées telle que SPCA qui est une similarité basée sur l'analyse par composante principale (PCA) Yang et Shahabi [2004] ou Compression-based Dissimilarity Measure (CDM) qui utilise la compression de la donnée [Keogh et collab., 2007].

Ces éléments peuvent être combinés. Cependant il existe souvent une incompatibilité entre l'utilisation de méthodes de représentation et de méthodes de mesure de similarité spécifiques

aux séries temporelles, les premières altérant souvent la dimension temporelle. De plus, cette pluralité de méthodes s'explique aussi par la diversité de formes des séries temporelles. Ainsi, pour les méthodes de représentation, la méthode Discrete Wavelet Transform sera très efficace pour l'élimination de bruit, mais risque fortement de donner de mauvais résultats sur l'étude d'un signal provenant d'un processus non-stationnaire. La méthode SAX permet une forte réduction de la dimensionnalité de la donnée à traiter mais au prix d'une approximation des données. Pour les mesures de similarité, DTW est une des méthodes les plus robustes, cependant, dans de nombreuses applications, la distance euclidienne reste toute à fait compétitive [Aghabozorgi et collab., 2015; Lkhagva et collab., 2006].

Cette diversité dans les séries temporelles rend d'autant plus difficile la formation de groupes pertinents de manière non-supervisée renforçant la nature mal-posé du clustering

## 1.2 Le clustering : un problème *mal-posé*

Le clustering est souvent présenté comme un problème *mal-posé* (*ill-posed* en anglais) [Domany, 1999; Jain et collab., 1999] du fait qu'il existe rarement une solution unique à la manière de regrouper les données. Ainsi, si on prend un ensemble de chiens et de chats, il peut être tout à fait pertinent de regrouper les sujets par couleur ou par espèce. Dans le cas des séries temporelles, on peut prendre en exemples les jeux de données contenus dans l'archive UCR. Il existe en effet plusieurs jeux de données identiques mais possédant un étiquetage des données différent. Par exemple, les ensembles de données GunPointAgeSpan, GunPointMaleVersusFemale et GunPointOldVersusYoung correspondent à un même ensemble d'enregistrements des mouvements de deux même acteurs à deux dates différentes (à 15 ans d'écart), mais visent respectivement à faire la distinction entre le type de geste (pointer un vrai pistolet, ou avec les doigts uniquement), le sexe de l'acteur et l'enregistrement le plus ancien par rapport au récent. De manière identique, les jeux de données DodgerLoopGame et DodgerLoopWeekend contiennent les mêmes données sur l'évolution du trafic routier dans une ville, mais visent à faire la distinction entre une journée avec un match dans un stade et pas de match pour le premier, et entre les jours de semaine et de week-end pour le second. Par conséquent, aucune méthode non supervisée ne peut fonctionner correctement sur un ensemble de données sans avoir de mauvais résultats sur les autres.

De ce fait il est nécessaire de pouvoir orienter le clustering en fonction du problème à résoudre et donc l'objectif de l'utilisateur.

## 1.3 Intégration de la connaissance de l'expert

De nombreuses approches permettent à l'expert humain d'incorporer directement des connaissances du domaine dans le processus de classification afin de le guider vers de meilleurs résultats. La manière la plus classique est de se tourner vers des méthodes supervisées. Cependant, nous avons vu précédemment que l'utilisation de ces méthodes n'est pas réaliste dans l'étude d'un nouveau domaine avec une grande quantité de données. En effet, mettre en évidence et formaliser de telles connaissances, tel que "simplement" établir le nombre de classes d'évolution potentielles, n'apparaît souvent pas réalisable à court ou moyen termes. En effet, pour effectuer cet étiquetage et en supposant les classes connues, l'expert doit identifier et annoter un grand nombre d'objets et ainsi déduire le label de chaque objet du jeu de données. En analyse temporelle, peu d'experts sont capables d'une telle performance, voire simplement volontaires pour cette tâche fastidieuse.

Le clustering peut alors être vu comme une manière de faciliter l'annotation en travaillant sur l'annotation de clusters plutôt que d'objets. Or, nous avons vu que les résultats obtenus peuvent être assez éloignés de ceux attendus par l'expert et donc ne pas permettre de séparer certaines classes. De plus, quel que soit la méthode utilisée et la qualité des résultats fournis,

il reste relativement difficile d'affecter directement une sémantique à chacun des objets ou clusters extraits.

Une solution intermédiaire est l'utilisation à la fois des données étiquetées et non étiquetées, on parle alors de classification semi-supervisée [Bair, 2013]. Cependant, pour que ces méthodes soient efficaces, elles requièrent souvent qu'une part substantielle des données soient annotées. De plus, elles ne répondent aucunement au problème de sémantisation des données en l'absence d'une typologie bien définie. En effet, ces méthodes nécessitent une connaissance sur les données à traiter tel que le nombre et la nature des classes à mettre en évidence.

Ainsi, pour répondre au problème d'intégration des connaissances de l'expert dans ce contexte, il est donc nécessaire de proposer une formalisation moins restrictive de ces connaissances et moins chronophage. L'utilisation de contraintes dans le processus de clustering répond à ce problème, notamment sur la formalisation. Les contraintes forment une liste d'énoncés que le résultat de clustering doit chercher à satisfaire. Les contraintes peuvent prendre des formes diverses mais elles ont généralement un poids sémantique moins fort qu'un étiquetage. L'expert peut ainsi indiquer que deux objets sont dissimilaires et devraient être séparés dans le résultat de clustering sans pour autant leur assigner une classe précise. Il peut également indiquer un nombre de clusters attendu, ou un intervalle des valeurs possibles, sans pour autant lister l'ensemble des classes devant se trouver dans le jeu de données.

Les méthodes reposant sur l'utilisation de contraintes sont appelées *méthodes de clustering sous contraintes*. Dans le chapitre suivant nous allons donc présenter et évaluer la pertinence de l'utilisation de ces méthodes dans notre contexte.

# Chapitre 2

# Guider le clustering avec des contraintes

**Sommaire**

Comme nous l'avons vu précédemment, la nature mal-posée du clustering rend la tâche de clustering complexe. En effet, plusieurs choix de regroupements cohérents peuvent exister pour un même jeu de données. Certains peuvent être plus proches des attentes de l'utilisateur. Il est donc important d'intégrer les connaissances de ce dernier dans le processus de clustering [Anand et collab., 1995]. Mais pour cela il est nécessaire de formaliser cette connaissance de manière à ce qu'elle soit exploitable par l'algorithme. Dans la suite nous allons donc présenter les contraintes utilisateur (section 2.1), un état de l'art des méthodes qui les utilisent (sections 2.2 et 2.3.1), l'évaluation de celles-ci (section 2.4) et notre proposition (section 2.5.3).

## 2.1 Typologie des contraintes

Communément, les contraintes peuvent être distinguées en deux grandes familles, les contraintes relatives aux objets eux-mêmes et celles relatives aux clusters [Basu et collab., 2008]. Les contraintes au niveau objet ont été introduites pour la première fois par Wagstaff et Cardie [2000]. Elles se résument essentiellement aux contraintes par paire, *Must-Link (ML)* et *Cannot-Link (CL)*, et aux *contraintes de label* :

- Une contrainte ML entre deux objets $o_i$ et $o_j$ est satisfaite si à l'issu du clustering $o_i$ et $o_j$ sont assignés au même cluster.
- Une contrainte CL entre deux objets $o_i$ et $o_j$ est satisfaite si à l'issu du clustering $o_i$ et $o_j$ sont assignés à des clusters différents.
- Une contrainte de label entre un ensemble $E$ d'objets est satisfaite si à l'issu du clustering, tous les objets appartenant à $E$ sont dans le même cluster.

Les contraintes au niveau des clusters permettent quant à elles de spécifier des caractéristiques qu'un ou plusieurs clusters doivent respecter, par exemple :

- le nombre de clusters $K$.
- leur cardinalité maximale ou minimale absolue ou relative.
- leur diamètre maximal $\gamma$ : les éléments d'un même cluster doivent avoir une dissimilarité d'au plus $\gamma$.
- leur division $\delta$ : les éléments de clusters différents doivent avoir une dissimilarité minimale de $\delta$.
- la $\epsilon$-contrainte : introduite par Davidson et Ravi [2005], impose que pour tout objet $o_i$ d'un cluster donné, il existe au moins un autre objet $o_j$ du même cluster tel que la dissimilarité entre $o_i$ et $o_j$ soit au maximum de $\epsilon$.

Les contraintes Must-Link et Cannot-Link sont les plus largement utilisées en pratique. Elles ont l'avantage de représenter un concept simple à appréhender pour l'utilisateur et ne nécessitent pas une connaissance complète de la donnée. De plus, elles peuvent facilement être déduites d'autres informations ou contraintes plus générales. Par exemple, une contrainte de diamètre maximal de cluster peut être traduite par l'ajout de contraintes CL entre les objets dont la dissimilarité est supérieure au diamètre maximal. Au cours de cette thèse, nous nous intéresserons principalement aux contraintes de type ML et CL et celles sur le nombre de clusters. Néanmoins, comme nous le verrons dans la suite, les méthodes que nous proposerons sont suffisamment génériques pour permettre une intégration rapide des autres contraintes sur les clusters ou sur les objets.

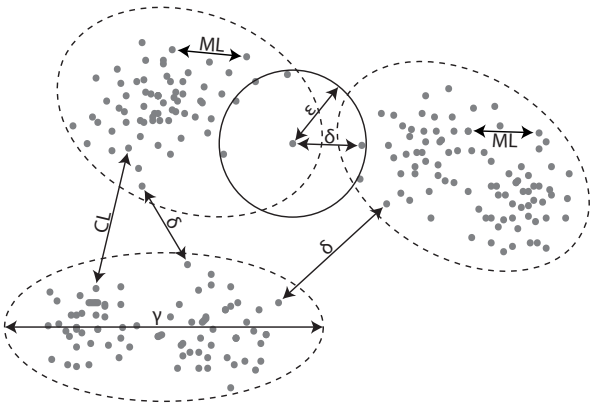



FIGURE 2.1 – Examples de contraintes ML, CL, $\delta$, $\gamma$, and $\epsilon$.

## 2.2 Clustering sous contraintes : les principales classiques

De nombreuse méthodes de clustering sous contraintes existent et peuvent être réparties dans deux famille principales, avec d'une part, les méthodes classiques monolithiques (basées sur un algorithme indépendant) et d'autres part, les méthodes dites par consensus ou ensemblistes (basées sur l'utilisation de plusieurs sous-méthodes de clustering). Nous exposons les méthodes monolithiques dans cette section, et celles par consensus dans la section suivante.

Pour les méthodes monolithiques on distingue différents types de méthodes :

**Les méthodes basées sur sur Kmeans :** On distingue généralement deux approches pour l'intégration des contraintes. D'un côté, celles qui modifient la stratégie d'affectation des clusters et de l'autre, celles qui modifient le critère d'optimisation de KMEANS. Dans la première approche, l'intégration des contraintes se fait en imposant le respect des contraintes par paires (ML et CL). Pour les méthodes COP-KMEANS [WAGSTAFF et collab., 2001] et CONSTRAINED-KMEANS [BASU et collab., 2002], chaque assignation d'un objet n'est validée que si elle respecte les contraintes données par l'utilisateur. Pour SEED-KMEANS [BASU et collab., 2002], les clusters initiaux sont initialisés par un calcul restreint aux données faisant l'objet d'une contrainte. Dans la seconde approche, les méthodes utilisent les contraintes par paire pour définir des pénalités dans la fonction d'optimisation, comme par exemple dans CVQE [DAVIDSON et RAVI, 2005], LCVQE [PELLEG et BARAS, 2007] et PCK-Means [BASU et collab., 2004]. Ces méthodes bénéficient de la robustesse et rapidité de l'algorithme KMEANS et sont souvent plus simples à mettre en œuvre. Toutefois, l'intégration de nouvelles contraintes n'est pas triviale car cela suppose de savoir comment la contrainte peut affecter l'assignation au cluster ce qui nécessite l'élaboration d'un traitement spécifique. Il a également été montré que ce type de méthodes est souvent enclin à un effet négatif de l'ajout de contraintes [DAVIDSON et collab., 2006]. De plus, certaines approches (COP-KMEANS et CONSTRAINED-KMEANS) ne garantissent pas la convergence et peuvent échouer si une contrainte ne peut être respectée.

**Les méthodes d'apprentissage de métrique :** Ces approches visent à apprendre automatiquement une mesure de dissimilarité à partir des données d'entraînement qui discrimine le mieux les échantillons des données. Les contraintes sont utilisées comme données d'entraînement pour apprendre la métrique. Deux ensembles de paires d'objets similaires et dissemblables sont construits à l'aide des contraintes ML et CL respectivement. Il existe plusieurs méthodes se basant sur différentes métriques, telles que la distance euclidienne et le chemin plus court [KLEIN et collab., 2002], la distance de Mahalanobis [BAR-HILLEL et collab., 2005], la divergence de Kullback-Leibler [COHN et collab., 2003] ou encore la méthode LRML (Laplacian regularizer metric learning)

pour le clustering d'images [Hoi et collab., 2010] ou même Dynamic Time Warping pour les séries temporelles [Salaou, 2020]. Même si ces méthodes permettent d'intégrer les contraintes au sein d'autres méthodes de clustering classiques, il est important de noter que ces algorithmes sont liés à une métrique de distance spécifique. Elles ne peuvent donc être appliquées telles quelles en cas d'utilisation d'une autre métrique.

**Les méthodes de clustering spectral :** Ces approches prennent en entrée une matrice de similarité pré-calculée et visent à minimiser un critère de coupe [Shi et Malik, 2000; Von Luxburg, 2007]. L'intégration des contraintes se fait en guidant l'espace de projection appris avec les contraintes, en modifiant soit la matrice d'affinité dérivée de la matrice de similarité [Kamvar et collab., 2003] soit la manière dont les vecteurs propres sont sélectionnés [Al-Razgan et Domeniconi, 2009; Li et collab., 2009; Wang et collab., 2014]. Le clustering spectral est souvent considéré comme supérieur aux algorithmes de clustering classiques, tels que Kmeans, car il est capable d'extraire des clusters de forme arbitraire [Von Luxburg, 2007]. Mais pour cela il est nécessaire de bien choisir les paramètres de calcul de la matrice de similarité, ce qui requiert souvent une supervision. De plus, même si l'agrégation spectrale a une complexité polynomiale, le temps de calcul de la matrice de similarité peut s'avérer assez long pour des grands jeux de données. Enfin, ces méthodes ne garantissent pas le respect de toutes les contraintes.

**Les méthodes déclaratives :** Ces approches offrent à l'utilisateur un cadre général pour formaliser le problème en choisissant une fonction d'optimisation et en énonçant explicitement les contraintes. Elles permettent la modélisation de différents types de contraintes utilisateur et la recherche d'une solution exacte définie comme un optimum global qui satisfait toutes les contraintes utilisateur. Les méthodes déclaratives sont généralement construites sur un outil d'optimisation général, tel que la programmation linéaire entière (ILP) [Babaki et collab., 2014], SAT [Davidson et collab., 2010], ou la programmation par contraintes (CP) [Dao et collab., 2017]. Alors que les autres approches se concentrent généralement sur les contraintes ML et CL au niveau de l'instance, les approches déclaratives utilisant CP ou ILP permettent une intégration directe des contraintes au niveau du cluster. Elles permettent également l'intégration de différents critères d'optimisation dans le même modèle d'apprentissage, tandis que d'autres approches sont généralement développées pour un critère d'optimisation particulier. Cependant ces méthodes sont sensibles à des incohérences dans les contraintes ce qui peut poser des problèmes de terminaison. De plus la recherche d'une solution optimale peut être très longue, même si en pratique le processus peut être arrêté sur une approximation de celle-ci.

## 2.3 Clustering sous-contraintes par consensus

L'abondance des méthodes de clustering trouvées dans la littérature peut s'expliquer par la nature mal-posée de ce problème, comme évoquée précédemment. En effet, différentes méthodes peuvent produire des résultats de clustering très différents à partir des mêmes données. De plus, un même algorithme peut produire des résultats différents en fonction de ses paramètres et de son initialisation. C'est pour cela que des méthodes multi-experts, basée sur la combinaison de plusieurs méthodes de clustering, ont été proposées. Parmi l'ensemble de ces méthodes, nous nous sommes intéressés à une approche en particulier qui se base sur le clustering collaboratif. En conséquence, dans un premier temps, nous allons faire une présentation générale des différentes méthode existantes, puis une présentation de la méthode sur laquelle nous allons travailler.

### Les méthodes existantes

A nouveau on peut découper ces approches en deux catégories :

**Les méthodes de clustering par ensemble :** Ces méthodes ont pour objectif d'améliorer la qualité globale du clustering en réduisant le biais de chaque classification [Hadjitodorov et Kuncheva, 2007]. Une méthode de clustering par ensemble est composée de deux étapes. Premièrement, plusieurs clusterings sont produits à partir de méthodes (souvent dénommées agents d'apprentissage) ayant des points de vue éventuellement différents. Ces méthodes peuvent être basées sur différents algorithmes de clustering différents [Strehl et Ghosh, 2002] (approches multi-stratégies) ou sur le même algorithme avec différentes valeurs de paramètres ou initialisations [Fred et Jain, 2002] (approches mono-stratégie). Elles peuvent aussi utiliser des sous-ensembles du jeu données. Le résultat final est dérivé des résultats indépendants par une fonction de consensus. On distingue deux cas d'utilisation des contraintes : chaque agent d'apprentissage les intègre directement [Yang et collab., 2012; Yu et collab., 2011b] ; ou les contraintes sont intégrées dans la fonction consensus [Al-Razgan et Domeniconi, 2009; Xiao et collab., 2016]. Dans le premier cas, le résultat final dépend fortement des agents d'apprentissage utilisés et de leur manière d'intégrer les contraintes. De plus cela restreint le clustering par ensemble à l'utilisation de méthodes de clustering sous contraintes. Dans le second cas, l'intégration au niveau de la fonction de consensus est souvent compliquée à mettre en oeuvre car les agents peuvent proposer des résultats incompatibles avec la satisfaction des contraintes.

**Les méthodes de clustering collaboratif :** Similaires au clustering d'ensemble, ces approches considèrent que les informations offertes par différentes sources et différents clusterings sont complémentaires [Kittler et collab., 1998]. Or, le processus de consensus dans le clustering d'ensemble fait face à deux difficultés majeures. D'une part, le nombre de clusters peut être différent entre les résultats des méthodes impliquées. D'autre part, il n'y pas nécessairement de lien de correspondance entre les clusters. De ce fait, l'utilisation d'une méthode de vote classique est rarement possible [Forestier et collab., 2010a]. Le clustering collaboratif consiste à faire collaborer plusieurs méthodes de clustering pour parvenir à un accord sur le partitionnement des données. Alors que le clustering d'ensemble se concentre sur la fusion des résultats du clustering, le clustering collaboratif se concentre sur la modification itérative des résultats des différents agents en partageant des informations entre eux [Gançarski et Wemmert, 2007; Pedrycz, 2002; Wemmert et collab., 2000]. Par conséquent, il étend le clustering par ensemble en ajoutant une étape de raffinement avant l'unification des résultats pour faciliter cette dernière étape. La collaboration prend souvent la forme d'un processus exploratoire guidé par une fonction d'optimisation favorisant la convergence et la qualité des résultats. Dans le clustering collaboratif, les contraintes utilisateur peuvent être intégrées à trois différents niveaux [Forestier et collab., 2010a] :

- au niveau du processus d'unification (par exemple en mettant plus de poids sur les résultats respectant les contraintes)
- au niveau de la phase de collaboration (par exemple en ajoutant une prise en compte des contrainte dans la fonction d'optimisation)
- au niveau des méthodes de clustering utilisées (en utilisant des méthodes de clustering sous contraintes comme agent)

Ces stratégies d'intégration peuvent s'appliquer séparément ou de manière combinée. Même les limitations liées au clustering par ensemble sont également présente au niveau du processus d'unification (résultats incompatibles avec la satisfaction des contraintes) et au niveau des méthodes de clustering (nécessité d'utiliser des méthodes de clustering sous contraintes), l'utilisation au niveau du processus collaboratif permet de conjointement favoriser le respect des contraintes et la convergence des résultats de clustering.

Cependant, ces méthodes peuvent engendrer des temps de calcul plus importants du fait qu'elles reposent sur des calculs itératifs de clustering, même si en pratique des optimisation permettent de réduire en partie ce surcoût (par exemple l'utilisation d'approximations).

### 2.3.1 Samarah : une méthode de clustering collaboratif sous contraintes

#### Un meta-classifieur

Le concept principal de Samarah [Forestier et collab., 2010a; Gançarski et Wemmert, 2005] est de faire collaborer différents algorithmes de clustering (agents) en les faisant échanger sur leur dissimilarité afin qu'ils parviennent, via des opérations simples, à faire converger leurs résultats de clustering, chaque agent apportant ses propres compétences. L'objectif de cette méthode est d'obtenir un clustering plus robuste et donc un clustering de meilleur qualité. De plus, la forte similarité obtenue entre les résultats des agents permet alors de simplifier leur unification par une fonction de consensus.
L'algorithme est divisé en trois étapes :

— Initialisation : chaque agent calcule un premier résultat de clustering ;
— Collaboration : raffinement mutuel des résultats de clustering des agents ;
— Unification : combinaison des résultats de clustering des agents.

La figure 2.2 et l'algorithme 1 résument le fonctionnement de cette méthode.

Dans la première étape (lignes 2 à 3 de l'algorithme 1), chaque agent applique sa méthode de clustering aux données. Chaque agent peut utiliser un algorithme différent, ou le même mais avec des valeurs de paramètres différentes (par exemple des hyperparamètres, une initialisation, un sous-ensemble de données, ...).

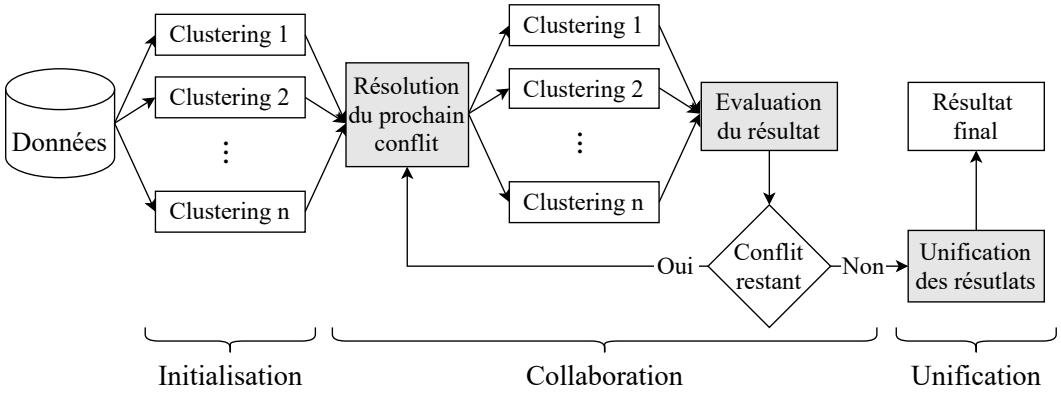



Figure 2.2 – La méthode Samarah

Dans la deuxième étape (lignes 4 à 14 de l'algorithme 1), les résultats calculés précédemment sont raffinés afin de résoudre les désaccords entre les résultats de clustering des différents agents. Les désaccords, également appelés *conflits*, sont calculés en comparant les clusters entre chaque couples d'agents. Pour cela il faut au préalable définir la notion de cluster correspondant.

Soient $\mathcal{R}_i$ le résultat de clustering de l'agent $i$ et $\mathcal{C}^i_{k1\leq k\leq n_i}$ l'ensemble des $n_i$ clusters de $\mathcal{R}_i$. Le cluster correspondant, $Corr(\mathcal{C}^i_k, \mathcal{R}_j)$ de $\mathcal{C}^i_k$ dans le résultat $\mathcal{R}_j$, $i \neq j$, est définit comme :

$$Corr(\mathcal{C}^i_k, \mathcal{R}_j) = arg \max_{\mathcal{C}^j_l \in \mathcal{R}_j} S(\mathcal{C}^i_k, \mathcal{C}^j_l), \tag{2.1}$$

où $S$ est la similarité inter-cluster entre deux clusters de deux résultats différents calculée par :

$$S(\mathcal{C}^i_k, \mathcal{C}^j_l) = \alpha^{j,i}_{l,k} \sum_{r=1}^{n_j} (\alpha^{j,i}_{k,r})^2, \tag{2.2}$$

**Algorithm 1** SAMARAH

1: **procedure** SAMARAH(Jeu de données $\mathcal{D}$, ensemble des $m$ agents $\mathcal{A}$, contraintes must-link ML $\subseteq \mathcal{D} \times \mathcal{D}$, contraintes cannot-link CL $\subseteq \mathcal{D} \times \mathcal{D}$)
2: **for** chaque agent $a_i$ dans $\mathcal{A}$ **do**
3: Calculer le clustering de $\mathcal{D}$ avec la méthode de l'agent $a_i$
4: Créer l'ensemble des **C**
5: Calculer $\Gamma$
6: $\Gamma_{max} = \Gamma$
7: **while C** non vide **do**
8: Choisir un conflit à résoudre dans **C**
9: Résolution du conflit localement avec les agents impliqués
10: Calculer $\Gamma_{max}$
11: **if** $\Gamma > \Gamma$ **then**
12: $\Gamma_{max} = \Gamma$
13: Appliquer la modifications aux agents impliqués
14: Calculer l'ensemble des nouveaux conflits et mettre à jour **C**
15: Calculer le résulter unifié depuis ceux des agents avec un algorithme de vote.

où $\alpha_{k,l}^{j,i}$ est la valeur de la matrice de confusion entre les résultat de clustering $\mathcal{R}_i$ et $\mathcal{R}_j$, définie pour la ligne $k$ et la colonne $l$ par :

$$\alpha_{k,l}^{j,i} = \frac{|\mathcal{C}_k^i \cap \mathcal{C}_l^j|}{|\mathcal{C}_k^i|} \tag{2.3}$$

L'ensemble des conflits est alors calculé par (ligne 4 de l'algorithme 1) :

$$\mathbf{C} = \{(C_k^i, C^j) : i \neq j, \omega^{i,j} < 1\} \tag{2.4}$$

où $\omega_k^{i,j} = S(\mathcal{C}_k^i, Corr(\mathcal{C}_k^i, \mathcal{R}_j))$.

Une fois **C** calculé, l'objectif est de modifier les résultats pour réduire ces différences, c'est-à-dire résoudre les conflits [FORESTIER et collab., 2010b]. Un conflit $\mathcal{K}_k^{i,j}$ est sélectionné dans l'ensemble des conflits selon un score d'importance :

$$Score\_conflit(\mathcal{K}_k^{i,j}) = 1 - \omega_k^{i,j} \tag{2.5}$$

Différentes possibilités existent pour l'ordre de sélection (par ordre ascendant, descendant ou au hasard). L'ordre est définit par l'expert à l'initialisation. L'algorithme effectue alors une résolution locale du conflit $\mathcal{K}_k^{i,j}$ qui consiste à appliquer un opérateur sur chaque résultat impliqué dans le conflit, $\mathcal{R}_i$ et $\mathcal{R}_j$, pour essayer d'améliorer leur similarité. Les opérateurs (dans la version actuelle de SAMARAH) pouvant être appliqués à un résultat sont :

— *fusion* de clusters ;

— *scission* d'un cluster en sous-clusters ;

— *reversement* des objets d'un cluster dans les autres clusters.

L'opérateur à appliquer est choisi en fonction du nombre de clusters impliqués dans le conflit (algorithme 2). Ce nombre est définit comme la cardinalité de l'ensemble des clusters $k \in \mathcal{R}_i$ tel que $\omega_k^{i,j} > p_{cr}$, où $0 \leq p_{cr} \leq 1$ est donné par l'utilisateur. Le paramètre $p_{cr}$ représente le pourcentage au-dessus duquel l'intersection entre les deux clusters est considérée comme significative. Par exemple, $p_{cr} = 0.2$ signifie que si $\mathcal{C}_k^i \cup \mathcal{C}_l^j$ représente moins de 20% des objets de $\mathcal{C}_k^i$, alors $\mathcal{C}_l^j$ n'est pas considéré comme un représentant significatif de $\mathcal{C}_k^i$.

Cette résolution locale peut être vue comme une remise en cause de chaque résultat en fonction des informations fournies par les autres agents de la collaboration. A noter que pour

**Algorithm 2** Application des opérateur de résolution de conflit

1: **procedure** APPLY_OPERATOR(Conflit à résoudre $\mathcal{K}_k^{i,j}$, Résultat de l'agent $i$ $\mathcal{R}_i$, Résultat de l'agent $j$ $\mathcal{R}_j$)
2: $\kappa = \{\mathcal{C}_l^j, \forall \mathcal{C}_l^j \in \mathcal{R}_j : S(\mathcal{C}_k^i, \mathcal{C}_l^j) > p_{cr}\}$
3: **if** $|\kappa| > 1$ **then**
4: $\mathcal{R}_{i'} = \mathcal{R}_i \setminus \{\mathcal{C}_k^i\} \cup scission(\mathcal{C}_k^i, |\kappa|)$
5: $\mathcal{R}_{j'} = \mathcal{R}_j \setminus \kappa \cup fusion(|\kappa|)$
6: **else**
7: $\mathcal{R}_{i'} = reversement(\mathcal{R}_i \setminus \{\mathcal{C}_k^i\})$
**return** $\mathcal{R}_i, \mathcal{R}_{i'}, \mathcal{R}_j, \mathcal{R}_{j'}$

un même conflit, différentes résolutions peuvent être proposées, en effet on peut retenir une des combinaisons parmi $\{(\mathcal{R}_i, \mathcal{R}_j), (\mathcal{R}_i, \mathcal{R}_{j'}), (\mathcal{R}_{i'}, \mathcal{R}_j), (\mathcal{R}_{i'}, \mathcal{R}_{j'})\}$ (obtenues par l'algorithme 2). Seule la combinaison qui maximise le critère de collaboration locale est retenue. Pour cela, le *critère local* est défini comme une combinaison entre la similarité des résultats de clustering et la qualité interne de chaque résultat. La *qualité interne* est une mesure du résultat du clustering propre à chaque agent tel que pour KMEANS l'inertie, ou la compacité [GANÇARSKI et WEMMERT, 2005] ou pour Cobweb la profondeur et la "qualité" des feuilles. Le critère local entre deux agents $i$ et $j$ est défini comme suit :

$$\gamma^{i,j} = \frac{1}{2}(p_s.(\frac{1}{n_i}\sum_{k=1}^{n_i}\omega_k^{i,j} + \frac{1}{n_j}\sum_{k=1}^{n_j}\omega_k^{(j,i)}) + p_q.(\delta^i + \delta^j)), \text{ où } p_s + p_q = 1 \tag{2.6}$$

où $\delta_i$ est la qualité interne du clustering de l'agent $i$, et $p_s$, $p_q$ sont des poids qui permettent à l'utilisateur d'équilibrer l'influence de la similarité et de la qualité interne. Après la sélection de la meilleure combinaison, son impact est évalué sur le clustering global (ligne 9 de l'algorithme 1). Le *critère global* de qualité est défini comme la moyenne des critères locaux entre l'ensemble des $m$ agents :

$$\Gamma = \frac{1}{m}\sum_{i=1}^{m}\Gamma^i, \quad \text{où} \quad \Gamma^i = \frac{1}{m-1}\sum_{j=1,j\neq i}^{m}\gamma^{i,j} \tag{2.7}$$

Si la modification apportée résulte en une amélioration du critère global $\Gamma$, alors les agents sont mis à jour, sinon la modification est annulée. En pratique un seuil de tolérance de dégradation potentielle, avant de revenir en arrière. Ensuite, l'algorithme passe à la résolution du conflit suivant. S'il ne reste plus aucun conflits à résoudre, le processus de collaboration s'arrête.

Lors de la troisième et dernière étape (ligne 14 de l'algorithme 1), les résultats raffinés sont combinés pour obtenir un résultat final unique. Pour cela une méthode de vote est appliquée sur les clusters obtenus.

### Intégration des contraintes

FORESTIER et collab. [2010a] ont proposé une adaptation de SAMARAH pour intégrer des contraintes dans le processus collaboratif. Ces connaissances sont intégrées au critère global $\Gamma$. Le critère est étendu pour prendre en compte les contraintes lors de la résolution des conflits (voir figure 2.3). Pour cela, une fonction mesurant le taux de satisfaction de la contrainte à intégrer dans la solution de l'agent $n$ ($\mathcal{R}_n$) sur la plage $[0, 1]$ a été définie. Par exemple, dans le cas d'un ensemble de contraintes Must-Link, $ML$, et Cannot-Link, $CL$, le calcul du taux de satisfaction consiste à mesurer la fraction de contraintes respectées, par :

$$\theta^n = \frac{1}{|ML| + |CL|}\left(\sum_{(r,s)\in \text{ML}} v_{ML}(\mathcal{R}_c, r, s) + \sum_{(r,s)\in \text{CL}} v_{CL}(\mathcal{R}_c, r, s)\right), \tag{2.8}$$

où

$$v_{ML}(\mathcal{R}_n, r, s) = \begin{cases} 1, & \text{si } o_r \in C_k^n \text{ et } o_s \in C_k^n \\ 0, & \text{autrement,} \end{cases}$$

et

$$v_{CL}(\mathcal{R}_n, r, s) = \begin{cases} 1, & \text{si } o_r \in C_k^n \text{ et } o_s \notin C_k^n \\ 0, & \text{autrement.} \end{cases}$$

Le critère local est aussi modifié pour intégrer le taux de satisfaction des contraintes $\theta$ :

$$\gamma^{(i,j)} = \frac{1}{2}(p_s.(\frac{1}{n_i}\sum_{k=1}^{n_i}\omega_k^{(i,j)} + \frac{1}{n_j}\sum_{k=1}^{n_j}\omega_k^{(j,i)}) + p_q.(\delta^i + \delta^j) + p_c.(\theta^i + \theta^j)), \tag{2.9}$$

où $p_s + p_q + p_c = 1$
Une grande variété de contraintes peut alors être intégrées, à la seule condition que son taux de satisfaction puisse être mesuré.

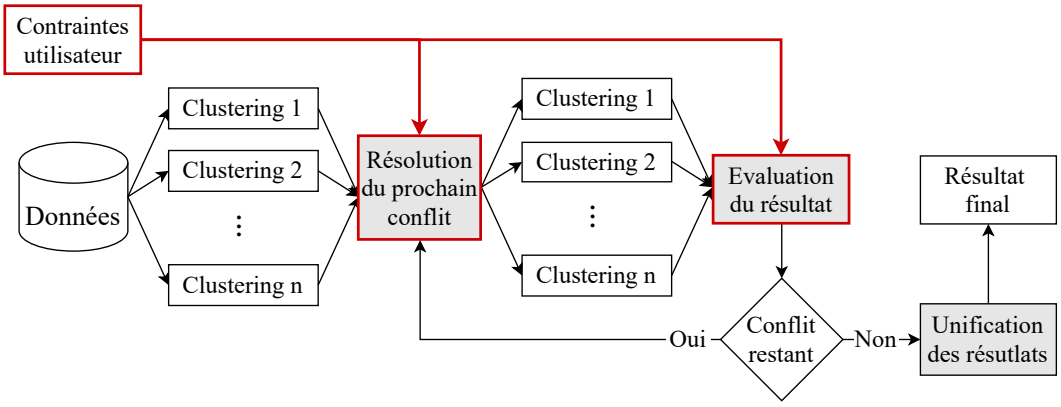



FIGURE 2.3 – La méthode SAMARAH sous contrainte

Cette modification permet un équilibre entre les connaissances injectées par l'expert, la similarité des agents et la qualité interne des clusterings de chaque agents lors de la résolution des conflits. Par exemple, une modification qui entraîne une amélioration importante de la similarité ou de la qualité du clustering d'un agent peut être approuvée même si la fraction de contraintes satisfaites diminue si cela améliore le critère global $\Gamma$. Par conséquent, les contraintes permettent de guider le processus de clustering, et peuvent ne pas être satisfaites. Cela peut être considéré comme un inconvénient si les contraintes doivent être absolument satisfaites, mais cela atténue également le problème de la sur-contrainte.

Que ce soit SAMARAH ou les méthodes vues dans la section précédente, chacune propose une stratégie d'intégration des contraintes très différentes, même au sein d'une même catégorie d'algorithmes. Il est donc important de les évaluer afin de mieux comprendre comment ces méthodes réagissent à l'ajout de contraintes.

## 2.4 Comparaison et limites

Dans LAMPERT et collab. [2018] nous avons proposé une évaluation des méthodes de clustering sous contraintes appliquées aux séries temporelles. Cette étude forme le point de départ et une motivation aux travaux présentés dans cette thèse. Il est donc important de revenir dessus pour mieux comprendre les choix fait par la suite. Nous verrons, donc dans un premier temps, les différents éléments utilisés lors de l'étude (section 2.4.1), dans un second temps les résultats obtenus (section 2.4.2) et nous finirons en mettant en évidence les verrous scientifiques rencontrés (section 2.4.3 et 2.4.4).

### 2.4.1 Paramètres expérimentaux

#### Méthodes sélectionnées

Le nombre de méthodes existantes dans la littérature étant considérable, nous avons limité leur évaluation à un sous-ensemble représentatif de celles-ci. Toutefois, pour des raisons d'équité de comparaison, il a été choisi d'utiliser la même mesure de dissimilarité.

**Choix de la mesure de dissimilarité :** Comme présenté dans la section 1.1.3, une grande variété de mesures existe pour répondre à l'incapacité des fonctions de distance classique (par exemple, la distance euclidienne) à gérer les distorsions temporelles. Parmi celles-ci, Dynamic Time Warping (DTW) [Sakoe et Chiba, 1978], par sa capacité à trouver un alignement optimal entre deux séries temporelles en les déformant de manière non linéaire, surmonte des limites. De plus, certains types d'algorithmes de clustering, par exemple Kmeans, calculent les centres lors de leur optimisation. Or, il n'existe pas à l'heure actuelle de méthode de calcul exact dans le cas des séries temporelles. L'algorithme DTW Barycenter Averaging (DBA) [Petitjean et collab., 2011] propose une solution efficace basée sur le raffinement de manière itérative d'une séquence moyenne afin d'obtenir une estimation de la moyenne réelle de bonne qualité. Ceci permet une utilisation simple des méthodes de clustering sous contraintes existantes nécessitant un tel calcul.

Il convient de noter que DTW n'est pas la seule alternative à la distance euclidienne pour mesurer la dissimilarité entre les séries temporelles. Il est néanmoins souvent constaté que les mesures de dissimilarité alternatives ne sont pas significativement meilleures que DTW sur des données concrètes [Bagnall et collab., 2017; Lines et Bagnall, 2015; Wang et collab., 2013]. Pour simplifier la comparaison des méthodes nous avons restreint cette évaluation des méthodes à l'utilisation de DTW (et DBA le cas échéant).

**Le choix des algorithmes de clustering sous contraintes :** Le choix des méthodes est en partie justifié par l'accès au code des auteurs et la possibilité de l'adapter à l'utilisation de DTW tout en gardant un temps de calcul raisonnable. Plus de détails peuvent être trouvés dans l'article lui même [Lampert et collab., 2018]. Les méthodes qui ont été sélectionnées sont les suivantes :

— Cop-Kmeans : Comme pour l'algorithme Kmeans classique, le clustering est effectué en affectant chaque point au centre le plus proche. Cependant, Cop-Kmeans [Wagstaff et collab., 2001] étend cela en validant toute affectation de point par rapport à l'ensemble de contraintes. Si l'affectation viole une contrainte, le cluster le plus proche suivant est testé jusqu'à ce qu'une affectation valide soit trouvée. S'il n'en existe pas, et à défaut de mécanisme de retour arrière, l'algorithme échoue. En tant que telles, les contraintes représentent une restriction stricte sur l'affectation de points. De cette manière, Cop-Kmeans peut être considéré comme une méthode heuristique qui tente d'imposer le respect des contraintes. DTW est utilisée comme distance pour Kmeans, avec DBA pour le calcul de moyennes.

— Spec : Les méthodes de clustering spectral forment un équilibre entre les informations dérivées de la fonction de distance et l'ensemble de contraintes, et n'imposent donc pas d'exigence stricte que les contraintes soient satisfaites. La méthode la plus basique est celle présentée par Kamvar et collab. [2003]. Le principe du clustering spectral est l'utilisation d'une matrice laplacienne $L$, calculée à partir d'une matrice de similarité. L'espace spectral est alors construit en utilisant les $K$ (nombre de clusters attendus) premiers vecteurs propres orthogonaux de $L$ pour obtenir une matrice $X$, de plus petite dimension, définie par la concaténation des vecteurs propres en colonnes. Le clustering est alors effectué en regroupant les lignes en 'blocs' (les clusters). La matrice laplacienne $L$ étant la représentation d'un graphe, noté $G$, l'objectif du clustering spectral

peut donc être vu comme la recherche d'une partition de $G$ tel que les arêtes entre différents clusters aient des poids très faibles (ce qui signifie que les objets dans différents clusters sont différents) et les arêtes au sein d'un groupe ont des poids élevés (ce qui signifie que les objets dans le même cluster sont similaires). Kamvar et collab. [2003] propose donc d'intégrer les contraintes en modifiant la matrice de similarité qui sert de base au calcul de $L$. De cette manière, les objets sujets à une contrainte must-link (ML) sont rendus fortement similaires en pondérant au maximum leur arête dans le graphe. Inversement, les objets sujet à une contrainte cannot-link (CL) sont rendus plus dissemblables en pondérant au minimum leur arête dans le graphe. Ceci permet alors pour les ML d'augmenter, respectivement diminuer pour les CL, la probabilité que les deux objets se trouvent dans la même sous-graphe.

L'utilisation de la mesure de dissimilarité DTW dans l'algorithme Spec ne pose pas de problème. Il suffit d'utiliser cette mesure pour construire la matrice de similarité $S$ par :

$$S_{ij} = -\text{DTW}(o_i, o_j). \tag{2.10}$$

$S$ est ensuite utilisée pour calculer la matrice d'affinité $A$ des poids du graphe définie par :

$$A_{ij} = \exp\left(-S_{ij}/2\sigma^2\right) \tag{2.11}$$

— CCSR : Cette méthode repose sur les même principes que Spec, excepté que pour l'intégration des contraintes ML et CL. Li et collab. [2009] propose de biaiser l'espace spectral appris pour le rendre plus cohérent avec les contraintes ML et CL proposées. En effet, l'espace spectral se compose des vecteurs propres ayant les plus petites valeurs propres de la matrice laplacienne $L$. La méthode consiste à adapter cette sélection pour que l'espace spectral créé s'accorde avec les contraintes. L'idée sous-jacente est que la déformation induite par cette sélection, et donc indirectement la connaissance injectée à travers les contraintes, s'applique aussi aux autres objets sans contraintes [Li et collab., 2009]. Ceci est fait en construisant l'espace spectral qui minimise la fonction de coût :

$$\mathcal{L}(X) = \sum_{i=1}^{n}\left(\mathbf{y}_i^T\mathbf{y}_i - 1\right)^2 + \sum_{(i,j)\in\text{ML}}\left(\mathbf{y}_i^T\mathbf{y}_j - 1\right)^2 + \sum_{(i,j)\in\text{CL}}\left(\mathbf{y}_i^T\mathbf{y}_j - 0\right)^2 \tag{2.12}$$

, où $X = (\mathbf{y}_1, \ldots, \mathbf{y}_n)^T$ est la représentation des données définie pour Spec.

La minimisation de $\mathcal{L}(F)$ est résolue en utilisant la programmation semi-définie (SDP). Le minimum de $\mathcal{L}$ résulte en une représentation dans laquelle les objets sont à la fois proches de la sphère unitaire (le premier terme), afin d'obtenir un espace spectral simple à clusteriser car normalisé sur les $K$ vecteurs propres : les objets liés par une contrainte ML sont proches les uns des autres (le deuxième terme), alors que et les objets liés par une contrainte CL sont éloignés (le deuxième terme).

Comme avec l'algorithme Spec, le seul changement nécessaire pour l'application de l'algorithme aux séries temporelles est de construire la matrice de similarité $S$ en utilisant la mesure DTW (équation 2.10).

— CPClustering : Dao et collab. [2017] ont développé une approche déclarative où l'objectif est de résoudre un problème d'optimisation de contraintes. Elle consiste à trouver une solution optimale, selon une fonction d'optimisation donnée $F$ (par exemple de densité, ou de compacité minimale), qui satisfait toutes les contraintes. Pour trouver cette solution optimale, la méthode utilise un *solveur* qui explore les différentes configurations de clustering possibles et qui utilise les contraintes pour élaguer l'espace de recherche. Chaque objet de l'ensemble de données est intégré, un à la fois, dans le résultat courant. Les objets faisant l'objet d'une contrainte sont traités en premier. Si une contrainte est violée, le *solveur* revient en arrière jusqu'à ce qu'il trouve une affectation valide différente. Chaque fois qu'une solution est atteinte, sa valeur d'objectif $f$ est calculée et le

| UCR dataset | # classes | # test data points | Length |
|---|---|---|---|
| ECG5000 | 5 | 4500 | 140 |
| ElectricDevices | 7 | 7711 | 96 |
| FacesUCR | 14 | 2050 | 131 |
| InsectWingbeatSound | 11 | 1980 | 256 |
| MALLAT | 8 | 2345 | 1024 |
| StarLightCurves | 3 | 8236 | 1024 |
| TwoPatterns | 4 | 4000 | 128 |
| UWaveGestureLibraryAll | 8 | 3582 | 945 |
| UWaveGestureLibraryX | 8 | 3582 | 315 |

TABLEAU 2.1 – Subset of the UCR repository used for experimentation.

*solveur* revient en arrière en ajoutant une nouvelle contrainte $F < f$, qui impose que la solution suivante soit meilleure que la solution actuelle. Par conséquent, la dernière solution trouvée est la meilleure. Dans l'implémentation utilisée, la propagation se fait à l'aide du calcul du plus proche K-médoïde. La fonction $F$ est définie comme la minimisation du diamètre maximal du cluster via l'introduction d'une variable $D$ dans le modèle. Cette variable représente le diamètre maximal des clusters. Par conséquent, deux objets quelconques $o_i$, $o_j$ dont la mesure de dissimilarité, $d(o_i, o_j)$, est supérieure à $D$, doivent être dans des clusters différents. Ceci s'exprime par la relation :

$$\forall i, j \in \{1, \ldots, N\}, \quad d(o_i, o_j) > D \longrightarrow G_i \neq G_j, \tag{2.13}$$

qui est intégrée via une nouvelle contrainte $diameter(D, G, d)$.

L'intégration de DTW consiste en un calcul préalable d'une matrice de dissimilarité $M$ qui sert alors de mesure de dissimilarité :

$$M_{ij} = \mathrm{DTW}(o_i, o_j). \tag{2.14}$$

— SAMARAH : Pour rappel (section 2.3.1), la méthode proposée par FORESTIER et collab. [2010a] consiste en un affinage itératif du résultat de clustering de plusieurs agents de clustering pour obtenir un accord entre les différents agents. L'algorithme se base sur la détection de conflits (différences entre les résultats de deux agents) et une méthode explorative pour les résoudre. Un critère de qualité est utilisé pour évaluer si les modifications sont pertinentes ou non. Dans sa version non contrainte, ce critère est une combinaison de la similarité entre les clusterings et la qualité de chaque clustering (par ex. l'inertie ou la compacité). Pour la version contrainte, ce critère est étendu pour favoriser les solutions respectant les contraintes lors de la résolution des conflits.

L'intégration de DTW doit se faire au niveau chaque agent. Dans le cas de cette étude les agents seront basés sur KMEANS avec des paramètres initiaux différents, les modifications sont donc identiques à celles décrites pour l'algorithme COP-KMEANS.

### Méthodologie

Plusieurs jeux de données ont été choisis pour évaluer les algorithmes afin qu'ils représentent des problèmes typiques de clustering de séries temporelles. Les jeux de données utilisés sont tirés de l'archive de l'UCR [CHEN et collab., 2015] qui est un ensemble de référence permettant la publication de résultats comparables. Un sous-ensemble de l'archive, contenant un grand nombre d'échantillons et un nombre modéré de classes, a été choisi pour refléter les caractéristiques des problèmes rencontrés en clustering (voir tableau 2.1).

Les contraintes ont été générées en sélectionnant aléatoirement des paires d'objets et en générant une contrainte must-link ou cannot-link selon que ces objets appartiennent ou non

à la même classe. Différentes tailles d'ensembles de contraintes ont été considérées : 5%, 10%, 15%, et 50% du nombre d'objets $N$ dans le jeu de données, ce qui représente une très petite fraction du nombre total $\frac{1}{2}N(N-1)$ de contraintes possibles. Dix ensembles pour chaque taille ont été générés, de sorte que pour chaque taille l'expérience a été répétée dix fois, avec à chaque fois un sous-ensemble aléatoire de contraintes différent. A noter que tous les algorithmes sont évalués en utilisant ces mêmes jeux de contraintes aléatoires.

Les données de l'échantillon ont été normalisées entre 0 et 1. L'indice de Rand ajusté (ARI) [Hubert et Arabie, 1985] et le taux de satisfaction des contraintes (Sat.) ont été utilisés pour évaluer les performances des méthodes. Le taux de satisfaction des contraintes est simplement le rapport du nombre contraintes satisfaites sur le nombre total de contraintes. L'ARI est quant à lui calculé comme suit :

$$ARI = \frac{\sum_{ij}\binom{n_{ij}}{2} - [\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}]/\binom{N}{2}}{\frac{1}{2}[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}] - [\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}]/\binom{N}{2}} \tag{2.15}$$

où $n_{ij}$ est la valeur de table de contingence $C$ de taille $m_c \times m_l$ , $a_i$ la somme des valeurs sur la ligne $i$ et $b_j$ sur la colonne $j$ et $N$ est la cardinalité du jeu de données. Les résultats reportés plus bas représentent la moyenne de dix répétitions de chaque expérience (chacune avec un ensemble aléatoire différent de contraintes) et sont arrondis à trois décimales.

Enfin, pour mesurer le degré d'accord entre la fonction d'objectif non supervisé et les contraintes, le taux satisfaction des contraintes a également été évalué sur les résultats de clustering sans contrainte. Pour cela, le taux de satisfaction a été calculé en utilisant une moyenne sur les ensembles de contraintes de taille 50%. Dans la litérature, cette mesure est appelée la *cohérence* [Wagstaff et collab., 2006]. Elle permet de mesurer le taux de contraintes qu'un algorithme est capable de déterminer correctement en utilisant ses biais par défaut. Le taux de satisfaction des contraintes est calculé pour un ensemble de contraintes must-link, $ML$, et cannot-link, $CL$, sur un résultat de clustering $\mathcal{R}_n = \{C_1^n, \ldots, C_k^n\}$ par :

$$Q(\mathcal{R}_n, ML, CL) = \frac{1}{|ML| + |CL|}\left(\sum_{(i,j)\in \mathrm{ML}} v_{ML}(\mathcal{R}_n, i, j) + \sum_{(i,j)\in \mathrm{CL}} v_{CL}(\mathcal{R}_n, i, j)\right), \tag{2.16}$$

où

$$v_{ML}(\mathcal{R}_n, i, j) = \begin{cases} 1, & \text{si } o_r \in C_k^n \text{ et } o_s \in C_k^n \\ 0, & \text{autrement,} \end{cases}$$

et

$$v_{CL}(\mathcal{R}_n, i, j) = \begin{cases} 1, & \text{si } o_r \in C_k^n \text{ et } o_s \notin C_k^n \\ 0, & \text{autrement.} \end{cases}$$

**Paramétrage des méthodes**

Du fait que ces algorithmes soient évalués dans un contexte de clustering sous contraintes, les données de référence ne peuvent être utilisées que pour créer les contraintes. Nous ne disposons donc pas de moyen d'optimiser les valeurs des différents paramètres des méthodes.

Le paramètre principal ayant tout de même requis une supervision est le nombre de clusters attendu $K$. Celui-ci a été donnée à l'ensemble des méthodes, avec comme valeur le nombre de classes du jeu de données, à l'exception de CPClustering qui fixe le nombre de clusters indépendemment.

Pour Cop-Kmeans, un nombre d'itérations très important est souvent nécessaire. Toutefois nous avons constaté expérimentalement que la valeur de la fonction de coût sous DBA diminue globalement (ou reste la même) à chaque itération, et donc le choix d'un nombre suffisamment grand d'itérations a permis d'obtenir une bonne approximation du résultat final (dans ces expériences, une valeur de 100 a été choisie). Cependant, la même approche pose plus de problèmes pour les approches spectrales (Spec et CCSR), car leurs paramètres ($\sigma$

dans l'équation 2.11 et le nombre de vecteurs propres) doivent être spécifiquement choisis pour chaque problème. Malheureusement, ceux-ci ne peuvent pas être sélectionnés intuitivement et les performances de l'algorithme dépendent fortement de ces valeurs [Ng et collab., 2001]. Pour permettre aux méthodes spectrales d'être incluses dans l'étude, le cas d'utilisation (peu réaliste en pratique) a été choisi est celui où ces paramètres ont été optimisés par *grid search* sur les ensembles d'apprentissage inclus dans les ensembles de données UCR avec une fraction de contraintes de 5%. Nous reviendrons sur ce choix dans la section 3.2.1. Pour Samarah, 3 agents de type Kmeans ont été utilisés avec un nombre de clusters initiaux de $K-2$, $K$ et $K+2$, et le nombre d'itérations a été fixé à 20 afin de permettre un temps de calcul raisonnable. Le choix de Kmeans pour les agents a été motivé par la présence de COP-Kmeans qui est basées sur le même algorithme de clustering. Cela permet de mieux séparer le gain dû au processus collaboratif de la performance des agents de clustering eux-mêmes. En dehors de cela, les valeurs par défaut des méthodes ont été utilisées.

Enfin le code utilisé pour réaliser cette étude est disponible en ligne dans un dépôt commun [1], à l'exception de Samarah qui est disponible sur un dépôt séparé [2].

### 2.4.2 Résultats de l'étude

Les performances des algorithmes sans contraintes sont présentées en premier pour déterminer une base de comparaison. Ces résultats sont présentés dans le tableau 2.2 et le taux de satisfaction non contraint moyen dans le tableau 2.3. Il ressort de ces résultats que Spec surpasse tous les autres algorithmes dans la plupart des ensembles de données, parfois avec un écart important, comme par exemple ECG5000 et UWaveGestureLibraryAll. Le classement général des algorithmes en termes de rang moyen sur l'ensemble des jeux de données par rapport à l'ARI est : Spec (1.6), COP-KMeans (2.6), Samarah (3), CCSR (3.7) et CPClustering (4.1). On constate également que Spec est la méthode qui semble la plus cohérente, ce qui est en partie lié à ses bon résultats d'ARI. Un classement similaire ressort si l'évaluation se fait en termes de nombre de jeux de données avec lesquels ils ont atteint la taux de satisfaction non-contraint le plus élevé avec dans l'ordre : Spec (1.9), COP-KMeans (2.2), CCSR (3.3), Samarah (3.3) et CPClustering (4.2).

| Méthode | ECG5000 | ElectricDevices | FacesUCR | InsectWingbeatSound | MALLAT | StarLightCurves | TwoPatterns | uWaveGestureLibraryX | UWaveGestureLibraryAll |
|---|---|---|---|---|---|---|---|---|---|
| COP-KMeans | 0.433 | **0.337** | 0.485 | 0.057 | 0.750 | 0.541 | 0.933 | **0.439** | 0.447 |
| Spec | **0.816** | 0.231 | **0.556** | **0.151** | **0.931** | **0.678** | **1.000** | 0.270 | **0.495** |
| CCSR | 0.029 | 0.328 | 0.410 | 0.125 | 0.861 | 0.231 | 0.000 | 0.260 | 0.291 |
| CPClustering | 0.455 | 0.180 | 0.188 | 0.098 | 0.642 | 0.616 | 0.257 | 0.223 | 0.178 |
| Samarah | 0.531 | 0.312 | 0.424 | 0.043 | 0.792 | 0.506 | 0.868 | 0.382 | 0.356 |

Tableau 2.2 – ARI moyenne - non contraint. Le meilleur score par jeu de données est mis en gras.

Les résultats moyens des ARI et de la satisfaction des contraintes pour les expériences de clustering sous contraintes sont présentés en annexe (tableaux A.1 à A.9) et sont résumés dans les tableaux 2.4 et 2.5. En raison des temps de calcul supplémentaires dus à l'intégration de DTW et DBA dans l'algorithme de COP-KMeans, celui-ci n'a pas toujours pu terminer

1. https://icube-forge.unistra.fr/lampert/TSCC
2. https://icube-forge.unistra.fr/lafabregue/Mustic/-/tree/master/src/test

| Méthode | ECG5000 | ElectricDevices | FacesUCR | InsectWingbeatSound | MALLAT | StarLightCurves | TwoPatterns | uWaveGestureLibraryX | UWaveGestureLibraryAll |
|---|---|---|---|---|---|---|---|---|---|
| COP-KMeans | 0.732 | **0.812** | 0.915 | 0.830 | 0.936 | 0.779 | 0.977 | **0.870** | 0.879 |
| Spec | **0.829** | 0.763 | **0.926** | 0.847 | **0.983** | **0.833** | **1.000** | 0.776 | **0.881** |
| CCSR | 0.525 | 0.796 | 0.906 | **0.850** | 0.970 | 0.615 | 0.617 | 0.828 | 0.834 |
| CPClustering* | 0.730 | 0.703 | 0.827 | 0.773 | 0.916 | 0.806 | 0.717 | 0.821 | 0.773 |
| SAMARAH | 0.774 | 0.784 | 0.894 | 0.808 | 0.887 | 0.761 | 0.950 | 0.847 | 0.843 |

TABLEAU 2.3 – Taux de satisfaction des contraintes moyen du clustering non contraint (mesuré avec les jeu de contraintes de 50%). Le meilleur score par jeu de données est mis en gras.

| Method | 5% | 10% | 15% | 50% |
|---|---|---|---|---|
| COP-KMeans | 0.001 (0.048) | 0.008 (0.055) | −0.003 (0.047) | 0.006 (0.064) |
| Spec | −0.034 (0.068) | −0.030 (0.062) | −0.043 (0.078) | −0.049 (0.101) |
| CCSR | 0.262 (0.298) | 0.263 (0.297) | 0.263 (0.296) | 0.261 (0.296) |
| CPClustering | −0.029 (0.089) | −0.023 (0.067) | −0.021 (0.076) | −0.036 (0.085) |
| SAMARAH | 0.067 (0.087) | 0.076 (0.080) | 0.063 (0.081) | 0.079 (0.089) |

TABLEAU 2.4 – Différence d'ARI entre le clustering contraint et non-contraint (ARI contraint − ARI non-contraint) pour chaque fraction de contraintes moyennée sur l'ensemble des jeux de données, l'écart type est indiqué entre parenthèse.

dans un laps de temps raisonnable. L'absence de ces résultats est marquée par un tiret dans le tableau correspondant. A noter, que les jeux de données concernés représentent ceux pour lesquels les longueurs des séries temporelles sont les plus grandes. De plus, plusieurs itérations d'ensembles de contraintes ont entraîné des violations de contraintes pendant le processus de clustering COP-KMeans et ces résultats sont marqués dans les tableaux (avec le nombre de contraintes traitées indiqué en légende).

L'algorithme Spec surpasse tous les autres algorithmes sur 5 jeux de données pour chaque fraction de contraintes. Cela n'est pas surprenant car il obtient déjà un très haut score sans contrainte sur ces 5 jeux de données. L'analyse des variations d'ARI moyen (ARI contraint − ARI non-contraint) pour chaque algorithme pour chaque fraction de contrainte (Tableau 2.4) révèle que tous les algorithmes ne bénéficient pas de l'introduction de contraintes. Seuls deux méthodes profitent vraiment de l'ajout de contraintes : CCSR qui en bénéficie le plus avec une augmentation de ∼0.262 d'ARI, suivi de SAMARAH avec ∼0.071 d'ARI. Il est intéressant de noter que l'ajout de contraintes supplémentaires n'entraîne pas directement une augmentation de l'ARI moyen. Dans certains cas (Spec et CPClustering), l'ajout de contraintes conduit

| Method | 5% | 10% | 15% | 50% |
|---|---|---|---|---|
| COP-KMeans | 0.141 (0.074) | 0.141 (0.074) | 0.140 (0.073) | 0.134 (0.070) |
| Spec | −0.006 (0.063) | 0.001 (0.037) | −0.011 (0.052) | −0.012 (0.050) |
| CCSR | 0.099 (0.129) | 0.100 (0.125) | 0.104 (0.123) | 0.099 (0.124) |
| CPClustering* | 0.215 (0.063) | 0.215 (0.063) | 0.215 (0.063) | 0.215 (0.063) |
| SAMARAH | 0.053 (0.041) | 0.048 (0.035) | 0.041 (0.033) | 0.039 (0.032) |

TABLEAU 2.5 – Difference de taux de satisfaction des contraintes entre le clustering contraint et non-contraint (Sat. − moyenne Sat. non-contraint) pour chaque fraction de contrainte moyennée sur l'ensemble des jeux de données, l'écart type est indiqué entre parenthèse.

même à une légère diminution de l'ARI. Cependant, tous les écarts types sont supérieurs à la variation de l'ARI. En prenant CCSR comme exemple, dans six des ensembles de données (ECG5000, FacesUCR, MALLAT, TwoPatterns, UWaveGestureLibraryX et UWaveGestureLibraryAll), une forte augmentation de l'ARI associée à une augmentation de la satisfaction des contraintes est observée, indiquant que l'algorithme a bénéficié des informations supplémentaires. Dans les autres ensembles de données, cependant, aucune augmentation ou diminution n'est observée. Sur le taux de satisfaction des contraintes, seuls CPClustering et Cop-Kmeans garantissent la pleine satisfaction, ce qui entraîne nécessairement une forte augmentation du taux de satisfaction. Cependant, cela n'est pas associé à une augmentation significative de l'ARI et, dans certains cas, plutôt à une diminution. Cela indique que ces algorithmes se concentrent de sur le fait de trouver un clustering correct vis à vis des contraintes must-link et cannot-link au détriment du clustering des données restantes. Il est donc clair que le simple ajout de contraintes ne conduit pas toujours à une augmentation des performances pour tous les algorithmes et tous les jeux de données, mais qu'au contraire plusieurs facteurs d'influence semblent être en jeu.

### 2.4.3 Algorithmes, données et impact des contraintes

Comme nous l'avons vu plus haut, l'impact des contraintes sur les performance d'ARI n'est pas forcément positif et peut fortement varier. En conséquent, nous avons voulu savoir s'il est possible d'anticiper le sens et l'ampleur de l'impact des contraintes. Pour cela, nous devons au préalable définir un ensemble de notions.

**Définition 2.4.1 (Informativité des contraintes)** *Pour un algorithme de clustering sous contraintes $\mathcal{A}(\theta, ., .)$, avec $\theta$ les paramètres de l'algorithme, un jeu de donnée $\mathcal{D}$ et un ensemble d'une ou plusieurs contraintes $\mathcal{C}$ sur $\mathcal{D}$ donnés, l'informativité des contraintes de $\mathcal{C}$ correspond à la différence de qualité $\Delta q$ entre le résultat de clustering contraint $\mathcal{R}_c = \mathcal{A}(\theta, \mathcal{D}, \mathcal{C})$ et non-contraint $\mathcal{R}_{nc} = \mathcal{A}(\theta, \mathcal{D}, \emptyset)$. On distingue donc trois cas :*

- $\Delta q = 0$*, le ou les contraintes sont dites non-informatives,*
- $\Delta > 0$*, le ou les contraintes sont dites informatives,*
- $\Delta < 0$*, le ou les contraintes sont dites négativement informatives,*

De par sa définition, l'informativité ne peut être mesurée qu'a posteriori, une fois le résultat de clustering calculé. Le tableau 2.6 illustre également que l'informativité des contraintes est fortement dépendante des données et de l'algorithme utilisé.

**Définition 2.4.2 (Consistance d'un algorithme)** *Pour un algorithme de clustering sous contraintes $\mathcal{A}(\theta, ., .)$, avec $\theta$ les paramètres de l'algorithme, un jeu de donnée $\mathcal{D}$ et un ensemble d'une ou plusieurs contraintes $\mathcal{C}$ sur $\mathcal{D}$ donnés, la consistance se définit comme la capacité de $\mathcal{A}$ à satisfaire l'ensemble des contraintes de $\mathcal{C}$ en utilisant ses propres biais (c'est-à-dire sans utiliser explicitement celles-ci).*

La consistance peut être estimée à partir du résultat de clustering non contraint : $\mathcal{R} = \mathcal{A}(\theta, \mathcal{D}, \emptyset)$, en mesurant le taux de satisfaction des contraintes dans $\mathcal{R}$ tel que défini par l'équation 2.16. Plus le taux de satisfaction des contraintes est élevé, plus la consistance de l'algorithme sur ces données et contraintes sera élevée. On peut d'ailleurs évaluer la consistance moyenne d'un algorithme vis à vis d'un jeu de donnée en mesurant le taux de satisfaction de contraintes moyen sur un ensemble de jeux de contraintes généré aléatoirement, tel que reporté dans le tableau 2.3.

**Définition 2.4.3 (Score de silhouette d'un objet)** *Le score de silhouette d'un objet $o_i$ indique son degré d'appartenance à son cluster et est calculé par :*

$$s(o_i) = \frac{b(o_i) - a(o_i)}{\max\{a(o_i), b(o_i)\}}, \tag{2.17}$$

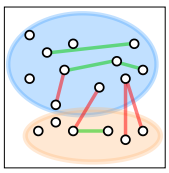
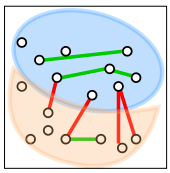
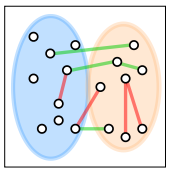
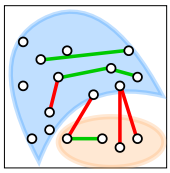
| | Clustering sans contraintes | | | Clustering avec contraintes | | | |
|---|---|---|---|---|---|---|---|
| Algorithme | Résultat | Sat. | Qualité (RI) | Résultat | Sat. | Qualité (RI) | $\Delta q$ |
| Méthode 1 |  | 87.5% forte consistance | 0.94 |  | 100% | 0.94 | 0.00 faible informativité |
| Méthode 2 |  | 37.5% faible consistance | 0.47 |  | 87.5% | 0.82 | 0.35 forte informativité |

TABLEAU 2.6 – Illustration de l'estimation de la consistance des algorithmes et de l'informativité des contraintes sur des données identiques mais des méthodes différentes.

*où, si $o_i$ appartient au cluster $C_k$, $a(o_i)$ est la dissimilarité moyenne de l'objet $o$ avec tous les autres objets dans le même cluster :*

$$a(o_i) = \frac{1}{|C_k| - 1} \sum_{o_j \in C_k, j \neq i} d(o_i, o_j) \tag{2.18}$$

*et $b(o_i)$ est la dissimilarité moyenne de l'objet $o$ avec les objets du cluster le plus proche :*

$$b(o_i) = \min_{k' \neq k} \frac{1}{|C_{k'}|} \sum_{o_{j'} \in C'_k} d(o_i, o_{j'}) \tag{2.19}$$

Ce score varie entre -1 et 1. Un score élevé indique que l'objet $o$ est plus proches de l'ensemble des autres objets du même cluster que de ceux du cluster le plus proche et inversement si le score est négatif.

**Définition 2.4.4 (Score de silhouette d'un jeu de donnée)** *Le score de silhouette d'un jeu de donnée $\mathcal{D}$ est définie comme :*

$$S(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{o_i \in \mathcal{D}} s(o_i) \tag{2.20}$$

*où $a(o_i)$ et $b(o_i)$ sont calculés par rapport aux classes des données de référence comme cluster [ROUSSEEUW, 1987].*

Ce score varie entre -1 et 1, un score élevé indiquant que les classes sont éloignées les une des autres dans l'espace des données ou qu'elles se chevauchent si le score est négatif.

**Définition 2.4.5 (La cohérence des contraintes)** *Pour un algorithme de clustering sous contraintes $\mathcal{A}(\theta, ., .)$, avec $\theta$ les paramètres de l'algorithme, un jeu de donnée $\mathcal{D}$ et un ensemble de contraintes $\mathcal{C} = \{c_1, .., c_n\}$, avec $n > 1$, sur $\mathcal{D}$ donnés, l'ensemble des contraintes $\mathcal{C}$ est cohérent s'il existe des paramètres $\theta'$ tels que le résultat de clustering $\mathcal{R}_c = \mathcal{A}(\theta', \mathcal{D}, \mathcal{C})$ satisfait l'ensemble des contraintes dans $\mathcal{C}$.*

Intuitivement, deux contraintes sont cohérentes si elles ne se contredisent pas, indépendemment de l'informativité de celles-ci. Toutefois, l'incohérence des contraintes ne peut bien souvent être montrée qu'en exécutant le clustering des données avec l'ensemble des paramètres $\theta$. DAVIDSON et collab. [2006] ont alors proposé une méthode pour estimer la cohérence

| Jeu de données | Score de silhouette |
|---|---|
| ECG5000 | 0.329 |
| ElectricDevices | -0.008 |
| FacesUCR | 0.124 |
| InsectWingbeatSound | -0.011 |
| MALLAT | 0.209 |
| StarLightCurves | 0.256 |
| TwoPatterns | 0.378 |
| UWaveGestureLibraryAll | 0.103 |
| UWaveGestureLibraryX | 0.122 |

TABLEAU 2.7 – Le score de silhouette calculé par jeu de donnée en utilisant les classes comme cluster

des contraintes a priori. Des vecteurs sont construits entre l'ensemble des objets qui sont joints soit par des contraintes must-link (ML), soit par des contraintes cannot-link (CL). Les contraintes sont alors dites cohérentes si les vecteurs des contraintes ML sont orthogonaux aux vecteurs des contraintes CL, et incohérentes s'ils sont parallèles (et se chevauchent). C'est une mesure utile lorsque l'on considère la métrique de distance euclidienne, cependant, il n'est pas évident de savoir si ce concept peut être étendu à DTW. En effet, DTW ne définit pas de projection vectorielle et par conséquent de nouvelles mesures pour quantifier l'utilité des contraintes doivent être développées pour cela, ce point reste une question de recherche ouverte. Par conséquent, nous devons mesurer indirectement certaines des propriétés de l'ensemble de contraintes afin de mieux comprendre l'effet de l'utilisation de DTW. Nous avons choisi d'utiliser la mesure du "ratio ML/CL Dist.", qui est le rapport de la distance moyenne entre les paires d'objets soumis à une contrainte ML et les paires d'objets soumis à une contrainte CL. Une valeur inférieure à 1 signifie que les paires ML sont plus proches les unes des autres que les paires CL et donc a priori plus cohérentes vis à vis de la mesure de distance.

### Contexte d'intégration

Une analyse par régression linéaire multiple a été réalisée pour découvrir les facteurs externes aux contraintes qui influencent le changement des performances de clustering lorsque des contraintes sont introduites. Trois paramètres principaux susceptibles d'influencer le changement des performances de clustering nous ont semblé importants, ce qui nous a mené aux hypothèses suivantes :

**Consistance de l'algorithme :** si la consistance des contraintes est élevée, l'ajout de contraintes ne devrait pas affecter considérablement les performances du clustering. Ce critère correspond au taux de satisfaction mesuré précédemment et reporté dans le tableau 2.5.

**Score de silhouette du jeu de données :** Lorsque les clusters sur un jeu de données se chevauchent, l'ajout de contraintes devrait conduire à une augmentation des performances de clustering, les contraintes permettant de plus finement dissocier les classes. Le score de silhouette par jeu de données est donné dans le tableau 2.7.

**Type d'algorithme :** Certains algorithmes peuvent bénéficier de l'introduction de contraintes plus que d'autres. Ce critère correspond à la différence d'ARI entre le clustering contraint et non-contraint mesurée précédemment et reportée dans le tableau 2.5.

Le résultat de l'analyse de régression est présenté dans le tableau 2.8. Comme on pouvait le déduire de manière intuitive, la consistance a une grande influence négative sur la différence ARI, indiquant que, lorsque tous les autres facteurs restent constants, plus la consistance d'un algorithme est élevée dans un cadre non supervisé, moins il en résultera une augmentation de l'ARI lors de l'ajout de contraintes. Ceci corrobore ce qui a été trouvé par WAGSTAFF

| Paramètre | Estimation de corrélation | $p$-value |
|---|---|---|
| Consistance | −0.694 | 1.537e−87 |
| Score de silhouette | 0.408 | 9.346e−71 |
| Cop-Kmeans | 0.031 | 0.100e−2 |
| Spec | −0.056 | 1.876e−16 |
| CCSR | 0.226 | 4.284e−100 |
| CPClustering | −0.054 | 4.274e−8 |
| Samarah | 0.078 | 1.260e−16 |

Tableau 2.8 – Coefficients de régression linéaire multiple (le seuil significatif a été fixé à 0.01).

et collab. [2006]. Comme il a été constaté précédemment, certains algorithmes réagissent plus favorablement à l'introduction de contraintes, Spec et CPClustering négativement et CCSR et Samarah positivement. En dehors de Spec qui est fortement sujet à la problématique de cohérence, il peut être remarqué que les méthodes garantissant par construction le respect des contraintes (CPClustering et Cop-Kmeans) sont bien plus sujettes à un impact négatif des contraintes. Forcer le respect des contraintes semble perturber le clustering sans toutefois nécessairement le guider vers la solution souhaitée. Un relâchement de la satisfaction des contraintes permettrait donc de s'en prémunir. Pour ce qui est du score de silhouette sur les jeux de données, une corrélation positive avec la différence ARI peut être observée. Ce constat infirme la proposition de départ qu'un haut score de silhouette devrait augmenter l'effet positif des contraintes. Cela peut s'expliquer en considérant ce qui se passe lorsqu'un objet soumis à une contrainte ML est entouré de points appartenant à un autre cluster (c'est-à-dire avec un score de silhouette faible ou négatif). L'algorithme est alors biaisé pour regrouper les deux objets. Cependant, cela peut également avoir pour effet de biaiser tout objet similaire (donc proche) d'un objet lié par une contrainte ML à être également assigné au même cluster. Or, dans un ensemble de données avec un score de silhouette faible cet objet est plus susceptible d'appartenir à un autre cluster et donc d'être assigné incorrectement. Les contraintes semblent donc plus adéquates quand elles permettent de confirmer ou orienter l'algorithme dans un partitionnement qui ne va pas en contradiction vis à vis de la mesure de similarité. Les contraintes ne permettent donc pas de créer des relations de proximité (ou d'éloignement), ou du moins les généraliser au niveau de la forme des clusters, si cela ne se traduit pas concrètement au niveau de la similarité des objets. Cette interprétation fait d'ailleurs écho au problème de relâchement de la satisfaction des contraintes. À trop vouloir forcer le respect des contraintes, les méthodes concernées semblent créer des relations artificielles entre les objets contraints sans que cela se répercute au niveau du cluster créant alors une perturbation qui n'est pas nécessairement de fondement au niveau des données.

### Caractéristiques des contraintes

Il a déjà été montré que l'ajout d'un nombre plus important de contraintes n'entraînait pas nécessairement une augmentation de la performance du clustering. Les résultats obtenus dans cette étude confirment cette hypothèse. Cela implique qu'il est nécessaire d'envisager des méthodes pour mesurer l'utilité des contraintes afin de déterminer si elles doivent être prises en compte ou non. Davidson et collab. [2006] montrent que les contraintes améliorent les performances lorsqu'elles sont à la fois informatives et cohérentes. Toutefois, de par la définition que nous avons donnée de l'informativité, celle-ci est par construction corrélée avec la performance des contraintes. Pour la cohérence, le tableau 2.9 présente la comparaison entre le ratio ML/CL Dist. et la gain moyen de l'ajout de contraintes sur l'ARI. Même si le ratio ML/CL Dist. ne reste qu'un indicateur partiel de la cohérence, une corrélation entre les deux mesures peut être observée, ce qui implique donc que plus les contraintes sont cohérentes, plus leur effet est bénéfique pour la méthode de clustering sous contrainte.

| Jeu de données | gain d'ARI | Ratio ML/CL Dist. |
|---|---|---|
| ECG5000 | 0.097 | 0.551 |
| ElectricDevices | 0.002 | 0.815 |
| FacesUCR | 0.013 | 0.694 |
| InsectWingbeatSound | 0.004 | 0.726 |
| MALLAT | 0.004 | 0.461 |
| StarLightCurves | 0.078 | 0.269 |
| TwoPatterns | 0.195 | 0.512 |
| UWaveGestureLibraryAll | 0.004 | 0.750 |
| UWaveGestureLibraryX | 0.031 | 0.649 |

TABLEAU 2.9 – Ratio entre la distance moyenne des ML et distance moyenne des CL comparé au gain d'ARI entre clustering non-containt et contraint.

### 2.4.4 Conclusion

L'objectif principal de cette étude a été de montrer que les contraintes n'ont pas nécessairement un impact positif sur le résultat du clustering et de déterminer leur impact. Ainsi, nous avons également montré que nous pouvons identifier certains facteurs déterminants de l'impact des contraintes.

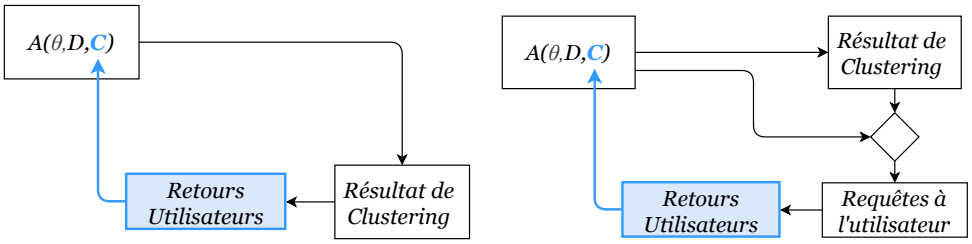
— La consistance : Cela correspond à la capacité de la méthode à satisfaire les contraintes proposées sur un jeu de données en utilisant ses propres biais.
Cette notion peut être abordées selon deux points de vue. Tout d'abord du point de vue de l'algorithme : si celui-ci permet d'obtenir déjà de bons résultats sans contraintes, alors les contraintes auront un effet faible. Mais également du point de vue des contraintes : pour que les contraintes aient un impact positif sur les performance du clustering alors elles doivent apporter des informations supplémentaires que celles déjà apprises par l'algorithme de clustering par ses propres biais. Ce qui correspondrait à utiliser des contraintes informatives.

— Score de silhouette et cohérence : ces deux notions reflètent une adéquation entre la données et les classes (et donc également les contraintes) attendues.
A nouveau, deux points de vue permettent d'éclairer ce problème. Du point de vue de la donnée, avec le score de silhouette : les contraintes auront un effet limité, voire négatif, si les classes se chevauchent fortement dans l'espace des données. Du point de vue des contraintes, avec la cohérence : si les contraintes sont en contradiction avec la structure interne des données, alors elles risquent de perturber trop fortement le clustering entraînant une dégradation des performances du résultat.

— Type d'algorithme : cela correspond à la manière dont la méthode intègre les contraintes. Selon la manière dont les contraintes sont utilisées dans le processus de clustering sous contraintes l'impact peut fortement varier. Les méthodes permettant un relâchement de la satisfaction des contraintes permettent d'obtenir un impact plus positif de l'ajout des contraintes sur le performance du clustering.

Il ressort de ces trois facteurs, que pour améliorer les performances de qualité des méthodes de clustering sous contraintes il faut s'orienter vers des méthodes qui permettent à la fois un relâchement du respect des contraintes, la sélection de contraintes qui ne seraient pas respectées a priori par la méthode (consistance) et qui ne viennent pas en contradiction avec la structure de la donnée (cohérence). La cohérence est fortement liée à la structure propre des données et de la métrique de distance utilisée et donc que très peu à la méthode utilisée. Nous nous sommes donc concentrés sur les deux autres points en proposant une méthode basée sur l'incrémentalité et n'imposant pas le respect des contraintes.

Dans la section suivante, nous présentons notre proposition qui portera sur la consistance liée aux contraintes.




(a) Approches incrémentales (b) Approches actives

FIGURE 2.4 – Comparaison des méthodes de clustering sous contraintes uniquement incrémentales et des actives.

## 2.5 Clustering sous contraintes incrémental

Dans cette section nous présentons notre approches incrémentale. Après avoir présenté les motivations de ce choix (section 2.5.1), nous présenterons ensuite le principe de notre méthode (section 2.5.2) et son implémentation (section 2.5.3 et 2.5.4).

### 2.5.1 Comment assister l'utilisateur dans le choix de contraintes ?

Plusieurs méthodes ont été proposées dans la littérature pour aider l'expert dans la sélection de contraintes. La majorité de ces méthodes reposent sur l'analyse d'un clustering existant qui sert à évaluer les contraintes les plus pertinentes. Ces contraintes sont ensuite fournies à la méthode de clustering qui redémarre en prenant en compte les nouvelles contraintes (voir figure 2.4a).

COHN et collab. [2003] ont proposé une méthode reposant sur ce concept. Cette méthode s'appuie sur les commentaires des utilisateurs pour générer des contraintes. Dans cette approche, appelée *semi-supervised clustering with user feedback*, le processus démarre à partir d'un clustering initial. L'utilisateur donne alors, itérativement, des indications sur son accord ou son désaccord avec le résultat proposé sous la forme de contraintes par paire entre groupes d'objets. Ils suggèrent aussi la possibilité d'incorporer des commentaires sur les clusters eux-mêmes, par exemple l'utilisateur indiquant s'ils sont "bons" ou "mauvais", impliquant que le prochain clustering devra le maintenir. Mais, ils proposent une implémentation uniquement sur les contraintes par paires et ne donnent pas d'indication sur la manière d'intégrer de nouvelles formes de contraintes. De plus, cette méthode a été construite autour d'une mesure de similarité particulière, la Kullback-Leibler divergence à la moyenne [PEREIRA et collab., 1993]. Cette mesure a été proposée pour la classification des documents. Cependant, à notre connaissance, aucune adaptation aux données temporelles n'a été faite. DAVIDSON et collab. [2007] ont étudié une autre approche pour contourner le problème, basée sur l'idée qu'il est souvent plus efficace de mettre à jour un clustering existant pour satisfaire de nouvelles (et anciennes) contraintes plutôt que de reclasser l'ensemble de données à partir de zéro. Les résultats de ces articles confirment que les retours utilisateurs sont plus efficaces que lorsque les contraintes sont choisies aléatoirement. Ils soutiennent également que le nombre de contraintes n'a pas besoin d'être élevé, et même plus, que si ce nombre augmente trop, les résultats peuvent se détériorer.

Certains travaux, vont plus loin, en proposant directement des contraintes potentielles à l'utilisateur. Ces approches sont alors qualifiées de méthodes de clustering sous contraintes actif [SETTLES, 2009] (voir figure 2.4b). Ces méthodes visent à simplifier le travail de l'expert, ce dernier n'ayant qu'à réponde aux requêtes envoyées par l'algorithme (par exemple, demander si deux objets doivent être groupées au sein du même cluster ou non). De plus elles visent également à obtenir des contraintes de meilleurs qualité. Pour cela, certaines se focalisent sur l'aspect informatif, en se basant par exemple sur l'incertitude [LEWIS et GALE, 1994; SETTLES et CRAVEN, 2008], où l'utilisateur est interrogé sur les instances pour lesquelles l'al-

gorithme a le degré de confiance dans leur affectation est le plus faible. D'autres portent sur la cohérence, avec des méthodes qui tentent de trouver l'instance qui aura le plus d'effet sur le clustering [Basu et collab., 2004; Settles et collab., 2008; Van Craenendonck et collab., 2018a] ou celles qui réduisent les erreurs potentielles du clustering [Guo et Greiner, 2007; Roy et McCallum, 2001].

Que ce soit les approches purement incrémentales, ou les approches actives, elles visent à réduire le fossé entre les résultats produits par les algorithmes et les intuitions thématiques de l'expert mais également de rendre les résultats plus compréhensibles pour celui-ci, donc plus aisés à associer à une sémantique du domaine d'application. Les approches actives peuvent alors sembler plus pertinentes car elles réduisent encore plus la charge de travail de l'expert. Cependant, les objets sujets aux contraintes proposées peuvent ne pas être dans la zone d'intérêt de l'utilisateur, voire ne pas être dans son domaine d'expertise (par exemple, dans le cas où seul un sous échantillon intéresse l'expert, ou dans le cas d'objets extrêmes ou fortement ambiguës).

Par conséquent, nous avons décidé de centrer notre approche sur le principe d'incrémentalité. De plus, nous faisons également l'hypothèse que si l'utilisateur injecte à la volée des informations en fonction de l'avancement de l'analyse, cela limitera néanmoins fortement son implication directe et permettra une sémantisation plus aisée. Enfin, nous estimons que placer l'utilisateur dans un contexte de critique/commentaire facilite la tache d'annotation en lui offrant un guidage indirect.

### 2.5.2 Un processus incrémental

Notre méthode repose sur le déroulé suivant :

1. Initialisation : $t = 0$ ; calcul du premier résultat de clustering sans contraintes $R_0$ ;
2. Incémentation :
   (a) Présentation du résultat courant $R_t$ ;
   (b) Validation des résultats par l'expert : Si valide allez à l'étape 3, sinon continuer ;
   (c) Donner de nouvelles contraintes $\mathcal{C}$ par l'expert ;
   (d) Cycle de clustering sous contraintes avec $R_t$ et $\mathcal{C}$ : obtention du nouveau résultat $R_{t+1}$, $t = t + 1$ ;
   (e) Retour à l'étape 2 ;
3. Fin du processus avec le résultat $R_t$

Ce déroulé est illustré dans la figure 2.5. Il consiste essentiellement en une boucle d'échange entre l'expert et l'algorithme de clustering (l'étape 2). Nous définition le terme *incrémentation* comme une itération de cette boucle.

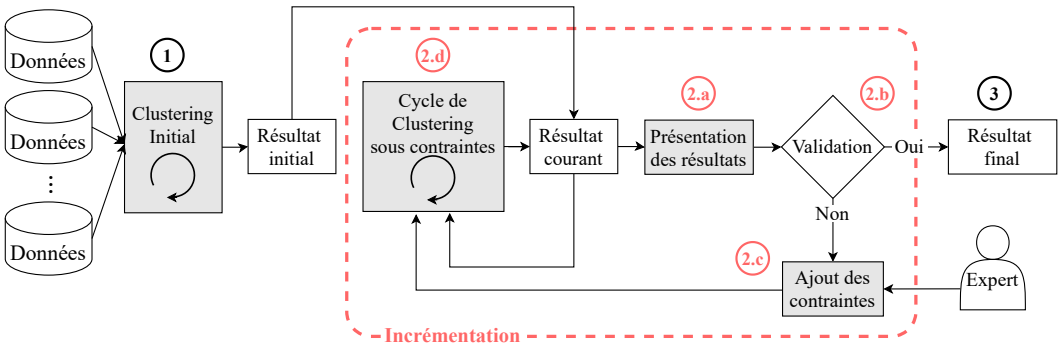



Figure 2.5 – Le principe de clustering sous contraintes incrémental

Néanmoins, pour que le processus incrémental soit pertinent pour l'utilisateur il faut qu'il réponde à certains critères :

— Une influence positive des contraintes sur la performance : nous avons vu que certains algorithmes peuvent être sujet à un impact négatif des contraintes, il faut donc favoriser ceux qui en sont le plus exempt.

— Une prise en compte d'un résultat précédent : le nouveau clustering doit prendre en compte les résultats précédents, car les contraintes lui sont relatives mais également pour ne pas perturber l'utilisateur avec un nouveau résultat trop dissimilaire, notamment sur les zones non-annotées.

— Une généricité des méthodes : la variété des formes de séries temporelles, ou l'application à un autre type de données, nécessite l'utilisation de mesures de similarité variées. De ce fait, nous souhaitons proposer un modèle ne reposant pas sur une métrique spécifique, propre à certains types de données.

Notre choix c'est porté sur la méthode Samarah :

En effet, premièrement, dans la section 2.4 nous avons vu que sa stratégie d'intégration des contraintes permet une relaxation du problème de satisfaction des contraintes, ce qui signifie que la méthode Samarah ne nécessite pas que toutes les contraintes soient satisfaites. Ceci est possible car la méthode utilise les contraintes uniquement dans le cadre d'un critère général composé d'autres critères (voir section 2.3.1). Cela rend également Samarah moins sujette aux contraintes incohérentes et lui permet d'être moins sensible à un effet négatif des contraintes.

Deuxièmement, le processus collaboratif consiste déjà en un processus itératif permettant le raffinement d'un état initial vers une solution consensuelle (voir section 2.3.1).

Troisièmement, Samarah peut intégrer une grande variété de méthodes de clustering, et de mesures de similarité. De plus les méthodes des agents peuvent ne pas gérer les contraintes, car celles-ci sont utilisées au niveau de la collaboration. Néanmoins, pour les méthodes aptes à prendre en compte les contraintes, celles-ci peuvent également être utilisées au niveau de l'agent (voir section 2.3.1). De fait, les contraintes vont être réparties entre les agents en fonction de leur capacité à les gérer.

Quatrièmement, Samarah n'est pas limitée aux contraintes par paire car elle repose sur un simple critère de guidage du processus collaboratif de Samarah. En effet, pour intégrer un nouveau type de contraintes, la méthode nécessite uniquement de pouvoir mesurer le taux de satisfaction des contraintes. Ce taux de satisfaction est ensuite intégré dans le calcul des critères de collaboration (voir section 2.3.1).

### 2.5.3 I-Samarah : une méthode de clustering sous contraintes incrémental

La méthode I-Samarah reprend le déroulé de la figure 2.4a, avec l'implémentation suivante :

— Le clustering initial est obtenu par l'exécution de Samarah sans contraintes ;

— Le résultat courant correspond donc au résultat après la phase d'unification, mais également aux résultats de chaque agent tel qu'obtenus à la fin de la phase de collaboration ;

— Le cycle de clustering sous contraintes correspond à reprise de l'exécution de Samarah, mais cette fois-ci avec les nouvelles contraintes de l'expert.

Cependant, l'utilisation de Samarah pour le cycle de clustering soulève certains problèmes que nous allons aborder dans la suite de cette section.

#### La mise en place de l'incrémentalité

La manière de reprendre l'exécution de Samarah, tout en prenant en compte le résultat courant, peut être fait de différentes manières. Une première option est d'interrompre le processus de collaboration de Samarah après un certain nombre de résolutions de conflit et de lancer l'unification des clustering des agents pour obtenir un résultat courant. La seconde

option est de laisser la méthode SAMARAH s'exécuter entièrement (jusqu'à ce que tous les conflits soient traités), puis d'exécuter à nouveau SAMARAH mais en supprimant la phase d'initialisation. Les agents gardent alors les valeurs obtenues à la fin de la phase de collaboration de la dernière exécution de SAMARAH (par exemple les valeurs des centres pour KMEANS).

Pour la première option, le choix de savoir quand interrompre le processus de collaboration n'est pas trivial (une discussion plus poussée du problème est présentée dans la section 2.5.4). Nous avons donc choisi d'exécuter l'ensemble du processus collaboratif de SAMARAH à chaque incrémentation.

Toutefois l'optimisation du critère global $\Gamma$ lors du processus collaboratif tend à faire croître la similarité entre les résultats de clustering des agents. Or, la dissimilarité entre les résultats initiaux est le moteur principal du processus de collaboration car elle permet la création des conflits, et donc une exploration de l'espace des solutions de clustering possibles. Reprendre l'exécution de SAMARAH à partir d'un premier résultat peut donc mener à une stagnation du résultat de clustering et donc un faible effet des contraintes. Ce problème peut se résumer à un potentiel manque de diversité entre les clusters qui contiennent des objets sujet à une contrainte non satisfaite.

### Un besoin de diversité

Pour résoudre ce problème, nous avons décidé de "perturber" de manière différente chaque agent en répartissant aléatoirement les contraintes entre les agents. Les contraintes sont alors utilisées pour ajouter des clusters dans le clustering actuel de chaque agent. Dans cette implémentation, par exemple pour des agents KMEANS, des centres sont ajoutés pour chaque contrainte, comme décrit dans l'algorithme 3. Pour une contrainte must-link, un nouveau centre est ajouté. Il est défini par la moyenne des deux objets contraints. Pour une contrainte cannot-link, les deux objets impliquées sont ajoutés directement comme nouveaux centres, voir figure 2.6. Pour les autres méthodes, une étude est en cours. Notez que l'ajout de nouveaux clusters n'est pas un problème car SAMARAH, par construction, supprimera (en fusionnant ou en re-clustering) les clusters vides à la fin du processus de résolution des conflits.

**Algorithm 3** Générer de la dissimilarité entre les agents

1: **procedure** GENERATE_DISSIMILARITY(Jeu de données $\mathcal{D}$, ensemble des agents $\mathcal{A}$, ensemble des contraintes $\mathcal{C} \subseteq \mathcal{D} \times \mathcal{D}$)
2: **for** chaque contrainte $c$, entre $c1$ and $c2$, dans $\mathcal{C}$ **do**
3: $A' = \emptyset$
4: **for** chaque agent $a_i$ dans $\mathcal{A}$ **do**
5: **if** $a_i$ ne satisfait pas $c$ **then**
6: $A' = A' \cup a_i$
7: **if** $A' \neq \emptyset$ **then**
8: Choisir aléatoirement $a$ dans $A'$ avec $Centers^a$ l'ensemble de ses centres,
9: **if** $c$ est une contrainte ML **then**
10: $Centers^a = Centers^a \cup \{average(\{c1; c2\})\}$
11: **if** $c$ est une contrainte CL **then**
12: $Centers^a = Centers^a \cup \{c1; c2\}$

La méthode finale est résumé dans la figure 2.7.

### Comment gérer les contraintes précédemment fournies à Samarah ?

La méthode I-SAMARAH permet à l'utilisateur de donner des contraintes entre deux itérations pour faire évoluer le résultat courant, mais l'utilisation des contraintes précédentes

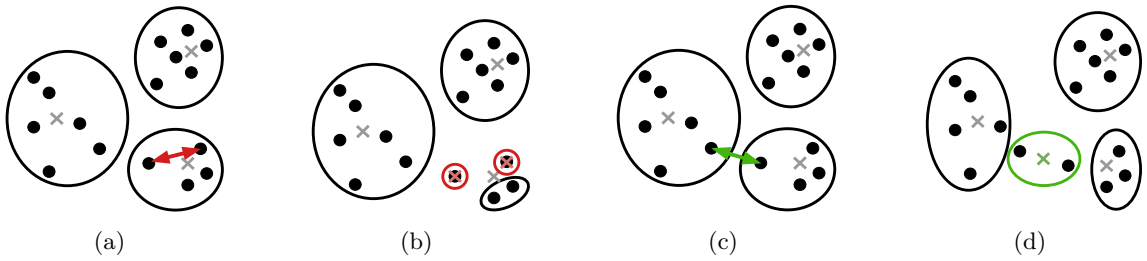



FIGURE 2.6 – Illustration de la génération de dissimilarité lorsqu'une contrainte CL (a) est utilisée avec le résultat qui en résulte (b), respectivement (c) et (d) avec une contrainte ML. Les croix représentent les centres des clusters

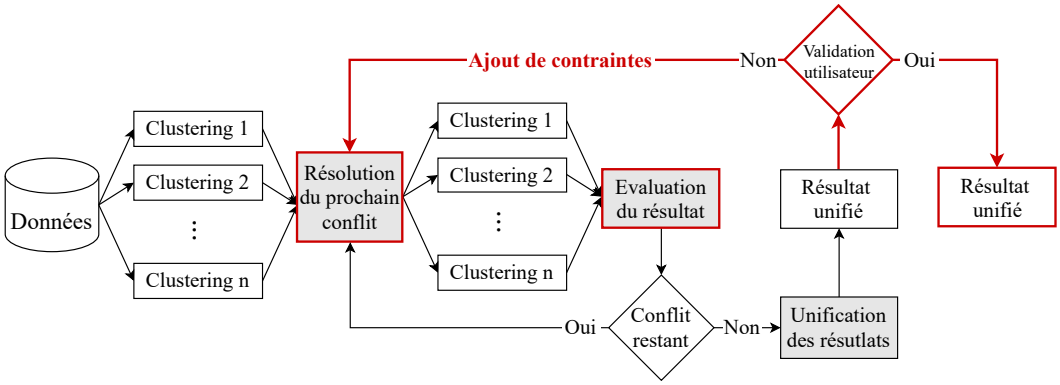



FIGURE 2.7 – La méthode I-SAMARAH

peut être remise en question. Il existe alors différentes stratégies :

— cumulative : l'utilisateur conserve toutes les contraintes précédentes. D'une part, cette stratégie augmente les connaissances injectées à chaque itération. D'autre part, de nouvelles contraintes peuvent générer des incohérences avec les anciennes, ce qui peut être difficile à évaluer pour l'utilisateur. De plus, à mesure que le nombre de contraintes augmente, elles vont amoindrir le poids de chacune des contraintes (anciennes comme nouvelles). Il est toutefois à noter que SAMARAH, de par la relaxation de la satisfaction des contraintes, peut gérer des contraintes incohérentes.

— non-cumulative : l'utilisateur supprime toutes les contraintes après chaque incrémentation. On considère alors qu'elles sont déjà partie intégrante du résultat courant. Ceci a l'avantage de garantir que les nouvelles contraintes auront un effet, car aucune contrainte antérieure ne peut créer de conflits ou minimiser leur importance. Cependant, le nouveau résultat peut donner lieu à des modifications de l'assignation de certains objets qui étaient auparavant contraints.

— semi-cumulative : l'utilisateur sélectionne un sous-ensemble de contraintes précédentes à conserver. Cela peut équilibrer le problème des deux stratégies précédentes. Cependant, cette stratégie suppose un mécanisme de choix pour sélectionner les contraintes à conserver qui peut être difficile à définir. En effet, ce mécanisme doit prendre en compte l'incohérence potentielle avec les nouvelles contraintes mais aussi le gain obtenu des contraintes précédentes. Cela devrait faire l'objet de recherches supplémentaires et n'est pas couvert par cette thèse.

Dans notre étude, seules les stratégies cumulatives et non cumulatives seront évaluées dans les expériences.

### 2.5.4 Discussion et améliorations potentielles

Comme nous l'avons vu dans la section 2.5.3, la reprise de l'exécution de SAMARAH pourrait se faire de manière différente, en choisissant d'interrompre le processus de collaboration. Ce choix permettrait de réduire en partie le temps de calcul de méthode, mais il faut également prendre en compte d'autres éléments. Même si le but principal pour l'utilisateur est que le nouveau clustering prenne en compte les nouvelles informations fournies (contraintes, étiquetage, ...), il faut aussi que le nouveau clustering soit aussi similaire que possible au clustering précédent. Ceci est primordiale car l'utilisateur doit pouvoir identifier rapidement les principales zones de changement entre deux incrémentations. On peut également supposer que tant que l'utilisateur n'apporte pas d'annotations sur un cluster, ce cluster peut être considéré par défaut comme valide pour l'utilisateur et ne nécessitant pas de modifications particulières. De plus, l'intégration des retours utilisateurs suppose la possibilité de lancer le processus de clustering de manière itérative pour que l'utilisateur puisse faire graduellement évoluer le clustering. Pour que l'option de l'interruption du processus collaboratif soit mis en oeuvre il faudrait donc déterminer à quelle étape du processus l'utilisateur doit pouvoir ajouter des contraintes sur le résultat courant.

Soit $n_{cs}$, le nombre de conflits traités par le processus de collaboration de SAMARAH entre deux incrémentations. On peut distinguer deux cas :

— $n_{cs}$ est petit : ce cas assure une stabilité du résultat. En effet le nombre de conflits résolus, et donc le nombre de modifications apportées au clustering sera limité, donc plus simple à assimiler par l'utilisateur. Cependant, cela a aussi pour conséquence que le nombre de clusters modifiés est limité, et donc que potentiellement les clusters impliqués dans de nouvelles contraintes ne sont pas modifiés. De ce fait, cela réduit fortement l'impact des contraintes entre deux incrémentations, ce qui peut décourager l'expert. De plus, même si un $n_{cs}$ petit peut mener à un nombre suffisant de phase d'incrémentations , il peut en fait s'avérer trop élevé. En effet, si le processus s'interrompt de manière trop régulière, l'utilisateur se retrouve trop sollicité (même s'il peut ne pas donner de contraintes entre chaque incrémentation) et donc peu enclin à aller au bout du processus, surtout si les modifications sont trop mineures.

— $n_{cs}$ est grand : inversement au cas précédent, le grand nombre de conflits résolus risque de fortement modifier le clustering. Ceci peut rendre l'analyse du nouveau résultat unifié difficile pour l'utilisateur. Mais dans le même temps cela augmente la probabilité de prise en compte de l'ensemble des contraintes. De plus, un $n_{cs}$ grand peut trop fortement limité le nombre d'incrémentation, l'utilisateur n'ayant alors pas assez de "d'essais" pour obtenir un résultat satisfaisant. Enfin, il est à noter que le nombre restant de conflits à résoudre peut ne pas être connu à un moment donné du processus car la résolution d'un conflit peut générer de nouveaux conflits, ce qui ne permet pas à l'utilisateur de savoir combien il reste d'incrémentations avant la fin du processus.

Une autre option pourrait être de se baser sur l'évolution du critère global $\Gamma$ au cours de l'apprentissage, avec par exemple une interruption après une stagnation prolongé de $\Gamma$. Cependant cette option soulève les même problèmes que précédemment. En effet, si l'interruption survient trop tôt on se rapporte au cas d'un $n_{cs}$ trop petit ou trop grand si elle intervient trop tard. De plus, la maîtrise d'un nombre d'interruptions suffisant devient dans ce cas d'autant plus complexe à assurer.

Ces différentes observations ont appuyé notre choix d'exécuter l'ensemble du processus collaboratif de SAMARAH à chaque incrémentation. Au regard de ce que nous avons évoqué précédemment, cette solution a plusieurs avantages :

— Le problème du choix d'un $n_{cs}$ fixe et d'un nombre d'incrémentations déterminé ne se pose plus, laissant ainsi plus de marge de décision à l'utilisateur.

— Sur la stabilité du résultat, le fait d'avoir augmenter la consensualité entre les résultats devrait permettre de réduire les perturbation du clustering sur les clusters non

contraints. En effet, nous avons pu constater expérimentalement qu'une nouvelle incrémentation sans contrainte ne modifie le clustering unifié que marginalement.

— Sur la résolution des contraintes, le fait de redémarrer le processus collaboratif de SAMARAH au complet permet d'obtenir une forte probabilité que les clusters impliqués dans les contraintes feront l'objet d'une résolution de conflit. De plus, cette solution assure une plus forte inconsistance des contraintes, car l'annotation est faite sur le résultat final de clustering et non un état intermédiaire (état intermédiaire qui aurait potentiellement pu mener à une résolution non supervisée des contraintes).

. Toutefois, une étude plus poussée devrait être menée pour analyser le comportement de ces différentes implémentations en pratique.

Dans le chapitre suivant nous allons appliquer et évaluer notre méthode dans le cadre de la télédétection.

# Chapitre 3

# Analyse de séries temporelles en télédétection

**Sommaire**

Ce chapitre décrit l'évaluation de notre méthode I-Samarah sur les données de télédétection. Nous avons voulu nous placer dans un cadre réel pour valider l'utilisation de notre méthode dans la pratique. Nous présentons tout d'abord les données ayant servi aux expérimentations (section 3.1). Nous présentons ensuite le protocole d'évaluation que nous avons suivi (section 3.2) et les résultats obtenus (sections 3.3), puis nous finissons par une discussion sur les limitations et perspectives de notre méthode (section 3.4).

## 3.1 Contexte

Dans cette section, nous revenons plus en détail sur le contexte de la télédétection (section 3.1.1). Nous présenterons également les jeux de données qui seront utilisés pour illustrer ces problématiques (section 3.1.2).

### 3.1.1 Les Séries Temporelles d'Images Satellites (STIS)

Le lancement, en mars 2017, du deuxième satellite de la constellation Sentinel-2 [1] par l'ESA (European Space Agency) marque un changement de paradigme dans le traitement et l'analyse des Séries Temporelles d'Images Satellites (STIS). Venant s'ajouter à d'autres programmes existants, tel que Landsat, Spot ou encore Venus, il est dorénavant possible d'accéder à des images multi-spectrales à la fois à haute résolution et à haute fréquence d'acquisition, gratuites, simples d'accès et simples d'utilisation, et ce sur l'ensemble du globe. Ces acquisitions quasi-continues débouchent sur une génération de données sans précèdent. A raison d'une capture tous les cinq jours, les satellites Sentinel-2 ont déjà permis l'obtention de 292 images rien qu'entre mars 2017 et mars 2021. Pour une tuile (unité de découpe des images Sentinel), qui couvre une zone grande comme l'Ile de france, cela représente pas loin de 1,5To de données. Ces séries temporelles d'images ouvrent de nouvelles perspectives en renforçant les domaines d'étude existants en télédétection, tels que le suivi agricole [Ottosen et collab., 2019; Russwurm et Körner, 2018], l'occupation des sols [Ienco et collab., 2019], la sylviculture [Grabska et collab., 2019], mais également en créant de nouvelles opportunités tel que le suivi d'activité minière [Lobo et collab., 2018], la détection de glissements de terrain [Lacroix et collab., 2018] ou la cartographie d'urgence avec Copernicus EMS (Emergency Management Service) [2].

Cependant, une fois les données acquises il est nécessaire de les analyser pour en tirer des informations utiles. Même si ces images sont de plus en plus faciles à manipuler techniquement, elles restent complexes à étudier pour un expert thématique de part leur taille (une tuile Sentinel représentant ∼110 x 110 $km^2$ a une taille de ∼11k x 11k pixels), leur nombre de bandes spectrales (12 bandes) et surtout leur dimension temporelle. En effet, il est très difficile, même pour un expert du domaine, d'analyser l'évolution d'une zone géographique dans sa globalité à travers une suite d'images. Cela est d'autant plus difficile que l'évolution peut être diffuse sur une longue plage temporelle ou restreinte à une ou plusieurs petites portions de la zone observée.

Pour illustrer cela, nous allons présenter les deux jeux de données que nous avons sélectionnés pour nos expériences.

### 3.1.2 Données utilisées

Pour évaluer notre méthode I-Samarah, les jeux de données sélectionnés sont, pour le premier, un cas de surveillance agricole (illustre une application fréquente de télédétection) et pour le second, la détection des coupes franches d'arbres sur une période de 3 ans.

1. https://sentinel.esa.int/web/sentinel/missions/sentinel-2
2. https://emergency.copernicus.eu/

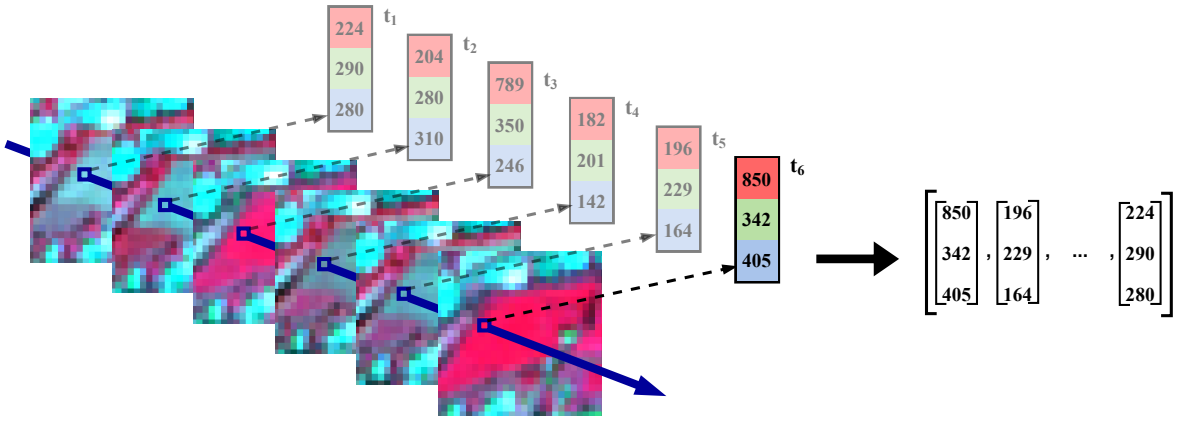



FIGURE 3.1 – Illustration de la création des séries temporelles d'images satellites.

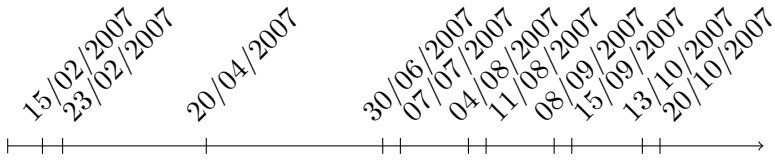



FIGURE 3.2 – Dates d'acquisition des images utilisées pour le jeu de données *Cultures*

### Génération des séries temporelles

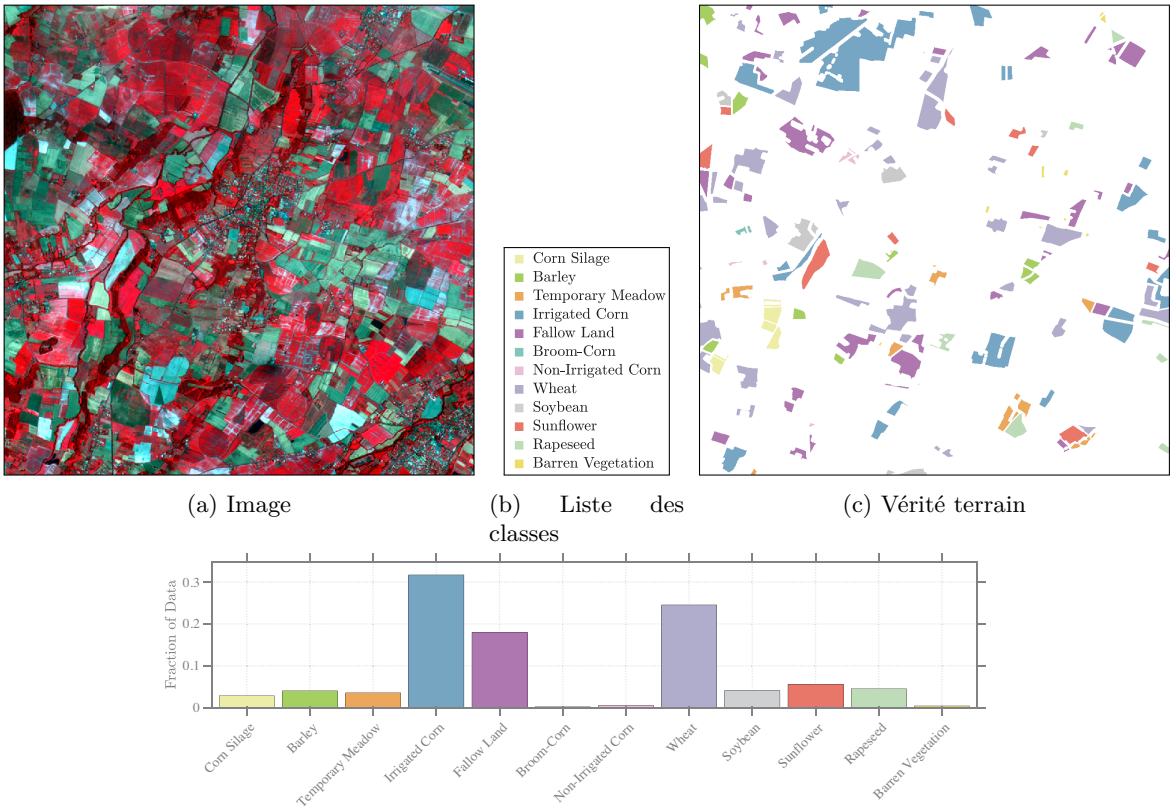
Les jeux de données sont issus de séries d'images satellites Sentinel-2. Avant toute opération, les images sont recalées pour avoir le même géo-référencement. Pour chaque pixel, sa valeur radiométrique correspondante dans chaque image de la série est extraite sous la forme d'un vecteur (de dimension égale au nombre de canaux dans l'image). Ces vecteurs sont ensuite concaténés chronologiquement dans une séquence et intégrés au jeu de données temporelles mutivariées (voir figure 3.1). Dans nos expériences, nous nous sommes limités à des échantillons afin de limiter les temps d'exécution des calculs et les besoins en mémoire vive.

### Jeu de données *Cultures*

**Motivation :** Le suivi de l'agriculture est essentiel pour de nombreuses applications, telles que l'estimation de la production, les décisions d'appui sur l'approvisionnement en eau ou encore le contrôle des subventions publiques. Les images de télédétection sont utiles dans ce processus, car elles fournissent des observations avec une résolution spectrale, spatiale et temporelle élevées. Par conséquent, la classification des cultures fait l'objet de nombreuses publications en télédétection [BELGIU ET CSILLIK, 2018; LAMPERT ET COLLAB., 2019; SAKAMOTO ET COLLAB., 2005]. De plus, l'évolution temporelles des parcelles agricoles est une caractéristique importante pour la classification des cultures, car la phénologie des cultures est essentielle pour faire la distinction entre les différents types [LAMPERT ET COLLAB., 2019; SAKAMOTO ET COLLAB., 2005].

Cependant, les données de référence sont souvent partielles, car elles ne couvrent qu'une partie des classes ou que certaines régions. C'est surtout un problème pour le suivi agricole car la phénologie peut varier en fonction du climat local. Ces variations limitent souvent la généralisation des méthodes supervisées.

**Description des données :** Le jeu de données porte sur une zone située près de Toulouse, d'une taille de $1000 \times 1000$ pixels, soit une superficie de $8km \times 8km$. Les données sont constituées de 11 images multi-spectrales échantillonnées sur une période de huit mois en




(a) Image (b) Liste des classes (c) Vérité terrain

(d) Distribution des classes

FIGURE 3.3 – Jeu de données *Culture* sur des données réelles de parcelles agricoles : 12 classes, et série de 11 images (la quatrième date du 30/06/2007 est affichée).

2007 (voir figure 3.2). Les images multi-spectrales (bandes verte, rouge et proche infrarouge) ont été capturées par le satellite Formosat-2 et ont été fournies par le *Centre d'Études Spatiales de la Biosphère (CESBIO) Unité Mixte de Recherche CNES-CNRS-IRD-UPS* de Toulouse. L'exemple d'une des images de la série temporelle (le 30/06/2007) est présentée dans la figure 3.3a. Un sous-ensemble aléatoire de pixels de l'image a été échantillonné et pour chacun des pixels une série temporelle a été générée en extrayant la valeur du pixel à chaque date de la série d'images comme illustré dans la figure 3.1. A noter que pour ce jeu de données, seul les régions pour lesquelles des données de référence sont disponibles ont été utilisées pour l'échantillonnage (les zones non blanches dans la figure 3.3c). L'ensemble de données utilisées pour les expériences totalise 9869 séries temporelles (environ 20% des données annotées).

### Jeu de données *Coupes franches*

**Motivation :** Le choix des coupes franches d'arbres est motivé par trois raisons principales.

Tout d'abord, la détection des coupes franches est une demande des services nationaux de gestion de forêt. Il existe actuellement une forte demande de bois d'origine écologique et en même temps une grande quantité d'arbres abattus dans les forêts européennes. Ainsi, il est nécessaire de surveiller les coupes d'arbres, pour contrôler les opérations de replantation afin de préserver la durabilité des exploitations, mais également pour lancer des incitations de coupes d'arbres là où elles sont nécessaires. Des articles ont déjà été publiés dans le domaine [COHEN et collab., 1998; SAKSA et collab., 2003] et il existe un intérêt politique récent pour le suivi de ces activités, avec par exemple le projet INTERREG RegioWood II [3].

Deuxièmement, la détection des coupes franches reste difficile, car l'objectif est de détecter les coupes d'arbres pouvant se produire à des dates différentes et donc n'apparaître que sur

3. http://www.regiowood2.info

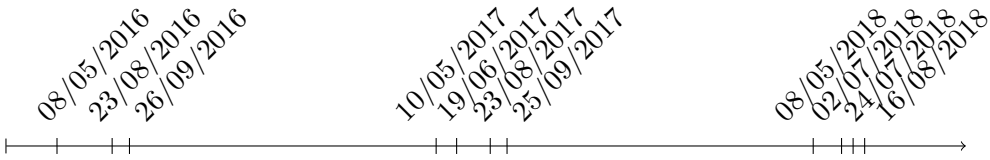
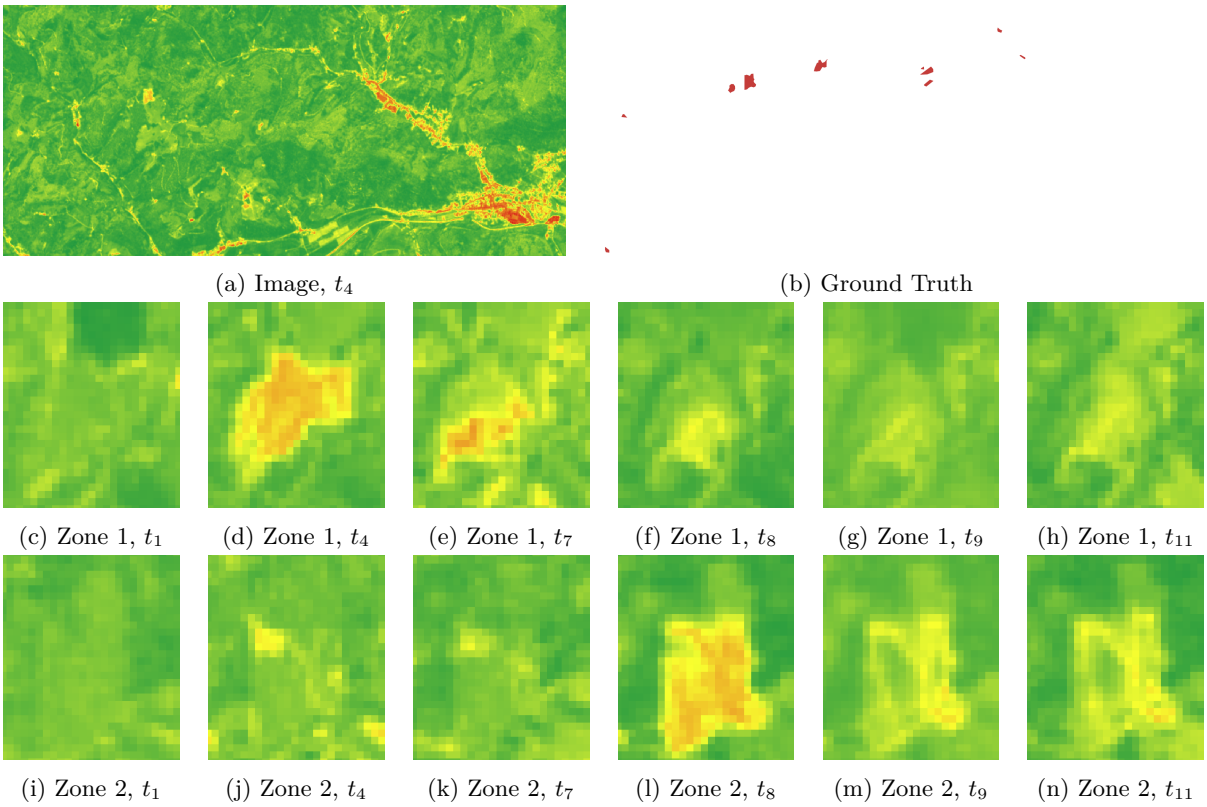



FIGURE 3.4 – Dates d'acquisition des images utilisée pour le jeu de données *Coupes franches*



(a) Image, $t_4$ (b) Ground Truth

(c) Zone 1, $t_1$ (d) Zone 1, $t_4$ (e) Zone 1, $t_7$ (f) Zone 1, $t_8$ (g) Zone 1, $t_9$ (h) Zone 1, $t_{11}$

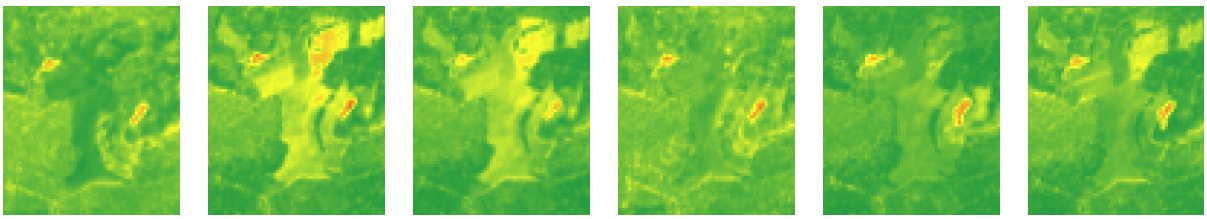
(i) Zone 2, $t_1$ (j) Zone 2, $t_4$ (k) Zone 2, $t_7$ (l) Zone 2, $t_8$ (m) Zone 2, $t_9$ (n) Zone 2, $t_{11}$

FIGURE 3.5 – (a) Images d'NDVI des coupes franches ($t_8$ est affiché), (b) vérité terrain des zones de coupes franches, (c) à (h) et (i) à (n) montrent l'évolution respective des zone 1 et 2 échantillonnées sur les dates $t_1$, $t_4$,$t_7$, $t_8$, $t_9$, $t_{11}$. Plus la couleur tend vers le vert, plus le signal d'NDVI est fort.

quelques images de la série temporelle. De bons, voir très bons, résultats peuvent être obtenus avec des méthodes supervisées [BUCHA et STIBIG, 2008; SAKSA et collab., 2003]. Néanmoins, elles nécessitent souvent beaucoup d'annotations ou des systèmes ad hoc complexes. Pour les coupes franches d'arbres, l'annotation est particulièrement complexe et prend du temps, car elle représente souvent une très petite fraction des données. De plus, la revégétalisation rend la détection difficile si l'on considère uniquement une comparaison binaire entre la première et la dernière date (voir figure 3.5). Ainsi, la coupe qui a lieu dans la zone 1 en $t_4$ a totalement disparue en $t_{11}$. Cela peut engendrer une confusion avec les fauches de prairies ou les récoltes agricoles. Un exemple de praire fauchée est donné dans la figure 3.6. Par conséquent, la classe des coupes franches d'arbre est très difficile à extraire avec les méthodes de clustering. Néanmoins, une fois qu'une coupe est détectée, elle peut être facilement appréhendée visuellement par l'expert pour, par exemple, être validée ou analysée. Il lui suffit de parcourir la série d'images en se concentrant sur l'emplacement concerné.

Troisièmement, dans notre cas spécifique, nous pouvons nous appuyer sur un groupe d'experts thématiques du SERTIT [4] capables d'évaluer les résultats du clustering et de fournir des retours sur la validité des clusters.

4. https://sertit.unistra.fr/



(a) $t_1$ (b) $t_2$ (c) $t_3$ (d) $t_4$ (e) $t_5$ (f) $t_6$

FIGURE 3.6 – Images d'NDVI qui montrent l'évolution d'une zone de prairie fauchée sur les dates $t_1$ à $t_6$. Plus la couleur tend vers le vert, plus le signal d'NDVI est fort.

**Description des données :** Une zone des Vosges (Alsace, France), d'une taille de $724 \times 337$ pixels, a été sélectionnée pour cette étude, voir figure 3.5a. La série temporelle est composée de 11 images échantillonnées sur 3 ans (voir figure 3.4). L'ensemble de données contient 10 zones de coupes : 8 apparaissent à la $4^{\text{ème}}$ date (10/05/2017) et 2 à la $8^{\text{ème}}$ (08/05/2018). Les images sont composées d'une seule bande qui contient les valeurs calculées du NDVI (Normalized Difference Vegetation Index) qui est l'indice de végétation le plus couramment utilisé. Les images d'indice ont été calculées à partir des données Copernicus Sentinel-2 (tuile T32ULU) traitées au niveau 2A/3A par le *CNES* pour le *Theia Land Data Center*. Un sous-ensemble de la zone a été sélectionné aléatoirement pour obtenir un échantillon de 40 000 séries temporelles de pixels (environ 16% des données). Dans la suite, les coupes franches d'arbres seront simplement appelées coupes ou coupes d'arbres.

## 3.2 Protocole d'évaluation

Dans cette section, nous présentons les différents éléments utilisés pour cette évaluation.

### 3.2.1 Méthodes de clustering sous contraintes

Pour cette évaluation, nous avons comparé notre méthode à celles présentées dans l'étude préliminaire (voir section 2.4.1) à l'exception des méthodes spectrales. En effet, les performances de ces algorithmes dépendent fortement de la valeur des hyper-paramètres qui contrôlent la connectivité dans le graphe de similarité et le nombre de vecteurs propres utilisés pour le clustering. Or, le choix de ces paramètres n'est pas intuitif pour l'utilisateur et influe fortement sur les performances de ces méthodes [VON LUXBURG, 2007]. Pour illustrer cela, nous avons reporté dans la figure 3.7 la variation de l'ARI sur le jeu de données *Cultures* lorsque différentes combinaisons de paramètres sont choisies par *grid search* pour les algorithmes Spec et CCSR. Étant donné les forts écarts d'ARI, il devenait difficile de fixer ces paramètres tout en permettant une comparaison équitable avec des méthodes non supervisées, ce qui a motivé cette exclusion.

Nous n'avons donc conservé que COP-KMEANS et CPClustering ainsi que SAMARAH (non-incrémentale). Nous avons par contre ajouté une autre méthode de clustering sous contraintes : MIP-KMEANS. Cette méthode, proposée par BABAKI [2017], est une extension de l'algorithme KMEANS, mais elle tire parti du solveur MIP (Mix Integer linear Programming model) pour réduire la sensibilité des K-moyennes à l'initialisation et sa tendance à se retrouver dans des optima locaux. Les contraintes sont ajoutées au solveur MIP. Le seul paramètre requis par cette méthode est le nombre de clusters. Nous utilisons l'implémentation disponible en ligne fournie par les auteurs [BABAKI, 2017] [5].

Nous avions également prévu de tester la méthode COBRAS [VAN CRAENENDONCK et collab., 2018a], qui est une méthode de clustering sous contraintes active. Son principe est de construire des super-instances d'objets et de les diviser ou de les fusionner selon les

5. https://github.com/Behrouz-Babaki/MIPKmeans

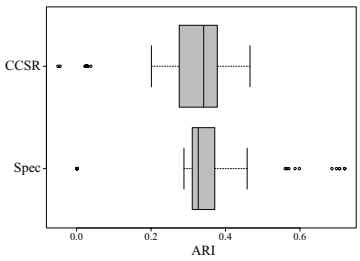



FIGURE 3.7 – Distribution du score d'ARI pour les méthodes de clustering spectral pour différentes valeurs de paramètres choisi par *grid search*. Spec – moyenne : 0.367, min : 0.001, max : 0.720, écart type : 0.184. CCSR – moyenne : 0.297, min : −0.050, max : 0.466, écart type : 0.139.

contraintes données par l'utilisateur. La méthode commence par créer une super-instance, qui contient tous les objets du jeu de données. Cette super-instance est ensuite divisée de manière itérative en super-instances, en continuant avec celle ayant la cardinalité la plus haute, en prenant en compte une pondération par une matrice de similitude. Après cela, l'algorithme détermine la relation entre les nouveaux clusters. Pour cela, COBRAS interroge l'oracle (l'expert) sur la relation (must-link/cannot-link) entre les médoïdes des super-instances les plus proches. L'expert arrête l'algorithme lorsqu'il obtient un résultat satisfaisant. Cette méthode a également l'avantage d'être itérative, car de nouvelles contraintes peuvent être demandées à partir d'un résultat intermédiaire, mais également de proposer une version pour les séries temporelles [VAN CRAENENDONCK et collab., 2018b]. Cependant, lorsqu'une contrainte est générée par la méthode et proposée à l'expert pour validation, ce dernier ne peut répondre que par "oui" ou "non". Malheureusement, il ne peut pas toujours avoir de réponse aussi binaire et il devrait pouvoir répondre au moins par "je ne sais pas". Les expériences montrent que c'est particulièrement le cas dans le jeu de données *Coupes franches*. Par conséquent, cette méthode n'a pas été incluse dans les méthodes comparées.

### 3.2.2 Génération des contraintes

#### Stratégies alternatives

Pour toutes les méthodes de clustering sous contraintes, nous utilisons les contraintes fournies par les experts. Mais nous avons vu qu'il existe des méthodes pour la sélection des contraintes actives (section 2.5.1). Par conséquent, nous avons sélectionné certaines de ces méthodes pour générer des contraintes afin de comparer leurs performances aux contraintes données manuellement par les experts :

— MinMax : MALLAPRAGADA et collab. [2008] ont proposé une approche basée sur la construction de voisinages des objets qui fonctionne en deux phases. Dans la première phase, *Explorer*, la méthode construit $K$ voisinages disjoints, $X_n$, où $K$ est le nombre total de clusters, appelé le squelette. Ce dernier est construit en sélectionnant itérativement l'objet le plus éloigné du squelette existant (en utilisant la méthode du *farthest-first traversal* [ROSENKRANTZ et collab., 1977]) jusqu'à ce que l'utilisateur indique que l'objet courant n'appartient plus à aucun des voisinages existants. La phase d'exploration s'arrête lorsque $K$ points ont été trouvés de telle sorte qu'il y ait une contrainte CL entre n'importe quelle paire des $K$ points. Dans la deuxième phase, *Consolider*, MinMax augmente progressivement le nombre d'objets dans l'ensemble de voisinages en sélectionnant l'objet le plus incertain, $q$, dans l'ensemble des objets restantes, $S$, selon le critère MinMax suivant :

$$q = arg \min_{x_i \in S} ( \max_{x_j \in X_n} (similarity(x_i, x_j)) \tag{3.1}$$

Ensuite, des requêtes sont soumises à l'utilisateur jusqu'à ce qu'une contrainte must-link soit trouvée entre $q$ et un voisinage existant. L'ensemble des voisinages est trié en

fonction de la similitude pour minimiser le nombre de requêtes. $q$ est ensuite ajouté au voisinage pour lequel une contrainte must-link est retournée par l'utilisateur. Les contraintes sont générées à la fin en créant des contraintes ML entre les objets du même voisinage et CL entre les objets de voisinages différents.

— NPU (Normalized Point-based Uncertainty) : Cette méthode, proposée par Xiong et collab. [2013], repose également sur la construction de voisinages, mais de façon itérative. Elle alterne entre regroupement des données et ajout de nouveaux objets dans l'ensemble des voisinages $N$. Les nouveaux objets à ajouter aux voisinages sont sélectionnés en basant sur une métrique de similarité, $similarity_{RF}$, calculée à partir d'un classifieur *Random Forest* sur des étiquettes de clustering. Cependant, au lieu d'utiliser uniquement les objets les plus dissemblables, comme MinMax, NPU choisit l'objet $x$ qui maximise le gain d'information, $H(N|x)$, pour un coût minimal attendu de requêtes, $E[query(x)]$. $E[query(x)]$ peut être vu comme le nombre attendu de requêtes à soumettre pour ajouter $x$ à un voisinage de $N$. Pour calculer ces valeurs, les auteurs ont défini la probabilité qu'un objet $x$ appartienne au voisinage $N_l$, à partir de l'ensemble des voisinages $N$, par :

$$p(x \in N_l) = \frac{\frac{1}{|N_l|} \sum\limits_{x_j \in N_l} similarity_{RF}(x, x_j)}{\sum\limits_{N_p \in N} \frac{1}{|N_p|} \sum\limits_{x_j \in N_p} similarity_{RF}(x, x_j)} \tag{3.2}$$

Partant de cette équation, les auteurs définissent l'incertitude d'un objet $x$ par l'entropie de son appartenance au voisinage, par :

$$H(N|x) = -\sum_{N_p \in N} p(x \in N_i) log_2 p(x \in N_i) \tag{3.3}$$

Le coût de l'espérance de sélectionner cet objet est alors :

$$E[query(x)] = \sum_{N_p \in N} i * p(x \in N_i) \tag{3.4}$$

L'objet $q$ à sélectionner, à partir de l'ensemble des objets restants $S$, est alors celui qui présente à la fois une incertitude élevée et un faible coût d'espérance :

$$q = arg\max_{x_i \in S}\left(\frac{H(N|x_i)}{E[query(x_i)]}\right) \tag{3.5}$$

Enfin, l'ensemble des voisinages est mis à jour de la même manière que pour MinMax (équation 3.1), mais le tri par similarité est basé sur la valeur $H(N|x)$. Les contraintes sont générées à la fin en créant des contraintes ML entre les objets du même voisinage et CL entre les objets de voisinages différents.

Pour le jeu de données *Coupes franches*, nous devons tenir compte du fait que nous ne connaissons que les étiquettes sur les coupes d'arbres. Cela implique que nous ne pouvons générer que des contraintes incluant au moins un pixel d'une coupe d'arbre. Par conséquent, pour MinMax et NPU l'objet initial du voisinage est choisi parmi les pixels de coupes d'arbres, car les contraintes ML ne peuvent pas être générées entre des objets qui ne sont pas étiquetés comme coupe d'arbres. Pour MinMax, nous avons également modifié l'appel à la fonction *argmin*, dans l'équation 3.1, en ignorant l'objet $x_i$ dans *argmin* si les deux conditions suivantes sont satisfaites :

— $x_i$ n'est pas un pixel de coupe d'arbre ;

— l'objet le plus similaire dans les autres voisinages, $\max_{x_j \in X_n}(similarity(x_i, x_j))$, n'est pas dans le voisinage qui regroupe les coupes d'arbres.

De cette manière, un objet, n'étant pas étiqueté comme faisant partie d'une coupe, ne sera ajouté à $X_n$ que si le voisinage le plus similaire est celui des coupes d'arbres.

Pour NPU, nous avons appliqué le même principe et nous avons modifié l'équation 3.5 en ignorant $x_i$ s'il remplit les deux conditions suivantes :

— $x_i$ n'est pas un pixel de coupe d'arbre ;

— la probabilité qu'$x_i$ appartienne au voisinage $N_l$, $p(x_i \in N_l)$, est maximale et $N_l$ n'est pas le voisinage des coupes d'arbres.

Pour nos expérience nous avons utilisé le package python `active-semi-supervised-clustering` qui implémente ces deux méthodes [6].

#### Retours utilisateurs et leurs simulations

Avec notre approche, nous nous appuyons sur les experts. Et sur leurs intuitions et connaissances, pour donner des contraintes utiles en prenant en compte le résultat du clustering courant.

Pour le jeu de données *Coupes franches*, nous avons pu rassembler un groupe d'experts en télédétection qui ont accepté de participer aux expériences, sans avoir pris connaissance au préalable de la vérité terrain. Les expert ont pour cela utilisé une des deux zones de coupes (figure 3.5) comme exemple de départ pour analyser les clusters recouvrant cette coupe (étape 2.a - figure 2.5). Ils ont annoté le résultat avec des contraintes must-link et cannot-link pour indiquer leur accord ou désaccord sur ces clusters (étape 2.c - figure 2.5). Une fois les contraintes données, les experts relançaient le cycle de clustering (étape 2.d - figure 2.5). Cette suite d'opérations, définie comme une incrémentation, a été réalisée de manière itérative jusqu'à ce que les experts ont estimé que le clustering obtenu était satisfaisant ou que les contraintes n'amélioraient plus le clustering (étape 2.b - figure 2.5).

Pour le jeu de données *Cultures*, aucun expert n'était disponible, car cela nécessite d'une part, d'avoir une bonne connaissance de la phénologie des différents types de cultures dans la zone sélectionnée et d'autre part, un travail important et fastidieux. Par conséquent, nous avons défini un mécanisme générique pour simuler l'expert. Nous avons vu dans la section 2.4.2 que si les contraintes sont sélectionnées au hasard, la plupart d'entre elles sont implicitement respectées par l'algorithme de clustering (principe de consistance). On peut donc en conclure que la plupart d'entre elles ont peu ou pas d'impact sur le résultat. Par contre, en pratique, nous avons observé que les utilisateurs ont tendance à donner majoritairement des contraintes qui ne sont pas encore satisfaites par le résultat actuel. Par conséquent, nous avons simulé le retour d'expérience des experts en ne sélectionnant que les contraintes non satisfaites par le clustering actuel.

Pour cela, nous avons utilisé une vérité terrain pour générer une grand nombre de contraintes aléatoirement sur le même mode que précédemment : deux pixels sont choisis aléatoirement et une contrainte est ajoutée entre eux de type must-link si les deux pixels sont de la même classe et de type cannot-link sinon. Une fois cet ensemble établi, à chaque phase d'annotation nous choisissons uniquement des contraintes non-satisfaites parmi celles de l'ensemble de contraintes. A noter que les méthodes de clustering sous contraintes n'ont pas d'accès direct aux vérités terrain. La génération des contraintes par ce mécanisme se fait à chaque phase d'incrémentation d'I-Samarah.

### 3.2.3 Choix de la mesure de la mesure de dissimilarité

Comme nous l'avons vu dans la section 1.1, il existe un grand nombre de métriques utilisables pour le clustering de séries temporelles, le choix étant fortement dépendant de la nature du phénomène temporel étudié. Dans notre première étude nous avons fait le choix de sélectionner DTW (voir section 2.4.1). Néanmoins, Samarah n'utilise pas de distance/métrique

6. https://pypi.org/project/active-semi-supervised-clustering/

pour piloter la collaboration. Seuls les métriques utilisées par les agents ont une influence sur les résultats. Sur le cas des données de télédétection, il s'avère que l'utilisation de DTW n'est pas toujours justifiée, notamment sur le jeu de données *Cultures*, où dans une étude préliminaire nous avons montré que la distance euclidienne est plus adéquate. En effet, les résultats dans [Lampert et collab., 2019] ont montré un gain moyen de 0.04 de score d'ARI lors de l'utilisation de la distance Euclidienne à la place de DTW. Cela peut s'expliquer par un calendrier assez figé des récoltes des différentes cultures. De plus, la distance Euclidienne permet de réduire fortement les temps de calcul en comparaison avec DTW.

En conséquence, pour la mesure de dissimilarité et la moyenne de Cop-Kmeans, Mip-Kmeans et MinMax, nous utilisons respectivement la distance euclidienne et la moyenne arithmétique pour le jeu de données *Cultures* et la mesure DTW et l'algorithme DBA pour le jeu de données *Coupes franches*. Comme évoqué précédemment, l'adaptation de I-Samarah aux séries temporelles consiste à utiliser des agents qui gèrent des séries temporelles. Comme nous n'avons utilisé que des agents Kmeans, nous avons fait les mêmes choix que pour Cop-Kmeans et Mip-Kmeans.

Pour CPClustering, la méthode nécessite le calcul d'une matrice de dissimilarité. Elle est calculée avec la distance euclidienne pour le jeu de données *Cultures* et DTW pour le jeu de données *Coupes franches*.

A noter que la méthode *Random Forest*, utilisée dans la méthode NPU, ne nécessite aucune modification, car elle ne repose pas sur une distance pour la classification. De plus, *Random Forest* est souvent utilisée sans adaptation pour la classification des séries temporelles en télédétection [Kane et collab., 2014; Tatsumi et collab., 2015].

### 3.2.4 Méthodologie de validation

Comme nous l'avons spécifié, dans toutes les expériences que nous avons menées, les méthodes sélectionnées n'utilisent aucune vérité terrain, ni aucune donnée de référence. Mais dans le domaine non supervisé, il est courant de comparer les résultats obtenus à une vérité terrain ou à des résultats de méthodes supervisées.

Pour cette évaluation nous avons utilisé une méthodologie différente pour chaque jeu de données :

**Pour le jeu de données *Cultures* :** Nous avons choisi une évaluation identique à la première étude en utilisant la mesure d'Adjusted Rand Index (ARI) entre la vérité terrain et le résultat de clustering.

**Pour le jeu de donnée *Coupes franches* :** Nous nous concentrons uniquement sur la séparation de la classe des coupes franches d'arbres des autres classes. Cela est dû au fait que c'est la seule classe pour laquelle nous disposons de données de référence. Or, une classification binaire sur des données aussi déséquilibrées et diverses ne serait pas possible dans un cadre non supervisé ou faiblement semi-supervisé. En règle générale, dans les problèmes de télédétection, le nombre de clusters dans une image est inconnu. Sur la base des recommandations des experts sur le nombre attendu de classes d'évolution dans la série d'images, ce nombre a été fixé à $k = 15$. Cependant, le vérité terrain permet d'évaluer uniquement les clusters qui incluent des coupes, les détails sur la sélection des clusters impliqués seront donnés plus loin dans la section 3.3.6.

Il faut également prendre en compte le contexte spatial des données. En effet, dans un scénario réel, l'expert peut connaître une partie des données ciblées, un sous-ensemble des coupes d'arbres dans notre cas. Cependant, ce sous-ensemble ne s'exprime pas en nombre de pixels, mais en nombre d'objets : la coupe entière dans notre cas. Ainsi quand les experts repèrent une coupe d'arbres, ils ne repèrent pas un unique pixel, mais bien l'ensemble des pixels constitutifs de cette coupe. De plus, dans l'étude de données spatiales, les objets proches spatialement ont tendance à être proches dans l'espace des données [Audebert et collab.,

2019]. De ce fait, une information sur un objet donné contient indirectement de l'information sur ses voisins spatiaux. Par conséquent, il nous a paru important d'éviter un cas où les experts cherchent visuellement l'ensemble des coupes et les annotent toutes, ce qui reviendrait à annoter indirectement l'ensemble des pixels de coupes. Un tel cas pourrait remettre en question la capacité de notre méthode à généraliser l'information apprise localement sur l'ensemble du jeu de données, l'algorithme pouvant se contenter de regrouper les données directement ou indirectement annotées. Nous avons donc demandé aux experts de concentrer leurs annotations sur une seule coupe donnée au début de l'expérience par test. A noter que ce choix ne suit pas la configuration dans [Lampert et collab., 2019] pour cet ensemble de données, où aucune restriction n'a été ajoutée, car nous voulions seulement évaluer la capacité des méthodes à gérer les contraintes. Pour atteindre notre objectif, nous avons utilisé deux coupes différentes pour illustrer l'impact sur le résultat final de commencer l'analyse à partir de coupes différentes (les zones 1 et 2 décrites dans la figure 3.5). Dans la suite, il en sera également fait référence comme la/les « zone(s) sélectionnée(s) ». Notez que cette restriction n'est utilisée que pour les annotations faites par les experts.

De plus, il est à noter que pour ce jeu de données l'évaluation se fait sur l'image complète. Cela implique que l'image entière doit être classée. Ce n'est pas un problème pour les méthodes supervisées, car elles appliquent simplement leur modèle sur l'ensemble de séries d'images. C'est également le cas des méthodes basées sur l'algorithme Kmeans, l'ensemble des centres pouvant être considéré comme un modèle de classification. Pour I-Samarah, chaque agent Kmeans applique son modèle sur toute l'image et l'unification à la fin de chaque incrémentation se fait sur ces résultats de clustering. Pour CPClustering aucun modèle ne peut être appliqué, car la méthode fonctionne sur une approche exploratoire (cf section 2.4.1), il faudra alors relancer le clustering en entier, ce qui serait beaucoup trop coûteux en temps de calcul. Nous avons donc utilisé un classifieur KNN (avec $k$ égal à trois) pour regrouper le reste de l'image. Le classifieur KNN utilise les assignations aux clusters comme données de référence.

Pour mesurer la qualité de nos résultats, nous avons utilisé la mesure F1-score, car elle prend en compte à la fois le rappel et de la précision du résultat. Mais nous ne mesurons le F1-score que sur la classe des coupes franches. Le F1-score est calculé avec la formule suivante :

$$\text{F1-score} = \frac{2 \cdot \textit{précision} \cdot \textit{rappel}}{\textit{précision} + \textit{rappel}}$$

Le F1-score atteint sa meilleure valeur à 1 (précision et rappel parfaits) et la pire à 0.

Pour les méthodes de classification supervisées, le F1-score est calculé sur les classes prédites. Cependant, pour les méthodes non supervisées, aucun cluster ne correspond par avance à une classe spécifique. Ainsi, nous devons spécifier un moyen de sélectionner le ou les clusters à prendre en compte pour l'évaluation. Dans un cas réel, l'expert sélectionnera les clusters proches de leur exemple connu, ces clusters doivent donc recouvrir une grande partie de cette zone, mais aussi être le plus précis possible. Pour cela, nous avons pris l'ensemble des clusters avec le meilleur rapport entre les pixels à l'intérieur et à l'extérieur de la zone sélectionnée et qui, pris ensemble, couvrent au moins la moitié de la zone sélectionnée. Cela permet d'inclure des clusters petits très représentatifs.

Nous avons également pris en compte le nombre absolu de coupes détectées. Dans un cas réel, l'expert est satisfait si toutes les coupes sont détectées même partiellement. Ceci est plus utile qu'un résultat où seule une partie des coupes est détectée même si pour ce sous groupe la qualité de détection (précision et rappel) est meilleure.

## 3.3 Expérimentations

### 3.3.1 Déroulé des expérimentations

Une méthode semi-supervisée est généralement évaluée en la comparant à d'autres méthodes de l'état de l'art. Cependant, dans notre cas, nous voulons démontrer que notre mé-

thode bénéficie à la fois d'une facilitation pour l'utilisateur à donner des contraintes pertinentes et de sa capacité à les prendre en compte de manière incrémentale.

Pour montrer comment la combinaison de ces deux facteurs intervient sur la qualité des résultats de clustering, nous avons testé l'effet de ces deux éléments séparément. Nous avons mené différents niveaux d'expériences sur les deux ensembles de données pour déterminer :

— Niveau 1 : la performance du clustering sans contrainte (section 3.3.2)

— Niveau 2 : la performance de la méthode de clustering sous contraintes standard (c'est-à-dire non incrémentale) avec des contraintes sélectionnées aléatoirement (section 3.3.3)

— Niveau 3 : la performance avec les retours utilisateurs et avec des méthodes non incrémentales (section 3.3.4)

— Niveau 4 : la performance avec des contraintes sélectionnées aléatoirement et une méthode incrémentale (section 3.3.5)

— Niveau 5 : la performance avec les retours utilisateurs et une méthode incrémentale (section 3.3.6)

A la fin, nous comparerons également notre méthode à des méthodes supervisées pour montrer le gain en termes d'investissement d'annotation (section 3.3.7).

Pour toutes les expériences, les résultats rapportés sont la moyenne sur 10 essais. De plus, toutes les expériences itératives/incrémentales sont lancées à partir des états initiaux calculés au niveau 1.

Dans la suite, nous définissons l'état initial de I-Samarah comme étant le résultat après une exécution de Samarah non supervisée (étape 1 - figure 2.5). L'incrémentation $i$ fait référence au résultat après un état initial suivi de $i$ incrémentations, soit la $i$ répétitions de la séquence retours utilisateur suivis d'un cycle de clustering d'I-Samarah (étapes 2.a à 2.d - figure 2.5). Le résultat évalué et celui obtenu après validation du résultat courant par l'expert (étape 3 - figure 2.5).

### 3.3.2 Clustering non-contraint (niveau 1) contre

Le but de notre processus expérimental est d'évaluer si les méthodes bénéficient de l'ajout de contraintes. Pour ce faire, nous avons réalisé les expériences du niveau 1 où chaque méthode est exécutée sans utiliser de contraintes. Les performances obtenues pour chaque ensemble de données sont données par le tableau 3.1 et serviront de base de comparaison.

| Méthode | *Cultures* (ARI) | *Coupes franches* (F1-score) |
|---|---|---|
| Cop-Kmeans | 0.420 | 0.104 |
| Mip-Kmeans | 0.407 | **0.107** |
| CPClustering | **0.681** | 0.106 |
| Samarah | 0.463 | 0.036 |

Tableau 3.1 – Niveau 1 : Performance moyenne sans contrainte par jeu de données. La meilleure performance pour chaque jeu de données est indiquée en gras

### 3.3.3 Clustering sous contraintes (niveau 2)

Pour le niveau 2, nous ajoutons des contraintes dans le processus de clustering. Pour la génération des contraintes, nous avons suivi le protocole de la première étude (voir section 2.4.1). Les contraintes ont été générées en prenant des paires de points au hasard et en générant une contrainte ML ou CL selon qu'ils appartiennent à la même classe ou non. Nous avons aussi repris l'utilisation de jeux de contraintes avec des tailles de 5%, 10%, 15% ou 50% de la cardinalité, $N$. Dix jeux de contraintes ont été générés pour chaque fraction

de contraintes pour permettre à chaque expérience d'être répétée dix fois, chacune avec un sous-ensemble aléatoire différent de contraintes (tous les algorithmes sont évalués en utilisant les mêmes ensembles de contraintes).

Pour le jeu de données *Coupes franches*, nous devons prendre en compte le fait que nous ne disposons que de contraintes qui incluent des coupes d'arbres, car nous ne connaissons pas les étiquettes des autres classes. Ainsi, pour les contraintes aléatoires, un pixel est choisi dans une coupe d'arbre, l'autre est choisi parmi tous les autres pixels de coupes ou non.

Avant dévaluer le gain à utiliser ces contraintes, nous donnons dans le tableau 3.2 les taux de consistance pour les différents algorithmes (voir définition 2.4.2) en mesurant le taux de satisfaction des contraintes par le clustering non contraint (en utilisant les ensembles de contraintes du niveau 2 de taille 50%), de manière identique au protocole de la première étude (voir section 2.4.1). Pour rappel, cela permet d'évaluer le taux de contraintes qui ont une chance d'être satisfaites par la méthode même si ces contraintes n'ont été explicitement utilisées lors du clustering.

| Méthode | *Cultures* | *Coupes franches* |
|---|---|---|
| COP-KMEANS | 0.807 | 0.917 |
| MIP-KMEANS | 0.803 | **0.920** |
| CPClustering | 0.413 | 0.877 |
| SAMARAH | **0.817** | 0.916 |

TABLEAU 3.2 – Taux moyen de satisfaction par jeu de données par les résultats du niveau 1. La meilleure performance pour chaque jeu de données est indiquée en gras

Les résultats montrent une consistance des méthodes élevée vis à vis de ces contraintes, en particulier pour le jeu de données *Coupes franches*. Cela montre que les contraintes sélectionnées aléatoirement n'apportent pas beaucoup de connaissances aux méthodes de clustering, ce qui peut réduire leur effet.

| | *Cultures* (ARI) | | | | *Coupes franches* (F1-score) | | | |
|---|---|---|---|---|---|---|---|---|
| Méthode | 5% | 10% | 15% | 50% | 5% | 10% | 15% | 50% |
| COP-KMEANS | 0.406 | 0.314 | 0.443 | 0.369 | 0.104 | 0.098 | **0.124** | 0.112 |
| MIP-KMEANS | 0.428 | 0.416 | 0.431 | 0.532 | **0.112** | **0.109** | 0.105 | 0.114 |
| CPClustering | 0.650 | 0.562 | 0.542 | 0.510 | 0.099 | 0.103 | 0.105 | 0.096 |
| SAMARAH | **0.691** | **0.682** | **0.714** | **0.702** | 0.102 | 0.097 | 0.108 | **0.121** |

TABLEAU 3.3 – Niveau 2 : Performance moyenne des clustering sous contraintes avec différentes tailles d'ensembles de contraintes aléatoires. La meilleure performance pour chaque jeu de données est indiquée en gras.

Les résultats de clustering contraint sont reportés dans le tableau 3.3. Ils n'incluent pas le taux de satisfaction, car toutes les méthodes satisfont toutes les contraintes par construction, à l'exception de SAMARAH qui obtient un taux de satisfaction moyen de 0,86 sur *Cultures* et 0,97 sur *Coupes franches*. On peut à nouveau faire le constat que les méthodes ne bénéficient pas forcément de l'information apportée par les contraintes.

Pour *Cultures*, MIP-KMEANS et SAMARAH bénéficient de l'ajout de contraintes. Pour MIP-KMEANS le gain n'est significatif que pour la taille de 50%. Pour COP-KMEANS et CP-Clustering, leurs résultats se détériorent avec les contraintes, en particulier CPClustering pour qui les performances d'ARI diminuent inversement avec le nombre de contraintes. Ce constat vient confirmer les observations faites sur les jeux de données de l'UCR où l'on constate que les méthodes permettant un relâchement de la satisfaction des contraintes donnent de meilleurs résultats (SAMARAH dans notre cas).

Pour *Coupes franches*, les résultats sont plus mitigés. SAMARAH est la seule méthode à bénéficier largement des contraintes, mais c'est aussi la méthode qui a le résultat sans contraintes le moins élevé. Globalement, aucune méthode ne donne vraiment de bons résultats pouvant justifier l'ajout de contraintes dans le processus, notamment avec le nombre de contraintes impliquées (5% de 40000 pixels/objets représente déjà 2000 contraintes).

De plus, la performance ne semble pas corrélée au nombre de contraintes, à l'exception de CPClustering sur les cultures où elle est négative, donc contradictoire avec l'intuition qu'on pourrait en avoir. Cette expérience montre que les méthodes de clustering sous contraintes ne bénéficient pas toujours des contraintes générées aléatoirement, même lorsqu'elles sont fournies en grand nombre.

### 3.3.4 Retours utilisateurs et méthodes non incrémentales (niveau 3)

Pour le niveau 3, nous proposons d'intégrer les retours des utilisateurs, mais aussi de comparer ces retours aux méthodes de sélection des contraintes actives NPU et MinMax. Cependant, dans un premier temps, nous continuons d'utiliser les méthodes de clustering contraintes standard (c'est-à-dire toutes sauf I-SAMARAH). Pour NPU et les retours utilisateurs, qui permettent à ces utilisateurs d'ajouter des contraintes basées sur un clustering existant, le processus de définition des contraintes a été répété sur quatre incrémentations. Ce processus consiste à récupérer les contraintes données par l'utilisateur (via les propositions de la méthode NPU, ou de l'utilisateur lui-même pour les retours utilisateurs) à partir d'un clustering précédent (sans contrainte pour la première incrémentation), puis à exécuter un clustering sous contraintes avec ces contraintes nouvellement acquises. A noter que les contraintes sont cumulées d'une incrémentation à l'autre. De plus, la méthode de clustering sous contraintes ne prend pas en compte l'ancien résultat de clustering, la méthode repartant d'un état initial avec les contraintes (les anciennes et les nouvelles) comme seule information supplémentaire. Pour les retours utilisateurs, le nombre de contraintes a été limité à 6 annotations par incrémentation afin de limiter l'implication de l'utilisateur. Pour *Coupes franches*, le nombre total de contraintes varie entre 13 et 21 contraintes, avec une moyenne de 16 contraintes. Pour *Cultures*, il est fixé à un total de 24 contraintes pour avoir un nombre de contraintes total équivalent à *Coupes franches*. Pour NPU, nous avons fixé le nombre de requêtes à 10 par incrémentation pour prendre en compte la plus grande simplicité d'annotation de cette méthode (et donc une possibilité pour l'utilisateur de donner plus de contraintes). Pour MinMax, une seule incrémentation est effectuée car cette méthode ne dépend pas d'un résultat précédent. Son nombre de requêtes a donc été fixé à 40 pour avoir un nombre de contraintes total équivalent à NPU. Pour cette expérience, la moyenne rapportée dans les résultats est également calculée sur 10 exécutions à l'exception des retours des utilisateurs sur *Coupes franches* avec 5 essais pour la zone 1 et 5 essais pour la zone 2. Pour rappel, les contraintes utilisateurs sont simulées pour le jeu de données *Cultures* (cf section 3.2.2). Pour les contraintes simulées, 6 contraintes non satisfaites sont sélectionnées aléatoirement parmi un des ensembles de contraintes pris au hasard de la fraction de 50% générée dans l'expérience du niveau 2.

Les résultats sont reportés dans le tableau 3.4 pour le jeu de données *Cultures* et dans le tableau 3.5 pour le jeu de données *Coupes franches*. Deux éléments peuvent être mis en évidence à partir de ces résultats.

Premièrement, à l'instar des contraintes sélectionnées aléatoirement, les méthodes ne bénéficient pas nécessairement des informations sur les contraintes même lorsqu'elles sont sélectionnées à partir d'une méthode active. COP-KMEANS semble particulièrement sujet à ce problème car il ne bénéficie globalement pas ou peu des contraintes. SAMARAH est à nouveau la méthode à qui profite le plus les contraintes, même si la différence n'est toutefois toujours pas importante.

Deuxièmement, les résultats des retours des utilisateurs indiquent que cette stratégie semble la plus performante, surtout si nous ne considérons pas COP-KMEANS qui fonctionne

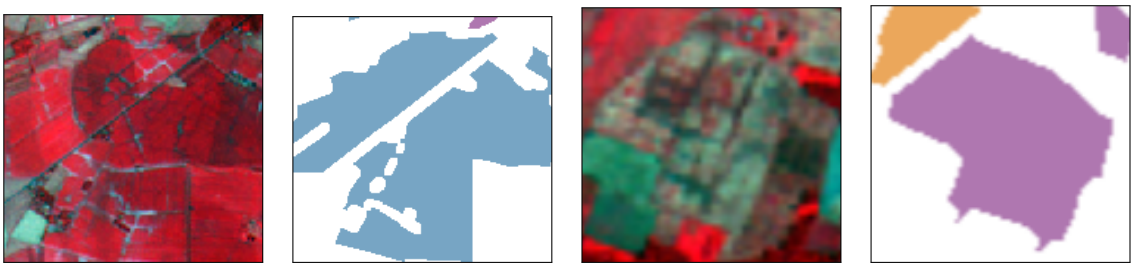
| | *Cultures* (ARI) | | |
|---|---|---|---|
| Method | Retours utilisateurs | NPU | MinMax |
| COP-KMEANS | 0.424 | **0.427** | 0.317 |
| MIP-KMEANS | **0.533** | 0.521 | 0.389 |
| CPClustering | **0.649** | 0.506 | 0.483 |
| SAMARAH | **0.715** | 0.587 | 0.431 |

TABLEAU 3.4 – Niveau 3 : ARI moyen des clustering sous contraintes avec différents stratégies de sélection de contraintes sur le jeu de données *Cultures*. La meilleure performance pour chaque méthode est indiquée en gras.

| | *Coupes franches* (F1-score) | | | |
|---|---|---|---|---|
| Method | Retours utilisateurs - Zone 1 | Retours utilisateurs - Zone 2 | NPU | MinMax |
| COP-KMEANS | 0.096 | **0.105** | 0.084 | 0.045 |
| MIP-KMEANS | **0.133** | 0.129 | 0.076 | 0.076 |
| CPClustering | **0.138** | 0.097 | 0.108 | 0.082 |
| SAMARAH | **0.151** | 0.126 | 0.119 | 0.081 |

TABLEAU 3.5 – Niveau 3 : F1-score moyen des clustering sous contraintes avec différents stratégies de sélection de contraintes sur le jeu de données *Coupes franches*. La meilleure performance pour chaque méthode est indiquée en gras.

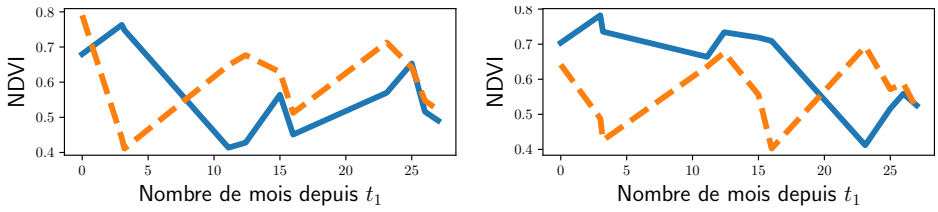
mal avec toutes les stratégies de contraintes. Mais plus surprenant, NPU et MinMax donnent de mauvais résultats, en particulier MinMax qui abouti même à une détérioration des résultats comparé au niveau 1 sans contraintes. Cela peut s'expliquer par le fonctionnement de ces deux méthodes, car elles reposent sur un mécanisme d'exploration qui tente de trouver l'instance la plus informative. Pour déterminer le caractère informatif, elles utilisent un critère qui favorise les instances les plus incertaines par rapport aux instances déjà interrogées. Mais les instances incertaines sont celles qui sont éloignées les unes des autres. Ainsi, ces méthodes ont tendance à se concentrer sur les clusters qui ont une forte inertie globale. Pour les coupes d'arbres qui ne représentent qu'une petite fraction de l'ensemble de données, il est peu probable que ces méthodes atteignent la granularité nécessaire pour le petit cluster de coupes. Même avec la modification ajoutée aux méthodes, ils n'arrivent jamais à sélectionner de bons contre-exemples et aboutissent sur des contre-exemples "évidents" (qui n'apportent aucune information). Pour le jeu de données *Cultures*, ces méthodes ont tendance à se concentrer sur les parties bruitées de l'ensemble de données (par exemple, une route ou une haie traversant le champ dans la figure 3.8).



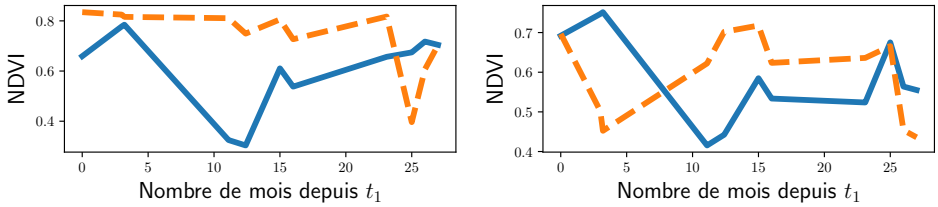
(a) Sous-image 1 (b) Vérité terrain 1 (c) Sous-image 2 (d) Vérité terrain 2

FIGURE 3.8 – Deux extractions d'une des images du jeu de données *Cultures* illustrant la présence de bruit dans la vérité terrain

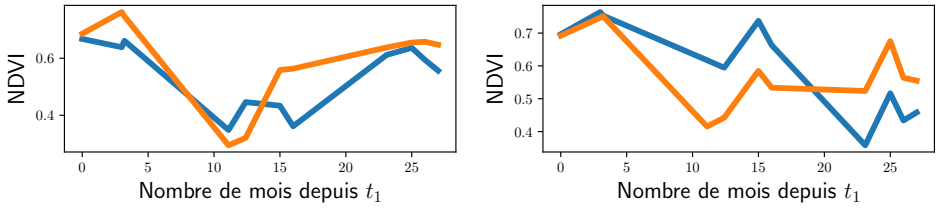
La figure 3.9 présente six exemples de contraintes. Dans cette figure, les lignes en pointillé correspondent à des pixels "non-coupes" alors que les lignes continues correspondent à des pixels de coupes. Nous observons pour tous les pixels une chute brutale du NDVI, ce qui




(a) Exemples de CL entre un pixel de prairie (ligne pointillée) vs coupe d'arbres (ligne continue)




(b) Exemples de CL entre un pixel de champ (ligne pointillée) vs coupe d'arbres (ligne continue)




(c) Exemples de ML entre deux pixels de coupes d'arbres

FIGURE 3.9 – Évolution du signal d'NDVI sur des pixels inclus dans une même contrainte Cannot-Link ou Must-Link, les lignes pleines indiquent des pixels de coupes d'arbre, les lignes en pointillé de pixels de non-coupe d'arbre. L'abscisse indique le nombre de mois écoulés à partir de la capture première image en $t_1$.

correspond à une dévégétalisation. Mais contrairement aux pixels de coupe, où la remonté de cet indice est relativement lente, le NDVI augmente rapidement par la suite ou diminue plusieurs fois pour les autres. Cette évolution est typique des prairies ou des parcelles agricoles qui repoussent plus vite que la forêt et sont souvent récoltées plusieurs fois sur une période de deux ans. Ces exemples de contraintes CL illustrent que l'analyse d'un résultat existant permet à l'expert d'identifier facilement les "faux-positifs".

Les retours utilisateurs ont permis d'obtenir les meilleurs résultats. Cependant, dans cette expérience, nous avons obtenu le clustering suivant avec le nouveau jeu de contraintes, mais en repartant de zéro (sans prendre en compte le clustering précédent). Dans le niveau suivant nous allons nous intéresser à l'aspect incrémentale de notre méthode.

### 3.3.5 Contraintes sélectionnées aléatoirement et méthode incrémentale (niveau 4)

Pour le niveau 4, nous voulons mesurer le bénéfice d'utiliser l'incrémentalité prise séparément. Nous avons donc utilisé le processus incrémental mais sans les retours utilisateurs donnés sur le résultat de clustering précédent afin d'évaluer le gain de chaque élément séparément. Pour cela, nous avons sélectionné 6 contraintes aléatoirement, de manière similaire au niveau 2, que nous avons fournies à I-Samarah sur un total de 4 incrémentations (étape 2 - figure 2.5). De plus, pour chaque état initial, nous avons appliqué à la fois une approche cumulative, où à chaque incrémentation les contraintes précédentes sont conservées, et une approche non cumulative, où à chaque incrémentation les contraintes précédentes sont supprimées. On peut noter que le premier jeu de contraintes est identique dans les deux scénarios, de sorte que les résultats ne diffèrent qu'à partir de la deuxième incrémentation. Les résultats sont rapportés dans le tableau 3.6.

| | *Cultures* (ARI) | | | *Coupes franches* (F1-score) | | |
|---|---|---|---|---|---|---|
| Type | Initial | Dernière inc. | Meilleur inc. | Initial | Dernière inc. | Meilleur inc. |
| Cumulative | 0.463 | 0.711 | 0.721 | 0.036 | 0.109 | 0.113 |
| Non-Cumulative | 0.463 | 0.689 | 0.713 | 0.036 | 0.110 | 0.116 |

Tableau 3.6 – Niveau 4 : Performance moyenne d'I-Samarah avec différents fractions de contraintes sélectionnées aléatoirement à différentes étapes du processus. (inc. : incrémentation).

Les résultats montrent que les performances ne diffèrent pas vraiment de Samarah avec des contraintes aléatoires. Le résultat final, après 4 incrémentations, est rapporté dans la colonne "Dernière inc.". Nous avons également reporté le meilleur résultat de toutes les incrémentations dans la colonne "Meilleur inc.". Ces deux colonnes diffèrent, ce qui montre que les performances peuvent diminuer d'une incrémentation à une autre. Dans le détail, cette diminution se produit dans la moitié des incrémentations. Cet effet est encore plus accentué pour la stratégie non cumulative pour le jeu de données *Cultures*. Les contraintes cumulées semblent régulariser le processus de formation des clusters : les contraintes semblent aider à maintenir le gain obtenu dans l'incrémentation précédente sans toutefois bloquer un potentiel nouveau gain. Cela conduit à des performances comparables à la meilleure fraction de contraintes aléatoires (15%) avec Samarah, mais avec un nombre de contraintes beaucoup plus petit. Cependant, pour le jeu de données *Coupes franches*, le bénéfice semble plus négligeable.

### 3.3.6 Retours utilisateurs et méthode incrémentale (niveau 5)

Pour ce dernier niveau, nous combinons les retours des utilisateurs avec l'incrémentalité. Pour le jeu de données *Cultures*, 10 exécutions d'I-Samarah sont lancées avec des retours d'utilisateurs simulés et pour *Coupes franches*, 5 exécutions pour chaque zone sont lancées

avec des retours réels d'experts. De même que pour le niveau 3, les contraintes simulées sont sélectionnées en prenant 6 contraintes non satisfaites parmi un des ensembles de contraintes de la fraction 50% générée dans l'expérience du niveau 2. Au final, nous disposons d'un total de 24 contraintes pour *Cultures*. Pour *Coupes franches*, ce nombre est déterminé par le choix de l'expert avec une restriction fixée à un maximum de 6 contraintes par incrémentation. De la même manière que pour l'expérience du niveau 4, nous avons demandé aux experts de suivre des stratégies cumulatives et non cumulatives. Les résultats sont reportés dans les tableaux 3.7 et 3.8.

| Type | Initial | #inc. | #contraintes | Dernière inc. | Une inc. en plus |
|---|---|---|---|---|---|
| Cumulative - Zone 1 | 0.036 | 2.6 | 13.4 | 0.411 | 0.301 |
| Non-Cumulative - Zone 1 | 0.036 | 4.2 | 26.4 | 0.262 | 0.194 |
| Cumulative - Zone 2 | 0.035 | 3.6 | 15.6 | 0.202 | 0.184 |
| Non-Cumulative - Zone 2 | 0.035 | 4.8 | 28.8 | 0.122 | 0.115 |

TABLEAU 3.7 – Niveau 5 : Évolution moyenne du F1-score sur 5 exécutions d'I-SAMARAH sur le jeu de donnée *Coupes franches* (inc. : incrémentation).

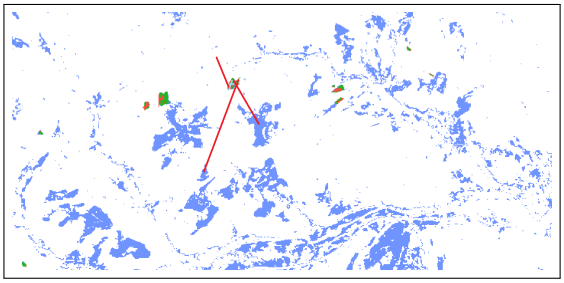
| Type | Initial | Dernière inc. | Meilleur inc. |
|---|---|---|---|
| Cumulative | 0.463 | 0.773 | 0.785 |
| Non-Cumulative | 0.463 | 0.751 | 0.762 |

TABLEAU 3.8 – Niveau 5 : Évolution moyenne de l'ARI sur 10 exécutions d'I-SAMARAH sur le jeu de donnée *Cultures* (inc. : incrémentation).

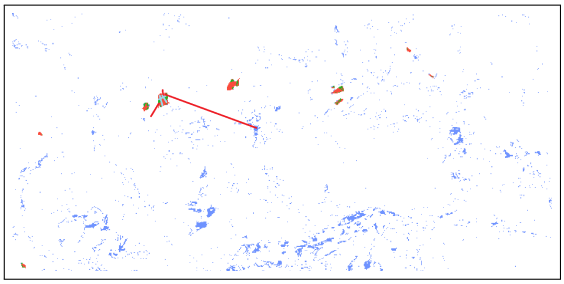
Les résultats montrent globalement une nette augmentation lorsqu'I-SAMARAH est utilisée avec les retours utilisateurs et ce de manière significative. C'est même le cas pour le jeu de données *Cultures*, où l'expertise n'est que simulée.

On peut également constater de meilleures performances pour la stratégie cumulative comparée à la non-cumulative, confirmant la première observation de l'expérience du niveau 4. Cette fois, cela peut également être observé sur le jeu de données *Coupes franches*, car nous notons une différence de moyenne de F1-score significative avec 0,15 pour la zone 1 et de 0,08 pour la zone 2, soit une augmentation de plus de 50%. Pour les coupes d'arbres, la stratégie non-cumulative nécessite également plus d'investissements de l'expert, avec plus de contraintes et plus d'incrémentations, donc plus de résultats à analyser. Il semble donc bien que dans la stratégie cumulative, les contraintes précédentes régularisent le processus d'apprentissage. A l'opposé, la stratégie non-cumulative semble moins stable et plus longue (plus d'incrémentations). Pour la stratégie non-cumulative, certaines informations fournies lors des premières incrémentations semblent être perdues lors du processus incrémental. Bien entendu, les stratégies qui permettent de ne supprimer qu'une partie des contraintes précédentes, ou de revenir en arrière, peuvent être plus pertinentes, car elles sont plus flexibles. Nous n'avons pas malheureusement pu, par manque de temps, valider cette hypothèse. Mais on peut tout de même en déduire que conserver les contraintes précédentes semble être déjà une bonne stratégie par défaut.

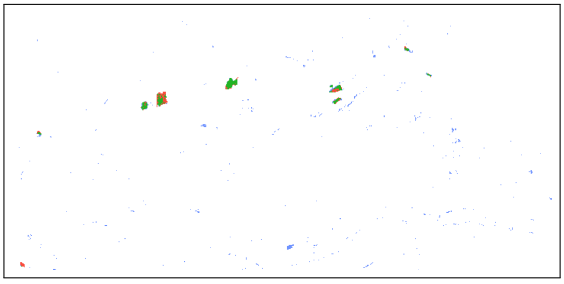
Dans le tableau 3.7, nous avons également retranscrit la progression du F1-score après chaque incrémentation supplémentaire. Pour cela nous avons demandé à l'expert d'arrêter le processus quand il était satisfait ou qu'il estimait ne plus pouvoir améliorer le résultat (reporté dans la colonne "Dernière inc."). Nous lui avons alors demandé de faire une dernière incrémentation pour tenter d'améliorer le résultat (reporté dans la colonne "Une inc. en



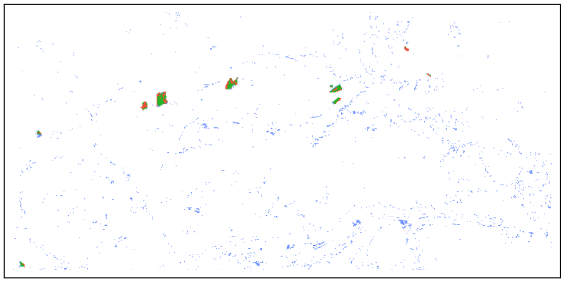
(a) Résultat initial avec l'ensemble de contraintes sur la zone 1



(b) Résultat initial avec l'ensemble de contraintes sur la zone 2



(c) Résultat avec la stratégie cumulative sur la zone 1

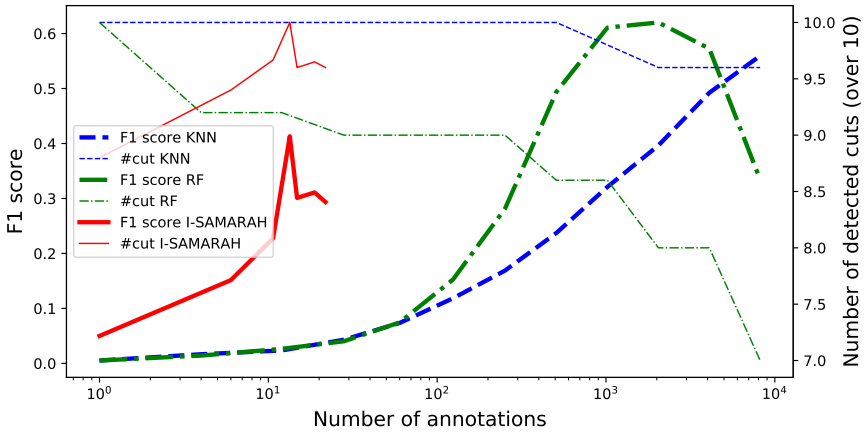


(d) Résultat avec la stratégie cumulative sur la zone 2

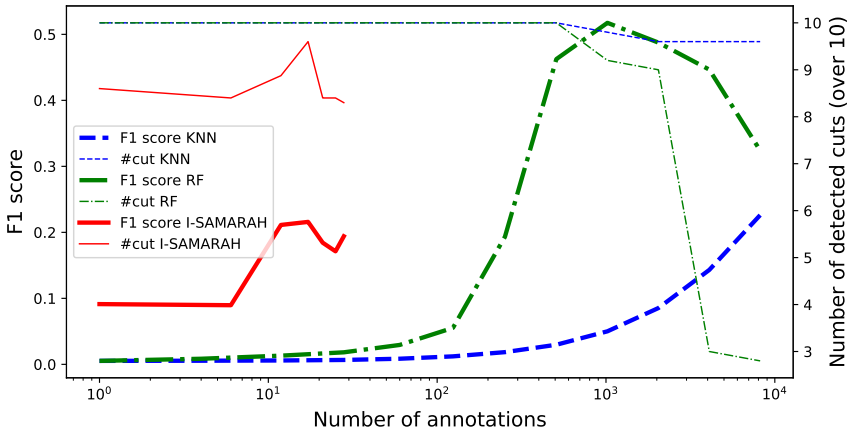
FIGURE 3.10 – Résultats d'I-SAMARAH avec une stratégie cumulative, en vert les Vrais Positifs, en rouge les Faux Négatifs et en bleu les Faux Positifs. Dans (a) et (b) les lignes rouges représentes des contraintes Cannot-Link et les cyan des contraintes Must-Link. Seul les contraintes de la première incrémentation sont affichées. (c) et (d) représentent le résultat final après plusieurs incrémentations de respectivement (a) et (b).

plus"). On peut observer que le résultat ne s'améliore pas nécessairement lorsque plus d'incrémentations sont effectuées, confirmant sur des contraintes utilisateur réelles, les résultats observés sur *Cultures* dans l'expérience du niveau 4. Il en résulte même en une diminution dans notre cas, malgré une augmentation de plus de 20% du nombre de contraintes, passant d'une moyenne de 14,5 et 27,6 à partir de la dernière incrémentation, pour respectivement le cumulatif et non-cumulatif, à 17,9 et 33,4 après l'incrémentation supplémentaire. Habituellement, cela est lié à une cohérence moindre entre les contraintes. En effet le respect des "dernières" contraintes implique que la méthode regroupe des objets trop dissemblables ou prenne en compte des valeurs aberrantes. Cela provient du fait que pour cette dernière incrémentation, les experts ont essayé de supprimer le bruit restant ou d'inclure tous les pixels de la zone sélectionnée. Il s'agissait alors des pixels "mal-classés" restants qui sont ceux qui posent le plus de problèmes pour l'algorithme. En termes de nombre de coupes détectées, nous avons en moyenne de 9,8 coupes sur 10 détectées pour la stratégie cumulative (une seule exécution sur la zone 2 a manqué deux coupes) et 9,4 pour la stratégie non-cumulative. Les deux approches donnent sur ce point des résultats assez proches. La différence de F1-score est surtout expliquée par davantage de faux positifs pour le cluster de coupes et/ou des coupes d'arbre qui ne sont que partiellement détectées. Des exemples de résultats obtenus avec la stratégie cumulative sont affichés dans la figure 3.10.

Pour le niveau 5, nous souhaitons également évaluer la stabilité du clustering lorsque l'expert se concentre sur une partie spécifique des données. Comme nous ne contrôlons pas les contraintes utilisées pour le jeu de données *Cultures*, nous n'avons évalué que les résultats sur le jeu de données *Coupes franches*. Nous avons mesuré la stabilité des clusters qui ne sont pas impliqués dans une coupe d'arbre au cours du processus d'I-SAMARAH par rapport à celle mesurée si on exécute SAMARAH à partir de zéro avec les nouvelles contraintes. Pour cela, nous avons mesuré l'indice Rand (RI) [RAND, 1971] entre le clustering initial et le clustering après le processus incrémental. Nous obtenons un score RI moyen de 0,90, 1.0 étant une correspondance parfaite entre les deux distributions, et 0.0 une totale dissemblance. Nous




(a) Évolution du F1-score et nombre de coupes sur la zone 1




(b) Évolution du F1-score et nombre de coupes sur la zone 2

FIGURE 3.11 – Évolution moyenne du F1-score (gras) et nombre de coupes détectées (normal) comparée entre I-SAMARAH, Random Forest et KNN quand le nombre d'annotations augmente pour le jeu de données *Coupes franches*. (a) correspond au scénario où la zone 1 est prise en référence et (b) où c'est la zone 2. Il est à noter que l'axe des abscisses est logarithmique.

comparons ensuite le résultat de SAMARAH non incrémentale avec les mêmes contraintes mais données dès l'exécution initiale. Dans ce cas, nous obtenons un score RI moyen de 0,86. La stabilité est encore plus forte si nous comparons la différence de score RI entre chaque incrémentation. Dans ce cas, nous obtenons un score RI moyen de 0,93. Cette stabilité permet à l'utilisateur de facilement repérer les changements entre chaque incrémentation et semble limiter les effets de bord indésirables sur les autres clusters.

### 3.3.7 I-Samarah et les méthodes supervisées

Pour notre étude, nous avons souhaité ajouter une dernière évaluation. Il ne s'agit pas de comparer I-SAMARAH aux méthodes supervisées uniquement en termes de qualité, car les méthodes supervisées surpassent généralement les méthodes semi-supervisées dés lors que les exemples sont de qualité et en nombre suffisant. Notre volonté est d'évaluer l'évolution de la qualité compte tenu du nombre d'annotations (contraintes vs étiquettes) fournies aux méthodes. En effet, l'expert souhaite souvent pouvoir évaluer si le temps passé sur les annotations vaut le gain potentiel en qualité du résultat.

Les figures 3.11 et 3.12 comparent les performances des méthodes supervisées Random Forest (RF) et KNN à celles d'I-SAMARAH lorsque le nombre d'annotations augmente. Les

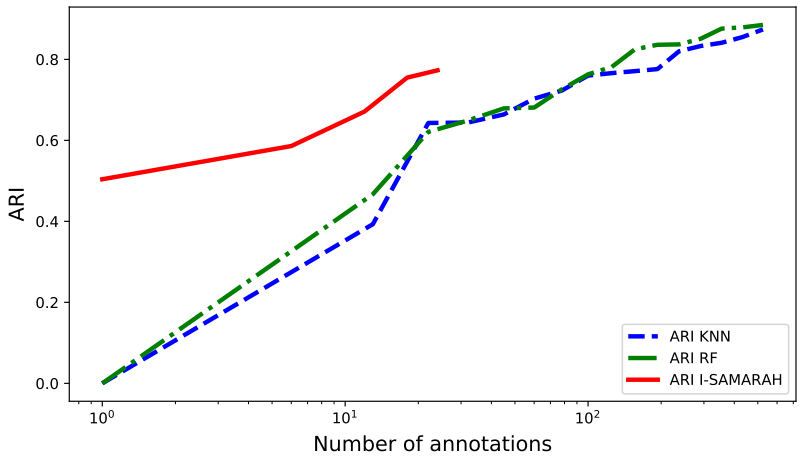



FIGURE 3.12 – Évolution moyenne de l'ARI comparée entre I-SAMARAH, Random Forest et KNN quand le nombre d'annotations augmente pour le jeu de données *Cultures*.

résultats rapportés sont la moyenne sur 10 essais. Pour I-SAMARAH, une contrainte donnée par l'utilisateur (ML ou CL) correspond à une annotation. Pour les méthodes supervisés, une annotation correspond à un pixel étiquetés comme "coupe" ou "non-coupe". Pour les méthodes supervisées sur le jeu de données *Coupes franches*, nous considérons l'ensemble des pixels de la coupe étudiée (zone 1 ou zone 2) comme étant déjà étiquetés comme "coupe" (ils ne sont pas inclus dans le nombre total d'annotations).

Pour le jeu de données *Cultures*, nous pouvons observer que les méthodes RF et KNN nécessitent environ 5 à 6 fois plus d'annotations pour obtenir un résultat similaire (24 pour I-SAMARAH, 156 pour KNN et 125 pour RF).

Pour *Coupes franches*, les graphiques montrent qu'il faut au moins 10 fois plus d'annotations (environ 250) pour obtenir un F1-score équivalent avec RF et 100 fois avec KNN (environ 2000). Cependant, les résultats de RF doivent être analysés plus en détail. On peut observer dans la figure 3.11 que le classificateur RF sur-apprend lorsque trop d'annotations sont fournis. De plus, si l'on évalue le résultat sur le nombre de coupes détectées par RF, ce sur-apprentissage se produit très tôt dans la phase d'entraînement. C'est particulièrement le cas lorsque la zone 1 est choisie comme exemple positif, où RF ignore systématiquement au moins une coupe lorsque plus de 12 annotations sont ajoutées. Le nombre de coupes détectées descend à une moyenne de 8,5 lorsque plus de 250 annotations sont ajoutées. Et si nous regardons des résultats de classification sur l'image, présentés dans l'annexe A.1 dans la figure A.1, nous pouvons observer que pour la zone 1 les coupes d'arbres sont soit complètement détectées, soit totalement ignorées, en particulier la coupe de la zone 2. Pour la zone 2, le rappel est très élevé, mais le résultat est fortement bruité. Pour avoir un résultat comparable en termes de densité de bruit, au moins 500 annotations doivent être fournies au classificateur RF, soit 20 fois plus d'annotations que pour I-SAMARAH.

Pour résumer, dans le meilleur scénario, les méthodes supervisées nécessiteraient environ 125 et 250 annotations, respectivement pour *Cultures* et *Coupes franches*, pour obtenir un résultat équivalent à celui d'I-SAMARAH. Ce nombre d'annotations est réaliste mais nécessite un investissement de temps important. Il convient également de noter que, contrairement aux méthodes supervisées, les méthodes de clustering sous contraintes ne nécessitent pas de définir un ensemble de classes avant le processus. De plus, pour les coupes d'arbre, il est nécessaire de passer à près de 2000 d'annotations pour assurer de faire mieux qu'I-SAMARAH dans tous les cas.

### 3.3.8 Temps d'exécution

Le temps d'exécution de l'algorithme est également un aspect à prendre en compte, notamment dans une configuration itérative. Les mesures sont rapportées dans le tableau 3.9 comme la moyenne sur 10 exécutions sur un processeur Intel® Core™ i5-4670 CPU @ 3.40GHz avec 32GB RAM. Pour les méthodes contraintes, nous avons utilisé les ensembles de contraintes aléatoires avec une fraction à 5% . Pour I-Samarah, nous avons retranscrit le temps de calcul moyen pris par les incrémentations avec retours de l'utilisateur à l'exception de la première (celle sans contrainte). Pour KNN et Random Forest, nous avons utilisé une moyenne globale sur l'ensemble de toutes les exécutions, car le nombre d'annotations n'a pas un fort impact sur les temps d'exécution.

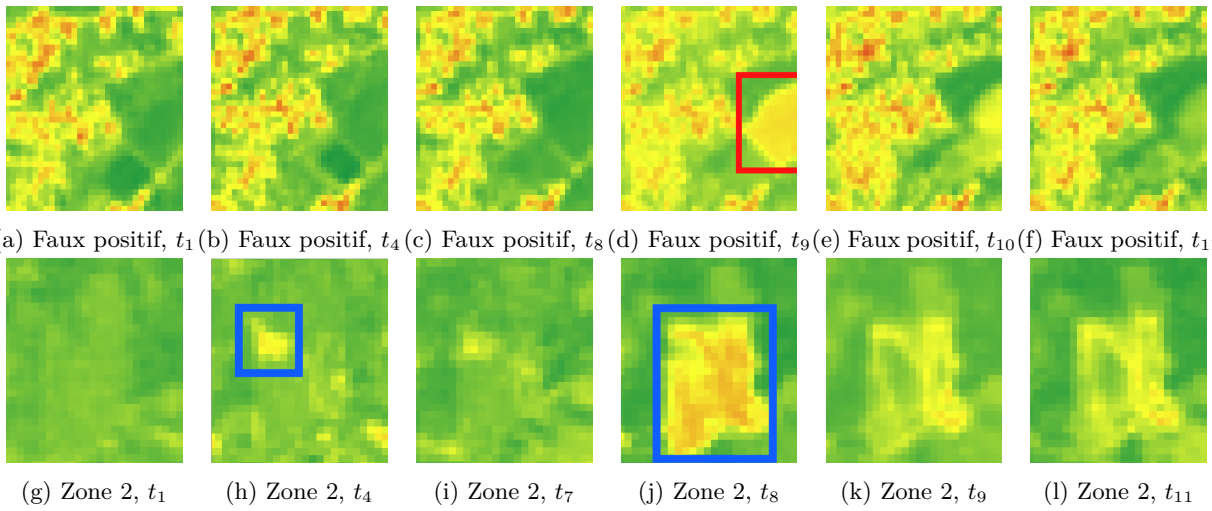
| | Crops | | | Tree clear-cuts | | |
|---|---|---|---|---|---|---|
| Method | Unconstr. | Constr. | Supervised | Unconstr. | Constr. | Supervised |
| Cop-Kmeans | 23.5 | 19.5 | _ | 2155.2 | 2326.4 | _ |
| Mip-Kmeans | 56.3 | 95.2 | _ | 1845.2 | 1690.2 | _ |
| CPClustering* | 732.2 | 321.5 | _ | 4910.5 | 889.4 | _ |
| Samarah | 229.5 | 181.2 | _ | 1987.8 | 1621.9 | _ |
| I-Samarah | 229.5 | 132.1 | _ | 1987.8 | 724.5 | _ |
| RandomForest | _ | _ | 5.1 | _ | _ | 9.1 |
| KNN* | _ | _ | 2.1 | _ | _ | 11.1 |

Tableau 3.9 – Temps d'exécution moyen (en secondes). * indique que le temps d'exécution ne comprend pas le calcul de la matrice de dissimilarité.

À partir des résultats du tableau 3.9, nous pouvons tout d'abord remarquer que le temps d'exécution pour le jeu de données *Cultures* est très faible alors qu'il est très élevé pour *Coupes franches*.

Pour Cop-Kmeans, Mip-Kmeans et Samarah, la différence s'explique principalement par l'utilisation intensive de DTW (pour l'affectation au cluter) et DBA (pour le calcul des centres) qui sont bien connus pour consommer plus de temps CPU que la distance euclidienne et la moyenne arithmétique. Cependant, pour CPClustering, la différence n'est expliquée que par la taille du jeu de données qui a un fort impact sur le temps de calcul, car cet algorithme vise à trouver la solution optimale. De plus, CPClustering et KNN nécessitent le calcul au préalable d'une matrice de distance. Ceci est négligeable pour la distance euclidienne, mais pour DTW sur les coupes claires de l'arbre, le calcul a nécessité environ deux heures. De plus, nous avons dû effectuer ce calcul sur une machine plus adaptée (avec 38 cœurs Intel Xeon E7-8891 avec 250 Go de RAM) car il nécessite un grand espace mémoire.

Deuxièmement, le tableau montre que les contraintes peuvent avoir un impact sur le temps d'exécution. Pour Cop-Kmeans et Mip-Kmeans, cet impact est faible et ne peut être distingué du bruit statistique. Cependant, pour CPClustering et Samarah, les contraintes semblent aider à réduire le temps d'exécution. Pour CPClustering, les contraintes sont conçues pour réduire l'espace de recherche, ce qui explique ce gain. Pour Samarah, le gain est plus élevé, il s'explique également par le fait que des contraintes réduisent l'espace de recherche disponible en éliminant l'exploration des solutions générant un conflit trop fort avec les contraintes. L'impact est encore plus fort lorsque l'incrémentalité est utilisée pour I-Samarah. Dans ce cas, la forte similitude déjà contenue dans le clustering précédent élimine une grande partie des conflits. Ceci est particulièrement le cas pour le jeu de données *Coupes franches*, du fait que les contraintes se concentrent uniquement sur de petites portions du clustering, le reste étant moins impacté. Il en résulte une durée d'exécution moyenne d'environ 12 minutes pour cet ensemble de données après le calcul du clustering initial, ce qui est raisonnable pour un utilisateur.



(a) Faux positif, $t_1$ (b) Faux positif, $t_4$ (c) Faux positif, $t_8$ (d) Faux positif, $t_9$ (e) Faux positif, $t_{10}$ (f) Faux positif, $t_{11}$

(g) Zone 2, $t_1$ (h) Zone 2, $t_4$ (i) Zone 2, $t_7$ (j) Zone 2, $t_8$ (k) Zone 2, $t_9$ (l) Zone 2, $t_{11}$

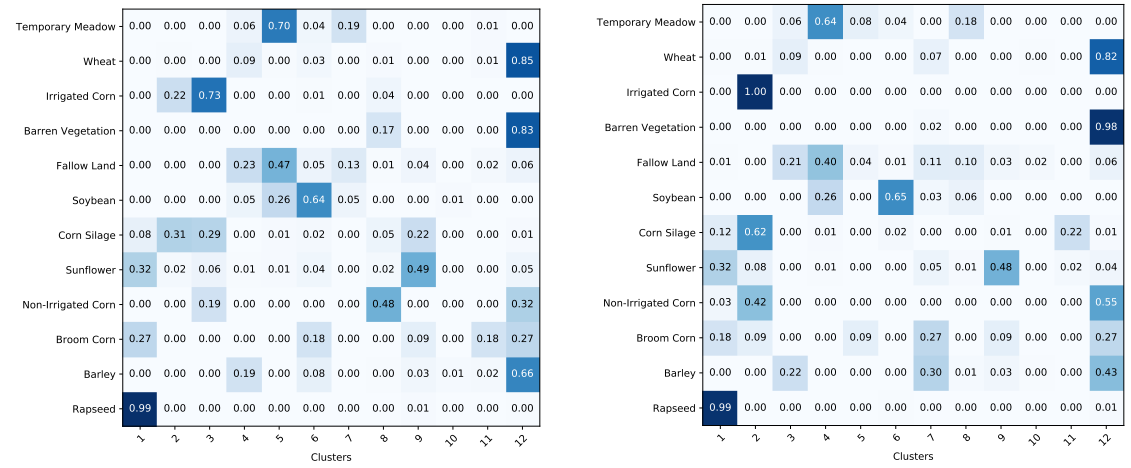
FIGURE 3.13 – Zoom sur un faux positif, correspondant à l'encadré rouge dans (a), comparer avec la coupes d'arbre de la zone 2 sur le jeu de données *Coupes franches*, correspondant aux encadrés bleu dans (h) et (j). Le faux positif est une fauche de prairie qui donne un évolution du NDVI similaire à celle des coupes franches d'arbres. Plus la couleur tend vers le vert, plus le signal d'NDVI est fort.

### 3.3.9 Limitation de l'apport des contraintes

Nous avons vu dans l'expérience du niveau 5 (section 3.3.6) que l'ajout de l'incrémentalité et des retours utilisateur permet d'obtenir un gain significatif des performances du clustering. Cela laisse à penser que notre effort pour augmenter l'informativité des contraintes est pertinent. Toutefois nous avons aussi relevé qu'au bout d'un certain nombre d'incrémentations l'apport des contraintes engendre une dégradation des résultats, ou a minima une instabilité. Nous avons donc voulu revenir plus en détail sur le comportement du clustering dans ce cas là.

Pour le cas du jeu de données *Coupes franches*, nous avons remarqué que les "faux positifs" étaient largement dus à des fauches ayant lieu dans des champs ou des prairies. Un exemple est illustré dans la figure 3.13 où une fauche de prairie est comparée à une coupe d'arbre. On peut constater que l'évolution diffère très peu, la différence se faisant surtout sur l'amplitude du phénomène, la prairie ayant un plus fort signal NDVI au départ et une revégétalisation plus rapide. Toutefois sur certains cas où peu de coupes ont eu lieu, la différenciation ne peut alors plus se faire. Dans ce cas précis, on peut donc constater que les données elles mêmes ne permettent pas d'exclure facilement ce faux positif sans trop perturber le résultat. Il faudrait pour cela utiliser la totalité des bandes optiques pour permettre la différenciation, mais nos tests ont montré que l'extraction des coupes redevient très difficile.

Pour le jeu de données *Cultures*, nous avons reporté les matrices de confusion de l'évolution des résultat d'I-SAMARAH après 1 puis 4 incrémentations dans la figure 3.14. On peut constater une forte augmentation du rappel sur le maïs irrigué (Irrigated Corn), qui est en fait la classe avec la plus grande cardinalité, et la végétation aride (Barren Vegetation). On remarque toutefois que ce gain est largement dû à un sous-ensemble de clusters plus gros, qui englobent plus de classes sans pour autant augmenter nettement la distinction entres les classes prises individuellement, notamment pour les petites classes qui ont tendance à se faire absorber par les grandes. Cela résulte en partie du fait que les contraintes ont statistiquement peu de chance de porter sur une petite classe, et donc a contrario permettent une bonne distinction des classes principales. Sur l'ensemble des résultats obtenus nous avons pu faire un constat similaire. Globalement, il semble que les confusions entre certaines classes sont systématiques, avec par exemple l'orge (Barlery) avec le blé (Wheat) ou le maïs irrigué (Irrigated Corn) avec l'ensilage de maïs (Corn Silage). Par contre, on peut on peut constater que des contraintes cannot-link entre ces classes étaient présentes dans la majorité des




(a) Matrice de confusion sur un résultat de clustering de 0.57 d'ARI (après une incrémentation)

(b) Matrice de confusion sur un résultat de clustering de 0.80 d'ARI (après 4 incrémentations)

FIGURE 3.14 – Matrices de confusion entre la vérité terrain et les résultats de clustering d'I-SAMARAH après une incrémentation (a) et le résultat final (b)

exécutions. Néanmoins, elles sont systématiquement ignorées par le processus de clustering sous contraintes, indiquant que les données ne permettent pas à la méthode de disjoindre ces couples de classes.

## 3.4 Discussion et conclusion

Dans ce chapitre, nous avons conduit une évaluation de notre nouvelle méthode, I-SAMARAH, qui prend en charge les retours utilisateur de manière incrémentale. I-SAMARAH offre un outil à l'utilisateur lui permettant de fournir des contraintes afin de guider le clustering dans un processus exploratoire. Cette approche permet à I-SAMARAH de modifier efficacement le clustering existant pour incorporer les connaissances de l'expert. Nos expérimentations démontrent l'efficacité de notre méthode en termes de qualité par rapport aux méthodes de clustering sous contraintes de l'état de l'art, mais aussi en termes d'investissement utilisateur par rapport aux méthodes supervisées. Par conséquent, cette méthode peut être une alternative aux méthodes supervisées, offrant un compromis entre la qualité et le temps d'annotation, ou encore pour aider à la constitution d'un ensemble d'apprentissage.

Nous avons montré à travers nos différents niveaux de test que ce gain en performance est justifié par l'augmentation de la qualité des contraintes fournies, mais également la capacité de notre méthode à prendre en compte à la fois les nouvelles contraintes et le contexte ayant mené à ces contraintes (le résultat de clustering précédent). La combinaison de ces deux leviers fournit une réponse efficace au problème de consistance posé dans le chapitre précédent. Cependant, les expériences ont également montré que même si I-SAMARAH bénéficie davantage des contraintes que des méthodes contraintes comparées, les résultats se détériorent lorsque le nombre d'incrémentations/contraintes est trop élevé. Ce sujet devrait faire l'objet d'une étude plus approfondie, mais il est susceptible d'être lié au problème de la sur-contrainte. En effet, les agents d'apprentissage K-means ont du mal à apprendre correctement la structure des données. Deux approches peuvent contribuer à résoudre ce problème.

Premièrement, l'expert peut bénéficier d'un mécanisme qui l'aide à sélectionner des contraintes "efficaces". Cela permettrait d'augmenter la probabilité d'un impact positif des nouvelles contraintes, mais aussi de permettre à l'utilisateur d'évaluer a priori si certaines contraintes doivent être conservées d'une incrémentation à l'autre. L'objectif est alors d'augmenter à nouveau la consistance liée aux contraintes.

Deuxièmement, nous avons montré que la limitation des contraintes est en partie liée à la structure de la donnée elle-même, celle-ci ne contenant pas assez d'information pour différencier les données avec les méthodes existantes. Cela pourrait être atténué en utilisant d'autres agents d'apprentissage qui tirent davantage parti des dimensions du temps et de l'espace et ont la capacité d'apprendre une représentation des données plus appropriée. L'objectif est alors de jouer sur la cohérence des contraintes

Les premiers tests effectués avec les méthodes actives NPU et MinMax ont montré que le problème d'informativité résiduelle des contraintes ne peut être solutionné simplement et nécessite donc des recherches plus approfondies, notamment sur le rôle individuel de chaque contrainte dans le résultat obtenu. Néanmoins, nos travaux ont déjà fortement amélioré cet aspect des contraintes. En conséquence, nous nous sommes orientés vers la problématique de cohérence et plus précisément sur l'apprentissage d'une représentation des données cohérente avec les contraintes. C'est ce que nous allons voir dans le chapitre suivant.

# Chapitre 4

# Apprentissage de représentation contraint

**Sommaire**

## 4.1 Contexte : Un intérêt grandissant pour l'apprentissage profond

### 4.1.1 Motivation

L'objectif initial de ce chapitre est d'apprendre une représentation des données permettant de capturer les principaux motifs temporels dans les données tout en reflétant l'information incluse dans des contraintes données par un utilisateur.

Dans la section 1.1.3, nous avons pu voir qu'il existe une multitude de méthodes de représentation pour les séries temporelles. Cependant, il n'existe quasiment aucuns travaux sur l'intégration des contraintes dans ces méthodes et le peu de travaux existants portent souvent sur l'utilisation de contraintes spécifiques [Mohammad et Nishida, 2009; Yu et collab., 2011a] non généralisables et donc peu à même de retranscrire la connaissance de l'expert. De plus, ces méthodes correspondent souvent à certains types de séries temporelles et ne fournissent donc pas de cadre général d'application. Nous avons également évoqué, dans la section 2.2, l'utilisation d'apprentissage de métriques. Bien que plusieurs travaux ont été effectués pour intégrer les contraintes à ces méthodes, elles restent souvent liées à une métrique spécifique. Cela tend à rendre l'intégration des contraintes dans plusieurs systèmes de clustering (avec ou sans contraintes) compliquée.

Dans le même temps, l'apprentissage de représentation a fait des progrès considérables au travers des méthodes d'apprentissage profond. Les premiers travaux se sont tout d'abord portés sur la vision par ordinateur, avec notamment les modèles de classification supervisée LeNet [LeCun et collab., 1998] et AlexNet [Krizhevsky et collab., 2012]. Même si les réseaux de neurones profonds (DNNs : *Deep Neural Networks*) sont toujours un élément central de recherche dans l'analyse supervisée d'images, des travaux plus récents ont permis d'étendre leur utilisation.

En effet, des progrès significatifs ont été effectués dans le domaine du clustering [Caron et collab., 2018; Ghasedi Dizaji et collab., 2017; Guo et collab., 2017a; Xie et collab., 2016; Yang et collab., 2019]. En plus de leurs hautes performances, ces DNNs fournissent un système complet, ne nécessitant pas d'étapes de pré-traitement supplémentaires, ce qui simplifie la conception et l'intégration de ces méthodes. Dans la suite de cette thèse nous ferons référence à ces méthode comme les méthodes de *clustering profond*.

En parallèle, les DNNs se sont avérés pertinents pour traiter des données temporelles. En effet, des travaux récents de classification supervisée sur les séries temporelles ont démontré qu'ils pouvaient obtenir des résultats compétitifs avec les méthodes de l'état de l'art [Dempster et collab., 2020; Fawaz et collab., 2019].

Partant de ces différents constats et de notre besoin de trouver des solutions au problème de cohérence nous nous sommes donc intéressés à cette famille de méthodes dans le cadre du clustering sous contraintes.

### 4.1.2 Apprentissage profond et clustering

Les méthodes à base d'apprentissage profond consistent à apprendre une représentation des données facilitant la résolution d'une tâche donnée. Cette représentation est obtenue à partir d'un réseau de neurone profond (DNNs : Deep Neural Networks). Dans cette section, nous expliquerons, dans un premier temps, comment le modèle de DNN est construit et entraîné pour la résolution d'une tâche de manière générale, puis, dans un second temps, comment ce processus peut être adapté à la tâche de clustering.

#### Apprentissage d'un réseau de neurone (DNNs)

Les DNNs, sont structurés comme un ensemble de $L$ fonctions paramétriques, appelées couches (*layer* en anglais), qui se succèdent les unes aux autres, chacune calculant une représentation de la couche précédente. Chaque couche contient un ensemble de neurones, chacun

étant une sous-unité permettant de calculer un élément de la représentation de sortie. Ainsi, la couche $l_i$ prend en entrée la sortie de sa couche précédente $l_{i-1}$ et applique une non-linéarité (telle que la fonction sigmoïde, ou tanh) pour calculer sa propre sortie. Le comportement de ces transformations non linéaires est contrôlé par un ensemble de paramètres $\theta_i$ pour chaque couche, aussi appelé *poids*. Un DNN peut donc être écrit comme :

$$f_\Theta(x) = l_L(\theta_L, l_{L-1}(\theta_{L-1}, ...(l_1(\theta_1(x))))) \tag{4.1}$$

Classiquement, les poids d'un DNN sont entraînés via la minimisation d'une fonction de coût (*loss function* en anglais) par un algorithme de descente de gradient. Pour cela, les poids sont initialisés de manière aléatoire. Ensuite, les données sont projetées en appliquant la fonction $f$ dans un nouvel espace de représentation, appelé *espace latent*. Par exemple, en classification supervisée, l'espace latent correspond à un vecteur qui contient le degré d'appartenance à chaque classe présente dans la donnée de référence.

A partir de cette représentation, une erreur est calculée via la fonction de coût $\mathcal{L}$. Cette fonction de coût dépend de la tâche qu'on souhaite résoudre. Par exemple, en classification supervisée, $\mathcal{L}$ correspond généralement au calcul de l'entropie croisée entre la prédiction du réseau (la représentation) et la classe de la donnée de référence. Ensuite, à l'aide de l'algorithme de rétropopagation de gradient [LeCun et collab., 1998], le gradient de l'erreur est calculé pour mettre à jour chaque neurone d'un réseau de neurones, de la dernière couche vers la première. Ainsi, en effectuant itérativement un passage en avant (*forward pass*) suivi d'une rétropropagation (*backpropagation*), les paramètres du réseau sont mis à jour de manière à minimiser la fonction de coût sur les données d'apprentissage.

### Application au clustering

Comme présenté précédemment dans la section 1.1, la tâche de clustering peut se définir comme trouver une partition de $X$, en $K$ clusters $c_1, \ldots, c_k$, qui à la fois maximise la similarité des objets appartenant au même cluster et maximise la dissimilarité entre les objets de clusters différents, avec $X$ un ensemble de $N$ objets :

$$X = \{x_1, ..., x_N\} \tag{4.2}$$

Dans le cas de l'utilisation de méthodes d'apprentissage profond dans le clustering, qu'on appelle clustering profond, la tâche est légèrement modifiée. Elle consiste généralement à apprendre une nouvelle représentation des données, puis à effectuer le clustering sur cette nouvelle représentation. Cette représentation est obtenue par la projection des données avec un réseau de neurone profond, que l'on appelle dans ce cas un *encodeur*.

Pour les encodeurs, la dernière couche, $l_L$, est généralement appelée la couche latente ou d'*embedding*. A noter que cette couche est souvent notée $l_z$. On parle alors de $Z$ comme l'espace latent appris par le DNN, par opposition à l'espace d'origine des données, définit par :

$$Z = \{f_\Theta(x_1), ..., f_\Theta(x_N)\} = \{z_1, ..., z_N\} \tag{4.3}$$

La méthode de clustering profond consiste à trouver une partition de l'ensemble $Z$ en $K$ clusters, par rapport à la mesure de dissimilarité $d_z(z_i, z_j)$. La plupart des méthodes de la littérature utilisent la distance euclidienne pour $d_z$ et Kmeans comme méthode de clustering [Bo et collab., 2020; Guo et collab., 2017a; Jiang et collab., 2016; Ma et collab., 2019; Xie et collab., 2016; Yang et collab., 2019]. Par conséquent, l'objectif est de trouver une application $f_\Theta$ qui permet d'obtenir une partition pertinente $C$ pour les données de séries temporelles. Dans les sections suivantes, $f_\Theta$ sera désigné par $f$.

Cependant, dans un contexte non-supervisé, le choix d'une fonction de coût permettant une bonne prédiction des classes attendues n'est pas simple, car aucune donnée de référence n'est disponible. Il existe de nombreuses approches pour contourner cette absence de références. Nous présenterons une de ces méthodes dans la suite de ce chapitre (section 4.2.1) et de manière plus approfondie dans le chapitre suivant (section 5.1).

## 4.2 Apprentissage profond et contraintes

Dans cette section, nous avons voulu tester avant tout la pertinence de l'utilisation de telles méthodes sur les données de séries temporelles de télédétection. L'incorporation dans notre méthode I-Samarah n'y sera donc pas traitée.

### 4.2.1 État de l'art

#### Les contraintes et le clustering profond

Du fait de l'intérêt récent pour l'intégration de contraintes dans des méthodes de clustering à base de réseaux de neurones, le nombre de travaux traitant de ce sujet reste assez limité. On dénombre plusieurs approches, mais une grande partie d'entre elles traitent de l'intégration de contraintes spécifiques et sont souvent très peu généralisables, avec par exemple Zhu et collab. [2019] qui intègrent des contraintes physiques ou Togami [2019] qui intègre une contrainte sur la spatialité. Cependant, trois méthodes ont été proposées pour inclure des contraintes par paires (must-link ou cannot-link), Semi-supervised Deep Embedded Clustering (SDEC) [Ren et collab., 2019], Deep Triplet-Driven Semi-supervised Embedding Clustering (DeTSEC) [Ienco et Pensa, 2019] et Deep Constrained Clustering (DCC) [Zhang et collab., 2019]. DeTSEC et DCC introduisent également un nouveau type de contraintes appelé *contraintes de triplet*. Une contrainte de triplet entre trois objets $o_i$, $o_j$ et $o_k$ signifie que la représentation latente de l'objet $o_i$ doit avoir une similarité avec celle de l'objet $o_j$ plus élevée qu'avec celle de l'objet $o_k$. A noter que ces contraintes de triplet peuvent être dérivées des ensembles de contraintes must-link et cannot-link [Ienco et Pensa, 2019].

Ces trois articles utilisent les contraintes au niveau de leur fonction de coût, dont l'objectif est donc de maximiser la similarité entre les représentations latentes des objets d'une contrainte must-link et respectivement la minimiser pour une contrainte cannot-link. Pour notre étude préliminaire nous avons utilisé la méthode Deep Constrained Clustering (DCC), du fait qu'elle était la seule dont l'article et le code étaient publiés à l'époque des tests.

#### La méthode Deep Constrained Clustering

La méthode Deep Constrained Clustering (DCC) proposée par Zhang et collab. [2019] est basée sur une méthode de clustering par apprentissage profond, Deep Embedded Clustering (DEC) [Xie et collab., 2016], plus précisément sur son extension Improved DEC (IDEC) [Guo et collab., 2017a]. Dans un premier temps, nous allons présenter rapidement la méthode IDEC, puis, dans un second temps, nous présenterons l'apport de DCC.

**Improved Deep Embedded Clustering :** Deep Embedded Clustering (DEC), repose sur l'utilisation d'un autoencodeur. Un *autoencoder* est constitué d'un encodeur $f$ ($f : X \to Z$) et d'un *décodeur* $g$, $g$ étant une fonction $g : Z \to X$, qui permet de reconstruire la donnée d'origine $x_i$ à partir de son encodage $z_i = f(x_i)$. L'encodeur et le décodeur sont alors entraînés via une *fonction de coût de reconstruction* :

$$L_r = \frac{1}{n}\sum_{i=1}^{n}\|x_i - g(f(x_i))\|^2 \tag{4.4}$$

Une fois l'autoencodeur pré-entraîné, seul l'encodeur est gardé et il lui est adjoint une nouvelle couche $l_c$, appelée couche de clustering. Le nouveau réseau obtenu (l'encodeur suivi par la couche de clustering) est alors entraîné pour obtenir une nouvelle représentation $Q \in \mathbb{R}^K$, avec $K$ le nombre attendu de clusters. Le modèle est entraîné pour que la représentation $Q$ permette la séparation des données en $K$ groupes distincts via une fonction de coût de clustering, $L_{clustering}$. Plus de détails sur la définition de $L_{clustering}$ et le fonctionnement de

DEC sont donnés dans le chapitre suivant section 5.1.3. Le clustering se fait alors sur :

$$Q = \{q_1, ..., q_N\} = \{l_c(f(x_1)), ..., l_c(f(x_N))\} \tag{4.5}$$

La valeur $q_{i,j}$ à chaque index $j$ de $q_i$ peut être vu comme le degré de confiance en l'appartenance de l'objet $x_i$ au cluster $j$.

L'apport de la méthode IDEC est de garder le décodeur et la fonction de coût de reconstruction de l'autoencodeur même après la phase initiale. L'intuition derrière cela est que la fonction de coût de clustering, en modifiant l'espace de représentation, peut altérer la représentativité de l'encodage généré et de ce fait la performance du clustering.

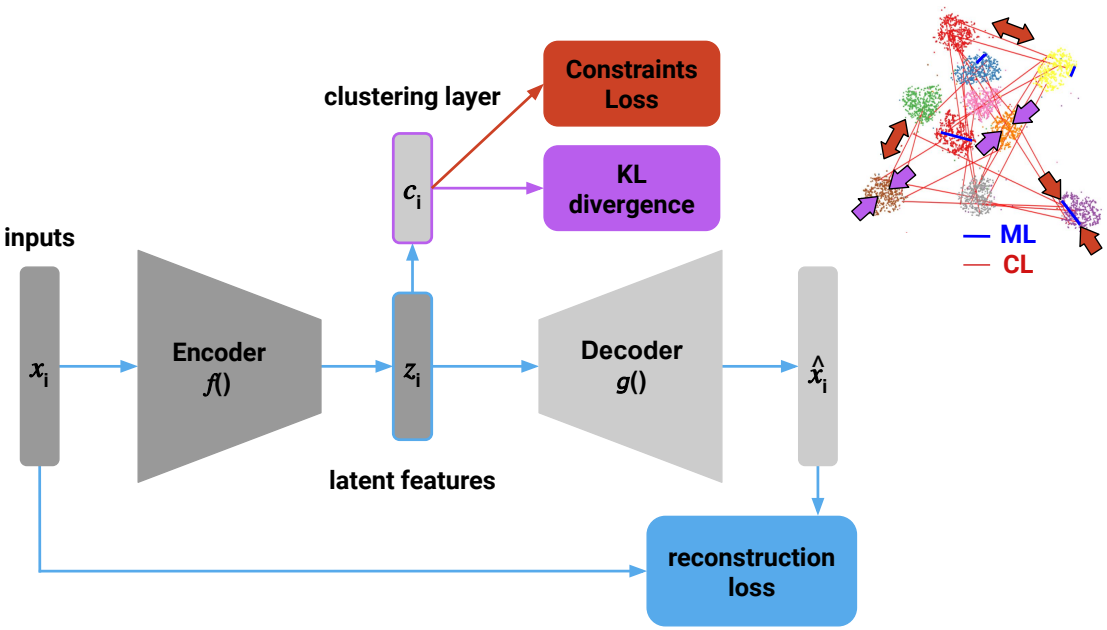



FIGURE 4.1 – La méthode *DCC* : Le modèle est entraîné à la fois pour reconstruire les données, pour obtenir une représentation plus densément distribuée avec la KL divergence (flèches violettes) et pour s'adapter aux contraintes (flèches rouges).

**L'intégration des contraintes :** la méthode DCC supporte quatre types de contraintes :

— les contraintes par paires : les contraintes must-link et cannot-link. La fonction de coût utilisée pour l'ensemble $ML$ des contraintes must-link est définie par :

$$l_{ML} = L_r - \gamma_{ML} * \sum_{(a,b)\in ML} log \sum_j q_{aj} * q_{bj} \tag{4.6}$$

De manière équivalente, la fonction de coût pour l'ensemble $CL$ des contraintes cannot-link est définie par :

$$l_{CL} = - \sum_{(a,b)\in CL} log(1 - \sum_j q_{aj} * q_{bj}) \tag{4.7}$$

De manière intuitive, la fonction de coût pour les ML va favoriser des instances avec les mêmes degrés d'assignation aux différents clusters et celles pour les CL qui ont des degrés d'assignation opposés. La fonction de coût pour les ML est régularisée par l'ajout de la fonction de coût de reconstruction $L_r$ pondérée par un facteur $\gamma_{ML} > 0$, de manière identique à $L_c$, afin d'éviter de tomber dans une solution triviale qui serait d'assigner toutes les instances concernées à un seul cluster.

— les contraintes de triplet : pour un triplet $(a, p, n)$, avec $x_a$ un objet d'ancrage, $x_n$ un exemple négatif et $x_p$ un exemple positif, cela indique que $x_a$ est plus similaire à $x_p$ qu'à

$x_n$. La fonction de coût est définie pour l'ensemble $T$ des contraintes de triplet par :

$$l_T = \sum_{(a,p,n)\in T} \max(d(q_a, q_n) - d(q_a, q_p) + \theta, 0), \tag{4.8}$$

où $d(q_a, q_b) = \sum_j q_{a,j} * q_{b_j}$ et $\theta > 0$ est une variable qui contrôle la distance minimale entre les exemples négatifs et positifs.

— les contraintes d'objets difficiles : la contrainte est associée à une valeur $M_i \in [-1, 1]$, $M_i < 0$ indiquant qu'un objet va être difficile à assigner à un cluster (qu'il peut être assigné à plusieurs clusters) et $M_i > 0$ que l'objet va être facile à assigner (qu'il appartient fortement à un unique cluster), $M = 0$ indiquant l'absence de connaissance sur l'objet. La fonction de coût est alors définie par :

$$l_I = \sum_{t\in\{M_t<0\}} M_t \sum_j q_{tj}^2 - \sum_{s\in\{M_s>0\}} M_s \sum_j q_{sj}^2 \tag{4.9}$$

— la contrainte de taille globale : elle indique que chaque cluster doit avoir approximativement la même taille. La fonction de coût est alors définie pour $k$ clusters par :

$$l_G = \sum_{c\in\{1,..,k\}} (\sum_{i=1}^{n} q_{ic}/n - \frac{1}{k})^2 \tag{4.10}$$

Le modèle est alors entraîné par une fonction de coût globale qui est la somme des fonctions de coût de contraintes, la fonction de coût de clustering et la fonction de coût de reconstruction. Dans notre étude, nous n'avons pris en considération que les contraintes par paires, pour permettre une comparaison avec les méthodes du chapitre précédent.

### 4.2.2 Évaluation en télédétection

L'objectif principal dans cette section est d'évaluer si cette approche de clustering contraint est pertinente sur des séries temporelles dans le contexte de la télédétection. Nous allons donc présenter notre proposition d'adaptation de DCC, puis comment nous avons procédé à l'évaluation avec les autres méthodes de clustering sous contraintes.

#### Adaptation aux séries temporelles

Nous avons testé la version originale de DCC, qui est composée de couches totalement connectées. Pour cette version, l'architecture est identique que celle proposé par les auteurs. Nous proposons également une version modifiée de DCC utilisant des convolutions 1D, ces dernières ayant montré leur efficacité pour les séries temporelles en classification supervisée Fawaz et collab. [2019]; Wang et collab. [2017]. Dans cette nouvelle version, nous gardons la dimension originale des séries temporelles et le réseau est composé uniquement de couches convolutives 1D suivies à chaque fois d'une couche de *batch-normalisation*, une couche de *global average pooling* est placée à la fin juste avant la couche finale d'encodage. Cette dernière reste une couche totalement connectée. Nous distinguons la version originale, DCC, à la version avec convolutions, DCC-conv. Plus de détails sur le fonctionnement des différents types de couches sont donnés dans le chapitre suivant (voir section 5.1.1).

#### Paramètres expérimentaux

Pour cette évaluation nous avons repris les jeux de données (*Cultures* et *Coupes franches*), les méthodes (Cop-Kmeans, Mip-Kmeans, CPClustering et Samarah), ainsi que les résultats du chapitre précédent, mais sans l'aspect incrémental (sections 3.3.2 et 3.3.3). Pour les méthodes de clustering par apprentissage profond, nous avons suivi le paramétrage proposé dans DEC et IDEC. Cependant, comme les résultats n'étaient pas stables (voir section 4.2.2

pour plus de détails) nous y avons apporté des modifications mineures. Pour la dimension de sortie de l'encodeur, nous l'avons fixée à 2 au lieu de 10, car cela a permis d'obtenir plus de stabilité lors de l'apprentissage du modèle, sans toutefois perdre en qualité des résultats. Pour DCC, l'encodeur est fixé aux dimensions $d/500/500/2000/2$, où $d = L * F$, $L$ étant la longueur de la série en entrée et $F$ le nombre d'attributs de la série. Pour DCC-Conv les dimensions sont $l * t/128/256/128/2$, avec respectivement des filtres 1D de dimension de 8, 5 et 3, suivant ainsi les recommandations dans Wang et collab. [2017]. Pour les deux versions, les dimensions des décodeurs sont en miroir de celles de l'encodeur. Pour les deux, la méthode d'optimisation utilisée est l'algorithme du gradient stochastique [Rumelhart et collab., 1986] avec un *momentum* de 0.9 et une valeur de *decay* de $1e-6$, pour compenser la variabilité mentionnée précédemment. Les paramètres $\gamma$ et $\gamma_{ML}$ sont tous deux fixés à 0.1 comme décrit dans les articles d'origine.

Pour la génération des contraintes nous avons repris les ensembles de contraintes générées aléatoirement de taille 5%, 10%, 15% et 50% de la taille jeu de données, ainsi que les contraintes utilisateurs et actives (sections 2.4.1 et 3.2.2).

Le code utilisé pour cet article est disponible en ligne [1].

### Résultat

Les résultats sans contraintes sont présentés dans le tableau 4.1. Les résultats avec contraintes sont présentés dans le tableau 4.2 pour le jeu de données *Cultures* et 4.3 pour le jeu de données *Coupes franches*.

| | *Cultures* | | *Coupes franches* | |
|---|---|---|---|---|
| Méthode | ARI | Sat. | F1 score | Sat. |
| Cop-Kmeans | 0.420 | 0.807 | 0.104 | 0.917 |
| Mip-Kmeans | 0.407 | 0.803 | **0.107** | **0.920** |
| CPClustering | 0.681 | 0.413 | 0.106 | 0.877 |
| Samarah | 0.463 | 0.817 | 0.036 | 0.916 |
| DCC | **0.703** | **0.885** | 0.102 | 0.915 |
| DCC-Conv | 0.508 | 0.833 | 0.106 | 0.917 |

Tableau 4.1 – Performance moyenne sans contrainte et taux moyen de satisfaction par jeu de données. La meilleure performance pour chaque jeu de données est indiquée en gras.

| Méthode | 5% | 10% | 15% | 50% | R.U. | NPU | MinMax |
|---|---|---|---|---|---|---|---|
| Cop-Kmeans | 0.406 | 0.314 | 0.443 | 0.369 | 0.424 | 0.427 | 0.317 |
| Mip-Kmeans | 0.428 | 0.416 | 0.431 | 0.532 | 0.533 | 0.521 | 0.389 |
| CPClustering | 0.650 | 0.562 | 0.542 | 0.510 | 0.649 | 0.506 | .483 |
| Samarah | **0.691** | **0.682** | **0.714** | 0.702 | **0.715** | **0.587** | 0.431 |
| DCC | 0.550 | 0.504 | 0.448 | 0.615 | 0.601 | 0.542 | **0.565** |
| DCC-Conv | 0.497 | 0.501 | 0.491 | **0.820** | 0.510 | 0.498 | 0.502 |

Tableau 4.2 – ARI moyenne des clustering sous contraintes avec différents ensembles de contraintes sur le jeu de données *Cultures*. La meilleure performance pour chaque jeu de données est indiquée en gras. R.U. : retours utilisateurs.

Il ressort des résultats de cette expérimentation que les méthodes basées sur l'apprentissage profond arrivent à obtenir de bons résultats sur certains cas, mais ont beaucoup de

1. https://github.com/blafabregue/DeepConstrainedClustering

| Méthode | 5% | 10% | 15% | 50% | R.U. - Zone 1 | R.U. - Zone 2 | NPU | MinMax |
|---|---|---|---|---|---|---|---|---|
| COP-KMEANS | 0.104 | 0.098 | **0.124** | 0.112 | 0.096 | 0.105 | 0.084 | 0.045 |
| MIP-KMEANS | **0.112** | **0.109** | 0.105 | 0.114 | 0.133 | **0.129** | 0.076 | 0.076 |
| CPClustering | 0.099 | 0.103 | 0.105 | 0.096 | 0.138 | 0.097 | 0.108 | 0.082 |
| SAMARAH | 0.102 | 0.097 | 0.108 | **0.121** | **0.151** | 0.126 | **0.119** | 0.081 |
| DCC | 0.102 | 0.103 | 0.112 | 0.094 | 0.110 | 0.105 | 0.098 | 0.042 |
| DCC-Conv | 0.106 | 0.106 | 0.102 | 0.118 | 0.103 | 0.113 | 0.112 | **0.087** |

TABLEAU 4.3 – F1 score moyen des clustering sous contraintes avec différents ensembles de contraintes sur le jeu de données *Coupes franches*. La meilleure performance pour chaque jeu de données est indiquée en gras. R.U. : retours utilisateurs

mal à gérer les contraintes. Les performances sur le jeu de données *Coupes franches* donnent des résultats très peu significatifs. En effet, les résultats sont bas, mais correspondent à ceux obtenus avec les autres méthodes, ce qui peut s'expliquer par la difficulté de ce jeu de données (extraction d'une unique classe peu représentée). Cependant, un point assez surprenant sur le jeu de données *Cultures* est la différence de comportement entre les deux versions.

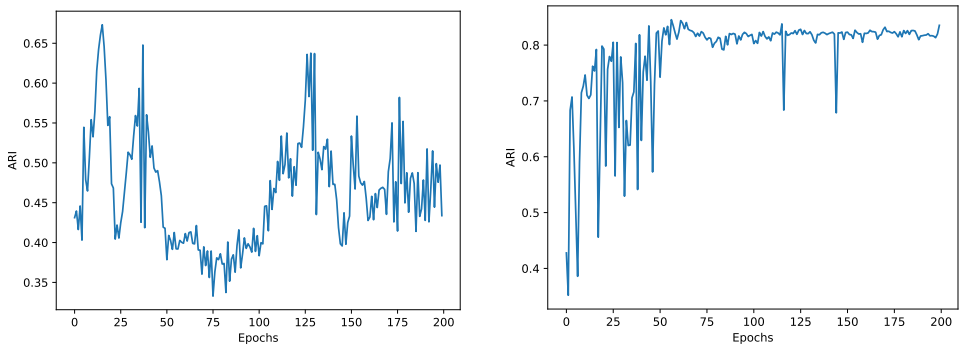
DCC, la version classique sans convolution, obtient les meilleurs résultats sans contraintes pour *Cultures*, mais les contraintes ont un fort effet négatif. Cette observation va dans le sens qu'une forte cohérence induit une corrélation inverse avec la performance de la méthode. Toutefois, ceci est en opposition avec les observations faîtes par ZHANG et collab. [2019], qui concluaient sur un absence de dégradation des résultats. Dans notre cas, cela peut s'expliquer par la forte présence de bruit dans les contraintes (route à travers les champs, pixels en frontière des champs, ...).

DCC-Conv, de son côté, obtient de bons résultats avec les contraintes, mais seulement quand leur nombre est assez élevé. Il est à noter que les contraintes sont utilisées lors de la descente de gradient, elles sont donc utilisées comme simple guidage ce qui permet un relâchement du respect des contraintes. Il semble cependant que cet effet n'est pas assez important lorsque le nombre de contraintes est trop faible, ou trop localisé au niveau des objets contraints. Toutefois, dans le cas de DCC-Conv, nous pouvons légitimement nous demander si le réseau ne commence pas à apprendre partiellement le jeu de données lui-même et non la structure des données, l'algorithme étant entraîné et testé sur le même jeu de données (celui de test).

Deux autres points doivent être abordés, qui ne sont pas directement visibles dans le tableau 4.2. Tout d'abord, pour le jeu de données *Cultures*, l'écart-type augmente fortement entre la version sans contrainte et celle avec contraintes, mais dans le même temps il tend à diminuer quand le nombre de contraintes augmente, que ce soit pour DCC ou DCC-Conv (pour DCC-Conv, l'écart-type est respectivement, sans contrainte, avec 5%, 15%, 50% de 0.005, 0.069, 0.010 et 0.015). L'effet des contraintes peut donc être une forte source de perturbation, cela peut s'expliquer par le bruit dans notre cas, mais quand le nombre de contraintes augmente cela permet d'atténuer cet effet. Le second point est que le réseau n'est pas stable durant l'apprentissage. C'est le cas pour les deux versions, mais d'une manière plus amplifiée pour DCC (pour DCC, l'ARI varie entre 0.45 et 0.74), et ceux malgré les légères modifications apportées dans la section 4.2.2. Une illustration de cette instabilité est reportée dans la figure 4.2. Cela se produit essentiellement quand des contraintes sont ajoutées, mais aussi sans contrainte dans une plus faible amplitude. Dans la section suivante, nous avons donc chercher à voir si ce problème était propre à notre jeu de données ou non.

### 4.2.3 Des limites non restreintes à la télédétection

Afin de s'abstraire des limitations potentielles liées à l'application à la télédétection, nous avons décidé d'évaluer DCC et DCC-Conv sur des jeux de données de domaines variés. Pour




(a) DCC-Conv avec 5% de contraintes (b) DCC-Conv avec 50% de contraintes

FIGURE 4.2 – Évolution de l'ARI au cours du processus d'apprentissage de DCC-Conv sur le jeu de données *Cultures*.

cela nous avons repris la sélection de jeux de données de l'archive UCR utilisés dans la section 2.4.

Nous avons donc repris et complété les résultats présentés dans la section 2.4.2 que nous avons reportés dans le tableau 4.4. Les résultats pour chaque jeu de données et chaque taille d'ensemble de contraintes sont reportés en annexe dans les tableaux A.10 à A.11.

| Méthode | ECG5000 | ElectricDevices | FacesUCR | InsectWingbeatSound | MALLAT | StarLightCurves | TwoPatterns | uWaveGestureLibraryX | UWaveGestureLibraryAll |
|---|---|---|---|---|---|---|---|---|---|
| COP-KMeans | 0.433 | **0.337** | 0.485 | 0.057 | 0.750 | 0.541 | 0.933 | **0.439** | 0.447 |
| Spec | **0.816** | 0.231 | **0.556** | 0.151 | **0.931** | **0.678** | **1.000** | 0.270 | **0.495** |
| CCSR | 0.029 | 0.328 | 0.410 | 0.125 | 0.861 | 0.231 | 0.000 | 0.260 | 0.291 |
| CPClustering | 0.455 | 0.180 | 0.188 | 0.098 | 0.642 | 0.616 | 0.257 | 0.223 | 0.178 |
| SAMARAH | 0.531 | 0.312 | 0.424 | 0.043 | 0.792 | 0.506 | 0.868 | 0.382 | 0.356 |
| DCC | 0.432 | 0.082 | 0.151 | **0.300** | 0.259 | 0.372 | 0.017 | 0.238 | 0.425 |
| DCC-Conv | 0.387 | 0.093 | 0.077 | 0.085 | 0.289 | 0.190 | 0.033 | 0.093 | 0.106 |

TABLEAU 4.4 – ARI moyenne - non contraint. Le meilleur score par jeu de données est mis en gras.

L'analyse des résultats permet de constater un comportement identique entre le jeu de données *Cultures* et les jeux de données provenant de l'archive UCR. DCC-Conv bénéficie fortement de l'ajout de contraintes, particulièrement de la portion à 50%, mais les résultats pris individuellement ne sont pas si bons car DCC-Conv part aussi de résultats assez faibles sans contrainte. De plus, il ressort du tableau 4.6 que le taux de satisfaction des contraintes est assez élevé, surtout lorsqu'il est mis en perspective avec les scores d'ARI, par rapport aux autres méthodes. Ceci laisse supposer une forte capacité à satisfaire les contraintes par cette méthode, sans toutefois que cela ait un impact équivalent sur l'espace latent. Dit autrement, les modifications de l'espace latent semblent ne pas se généraliser, mais restent localisées aux objets contraints. A l'opposé, DCC ne bénéficie pas vraiment des contraintes, les performances passant même régulièrement en dessous de celles de DCC-Conv, alors que les résultats sans contraintes sont nettement meilleurs.

Concernant l'évolution de l'ARI au cours de l'apprentissage, nous avons constaté un comportement similaire dans la majorité des cas, avec par exemple le jeu de données TwoPatterns dans la figure 4.3. De plus, nous avons constaté une forte baisse de l'ARI entre la version

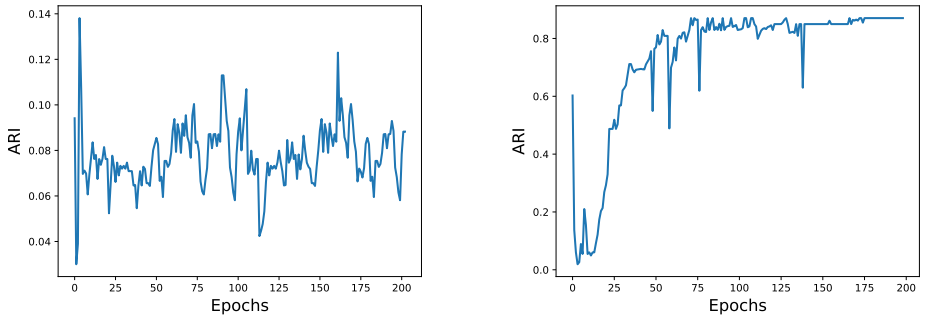
| Method | 5% | 10% | 15% | 50% |
|---|---|---|---|---|
| COP-KMeans | 0.001 (0.048) | 0.008 (0.055) | −0.003 (0.047) | 0.006 (0.064) |
| Spec | −0.034 (0.068) | −0.030 (0.062) | −0.043 (0.078) | −0.049 (0.101) |
| CCSR | 0.262 (0.298) | 0.263 (0.297) | 0.263 (0.296) | 0.261 (0.296) |
| CPClustering | −0.029 (0.089) | −0.023 (0.067) | −0.021 (0.076) | −0.036 (0.085) |
| SAMARAH | 0.067 (0.087) | 0.076 (0.080) | 0.063 (0.081) | 0.079 (0.089) |
| DCC | −0.033 (0.096) | −0.041 (0.114) | −0.046 (0.113) | −0.031 (0.125) |
| DCC-Conv | 0.066 (0.078) | 0.089 (0.110) | 0.116 (0.103) | 0.188 (0.229) |

TABLEAU 4.5 – Difference d'ARI entre le clustering contraint et non-contraint (ARI contraint − ARI non-contraint) pour chaque fraction de contrainte moyennée sur l'ensemble des jeux de données, l'écart-type est indiqué entre parenthèse.

| Méthode | ECG5000 | ElectricDevices | FacesUCR | InsectWingbeatSound | MALLAT | StarLightCurves | TwoPatterns | uWaveGestureLibraryX | UWaveGestureLibraryAll |
|---|---|---|---|---|---|---|---|---|---|
| DCC | -0.047 | -0.027 | -0.067 | -0.027 | -0.563 | -0.042 | -0.003 | -0.084 | -0.084 |
| DCC-Conv | -0.020 | -0.248 | -0.169 | -0.019 | -0.141 | -0.317 | -0.139 | -0.130 | -0.037 |

TABLEAU 4.6 – Différence d'ARI entre la version de l'encodeur pré-entraîné et la version complète de DCC sans-contrainte. Un écart positif indique un gain à l'utilisation de DCC, un écart négatif indique une dégradation du résultat.

pré-entraînée de l'encodeur (via la fonction de coût de reconstruction) et la version complète d'IDEC (avec le fonction de coût de clustering). Le clustering de la version pré-entrainée a été obtenu en utilisant la méthode KMEANS sur l'espace latent créé par l'encodeur. Le tableau 4.6 montre l'amplitude de cette chute d'ARI en indiquant l'écart d'ARI entre la fin du pré-entraînement et la fin du processus de DCC. On aurait pu s'attendre à une augmentation de l'ARI, mais en fait on constate une baisse systématique de l'ARI, avec une baisse qui peut être très importante, par exemple le jeu de données Mallat et la méthode DCC standard qui obtient une différence de −0.563 qui passe d'un très bon score (de 0.823, le meilleur à l'exclusion des méthodes spectrales) à un très mauvais score (de 0.260, le plus bas parmi l'ensemble des méthodes).




(a) DCC-Conv avec 5% de contraintes (b) DCC-Conv avec 50% de contraintes

FIGURE 4.3 – Évolution de l'ARI au cours du processus d'apprentissage de DCC-Conv sur le jeu de données TwoPatterns.

## 4.3 Discussion et conclusion

Cette première étude sur l'utilisation de méthodes d'apprentissage profond pour le clustering sous contraintes appliqué aux séries temporelles a montré que, même si dans certains cas on peut obtenir de bon résultats, les méthodes existantes ne semblent pas adaptées à ce contexte. La méthode testée ne semble fonctionner que lorsque le nombre de contraintes est très important en proportion de la taille du jeu de données, ce qui n'est pas du tout réaliste dans le cadre de la télédétection, où il faudrait alors définir plusieurs milliers de contraintes.

Il ressort également de cette étude que la méthode de clustering par apprentissage profond étudiée (IDEC), même sans l'utilisation de contraintes ne semble pas vraiment adaptée à l'analyse de séries temporelles. En apprentissage profond, de nombreuses études ont montré que la modification de certains paramètres de ces méthodes pouvait avoir un fort impact sur les performances de celles-ci. Il nous a donc semblé important qu'avant de revenir sur l'impact des contraintes pour l'apprentissage d'une représentation adéquate, il fallait s'assurer que le modèle d'apprentissage profond utilisé était approprié pour le clustering de séries temporelles.

# Chapitre 5

# Apprentissage profond non supervisé et séries temporelles

**Sommaire**

Nous avons vu dans le chapitre précédent que le choix des types de couches utilisés ou la manière d'entraîner le réseau de neurone (DNN) peut avoir un fort impact sur la qualité de la représentation apprise. Dans ce dernier chapitre nous allons présenter une étude que nous avons conduit afin de mieux comprendre l'impact de ces choix dans le cadre du clustering de séries temporelles.

Pour cela, nous allons tous d'abord revenir sur les différentes méthodes de l'état de l'art qui permettent de construire et entraîner un modèle DNN pour le clustering de séries temporelles (section 5.1), nous expliquons ensuite plus en détails la méthodologie et les résultats de notre étude (section 5.2). Nous proposons ensuite différentes approches pour mieux comprendre les résultats, avec notamment la proposition d'une nouvelle méthode d'interprétabilité des résultats (section 5.3). Enfin nous conclurons sur les limites des modèles existants et les perspectives de développement que nous avons identifiées (section 5.4).

## 5.1 État de l'art et adaptation de l'existant

En apprentissage profond, il est courant de proposer une méthode reposant sur la modification d'un ou plusieurs éléments d'un modèle existant, par exemple, un nouveau type de couche, une nouvelle méthode d'optimisation, une nouvelle fonction de coût ou encore une autre manière de préparer les données. Toutefois, nous avons pu relever dans la littérature que les méthodes proposées en clustering reposent généralement sur le choix ou la proposition de trois éléments principaux, l'architecture de l'encodeur (section 5.1.1), la fonction de coût de prétexte (section 5.1.2) et la fonction de coût de clustering (section 5.1.3). Dans la suite nous allons donc aborder ces trois éléments séparément et présenter les principales approches existantes dans la littérature.

### 5.1.1 L'architecture de l'encodeur

Nous avons vu précédemment qu'un réseau de neurones, l'encodeur dans notre cas, est composé d'un ensemble de couches. Dans cette thèse nous ferons référence au terme *architecture* comme étant l'ensemble de ces couches avec leurs hyper-paramètres (nombres de couches, taille de chaque couches, fonction d'activation, ...).

Différents types de couches existent, dont certains ont été développés pour prendre en compte spécifiquement la dimension temporelle des données. Ces types de couches peuvent être divisés en trois principales familles : les couches complètement connectées, convolutives et récurrentes.

#### FCNN : Couches complètement connectées (*Fully Connected Neural Network*)

Il s'agit de la forme la plus simple de couches pour les DNNs [Rosenblatt, 1958]. Dans ce type de couche, chaque neurone de la couche courante est connecté à chaque neurone de la couche précédente. Dans le cas de l'encodage d'une série temporelle $x \in \mathbb{R}^{d \times T}$ avec un réseau de $L$ couches, le résultat de la $i^{\text{ème}}$ couche $l_i$ peut être exprimé par l'équation suivante :

$$\begin{aligned} out_{l_0} &= x, \\ \forall 0 < i < L, out_{l_i} &= a_{l_i}(W_{l_i} out_{l_{i-1}} + b), \end{aligned} \tag{5.1}$$

où $W_{l_i} \in \mathbb{R}^{r \times s}$ est l'ensemble des poids de la couche, $r$ est la dimension du vecteur $out_{l_{i-1}}$, $s$ le nombre de neurones dans la couche $l_i$, $b$ le terme de biais et $a_{l_i}$ une fonction d'activation (par exemple, $ReLU$, tanh ou l'identité).

Les couches FCNN peuvent être utilisées pour tout type de données, indépendamment du nombre de dimensions ou de la taille des données en entrée, tant que les données ont une taille fixe. Notez qu'une donnée peut être complétée, par exemple avec des zéros, pour atteindre la taille d'entrée fixée. Cela rend ce type de couches facile à utiliser dans divers domaines,

comme le clustering d'image [Guo et collab., 2017a; Xie et collab., 2016], la segmentation d'image [Sun et collab., 2013], ou la classification de séries temporelles [Fawaz et collab., 2019]. Cependant, ces couches ont plus de difficultés à capturer les relations de dimensions spécifiques telles que l'espace ou le temps. En effet, pour une série temporelle, chaque pas de temps $t_i$ a son propre neurone et les liens avec ses *voisins temporels* ($t_{i-1}$ et $t_{i+1}$) sont perdus lors du calcul de la sortie de la couche (voir équation 5.1). D'ailleurs les couches FCNN sont en général utilisées pour la couche latente $l_z$ afin de s'affranchir de la dimensionnalité spécifique des données, du nombre de pas de temps $T$, et du nombre de caractéristiques $d$ dans notre cas. Ainsi, nous obtenons une représentation sous la forme d'un vecteur de taille fixe donnée par l'utilisateur. Cela permet d'utiliser une méthode de clustering agnostique à la structure spécifique des données tout en uniformisant le traitement des données.

### CNN : Couches convolutives 1D (*Convolutional Neural Network*)

Les couches convolutives ont permis d'obtenir d'énormes progrès dans les performances des DNNs, notamment en vision par ordinateur [Krizhevsky et collab., 2012; LeCun et collab., 1998]. Contrairement aux couches FCNN qui traitent tous les pas de temps indépendamment, elles visent à tirer profit des motifs hiérarchiques dans les données en apprenant des motifs petits et simples dans les premières couches et en les assemblant en allant vers les dernières couches. De la même manière que pour les images les convolutions 2D sont utilisées pour capturer les motifs spatiaux, les convolutions 1D sont utilisées pour capturer les motifs temporels. Ce type de couches a déjà démontré sa capacité à apprendre des représentations utiles sur des tâches supervisées [Fawaz et collab., 2019; Wang et collab., 2017]. Pour les convolutions 1D, chaque couche consiste à appliquer un ensemble $F$ de $m$ filtres de taille $k$ à la séquence d'entrée. Cela correspond pour chaque pas de temps $t$ à faire le calcul suivant :

$$out_{l_i,t} = a_{l_i}(F[out_{l_{i-1},t-\lfloor\frac{k}{2}\rfloor}, out_{l_{i-1},t-\lfloor\frac{k}{2}\rfloor+1}, ..., out_{l_{i-1},t+\lfloor\frac{k+1}{2}\rfloor}]) \tag{5.2}$$

où $out_{l_i,t} \in \mathbb{R}^m$. Ceci nous permet d'obtenir en sortie la séquence suivante :

$$out_{l_i} = (out_{l_i,1+\lfloor\frac{k}{2}\rfloor}, out_{l_i,2+\lfloor\frac{k}{2}\rfloor}, ..., out_{l_i,T-\lfloor\frac{k}{2}\rfloor}) \tag{5.3}$$

où $out_{l_i} \in \mathbb{R}^{m\times T}$.

Pour les convolutions 2D, deux paramètres, en plus du nombre et de la taille des filtres, sont souvent utilisés :

— le *stride* : ce paramètre contrôle la taille du décalage (nombre de pas de temps pour les convolutions 1D) effectués par le filtre entre chaque calcul.

— Le *padding* : ce paramètre définit la manière dont le bord d'un échantillon est traité par le filtre, en remplissant les extrémités la séquence d'un certain nombre de zéros.

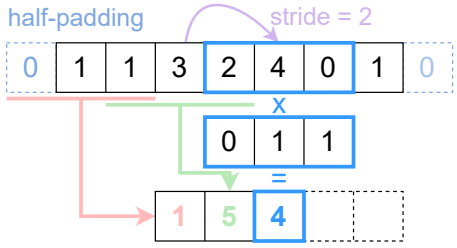
Pour un *stride* de valeur $s$, et une valeur de *padding* de $p$, le calcul de la sortie d'une couche devient :

$$out_{l_i} = (out_{l_i,1+\delta+0\times s}, out_{l_i,1+\delta+1\times s}, ..., out_{l_i,\lceil\frac{T-\delta}{s}\rceil}) \tag{5.4}$$

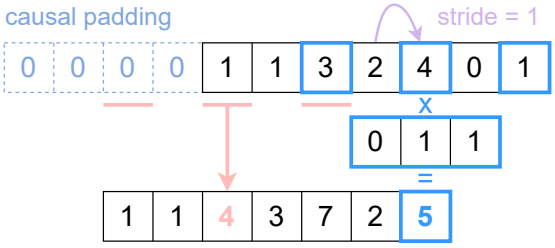
où $\delta = \lfloor\frac{k}{2}\rfloor - p$.

Un *half-padding* (*padding* de la moitié de la taille du filtre arrondie à l'inférieur) est souvent utilisé (figure 5.1a). En effet, si la taille $k$ du filtre est supérieure à 1, la convolution rognerait le bord de la séquence produite. Même si ces paramètres sont souvent modifiés en traitement d'images, un *stride* de 1 et un *half-padding* sont généralement utilisés pour les séries temporelles [Fawaz et collab., 2019; Wang et collab., 2017; Xiao et Cho, 2016]. En conséquence, la sortie de chaque couche conserve la taille $T$ de sa dimension temporelle.

Cependant, une autre technique de *padding* est parfois utilisée pour les séries temporelles afin de prendre en compte la spécificité de la dimension temporelle, appelée *causal padding*. Au lieu d'un *padding* des deux côtés de la séquence d'entrée, un *padding* de $k-1$ est ajouté




(a) Convolution 1D avec un filtre de taille 3, un *stride* 2 et un *half-padding*




(b) Convolution 1D avec un filtre de taille 3, un *stride* 1, un *padding* causal et une **dilatation** de 2

FIGURE 5.1 – Illustrations du calcul de convolutions 1D.

au début de la séquence. En conséquence, le pas de temps $t$ de la séquence de sortie est calculé/prédit, uniquement sur la base des pas de temps antérieurs ou égaux à $t$ de la séquence d'entrée. Notez que dans ce cas, le *padding* n'est ajouté qu'au début et la taille de la dimension temporelle est également conservée à la sortie de chaque couche CNN.

Le *padding* causal est, en général, combiné avec un autre paramètre appelé *dilatation* [YU et KOLTUN, 2015] (figure 5.1b). La convolution dilatée fonctionne comme le *stride*, mais l'effet de *stride* est appliqué sur le noyau au lieu de la séquence d'entrée. Ainsi, l'équation 5.2 peut être réécrite avec un facteur de *dilatation* $d$ comme :

$$out_{l_i,t} = a_{l_i}(F[out_{l_{i-1},t-\lfloor\frac{k\times d}{2}\rfloor+0\times d}, out_{l_{i-1},t-\lfloor\frac{k\times d}{2}\rfloor+1\times d}, ..., out_{l_{i-1},t+\lfloor\frac{k\times d+1}{2}\rfloor}]) \tag{5.5}$$

On peut noter qu'un facteur de dilatation de 1 correspond à la convolution basique. Habituellement, le facteur $d$ est fixé de manière à croître exponentiellement, le facteur de dilatation étant multiplié par 2 à chaque couche et la première couche utilise un facteur de dilatation de 1 pour préserver la dépendance au pas de temps précédent.

### RNN : couche récurente (Recurrent Neural Network)

Les couches récurrentes ont été proposées spécifiquement pour prendre en compte la dimension temporelle [HOCHREITER et SCHMIDHUBER, 1997; HOPFIELD, 1982]. Ce type de couche a permis d'énormes progrès en traitement automatique du langage [SAK et collab., 2014; SUTSKEVER et collab., 2014]. Contrairement aux autres types de couches, la séquence d'entrée est introduite dans la couche par pas de temps pour mettre à jour l'état caché (*hidden state* en anglais) de la couche. Cet état peut être considéré comme la mémoire du traitement des étapes précédentes. La couche elle-même consiste à appliquer une fonction récursive $g$ qui prend en entrée le pas de temps courant $x_t$ et l'état caché précédent $h_t$ et qui produit en sortie le nouvel état caché :

$$h_t = g(x_t, h_{t-1}) \tag{5.6}$$

En général, le vecteur $h_0$ est initialisé avec des valeurs à zéro. La fonction récursive originellement proposée est définie comme :

$$h_t = \tanh(Wx_t + Uh_{t-1} + b) \tag{5.7}$$

où $h_t$ est un vecteur de taille $u$, $u$ étant le nombre d'unités, $W \in \mathbb{R}^{u\times d}$ et $U \in \mathbb{R}^{u\times u}$ étant les poids et $b \in \mathbb{R}^u$ le vecteur de biais de la couche qui sont appris pendant l'apprentissage. Cependant, ce type de fonctions récursives, appelée *cellule* pour les RNN, conduit souvent au problème de disparition de gradient [BENGIO et collab., 1994] (le gradient atteint une valeur trop petite au point de devenir nul), rendant l'apprentissage difficile. Pour palier ce problème, d'autres types de cellules ont été proposés dans la littérature : les cellules de mémoire à long terme (LSTM : *Long Short-Term Memory*) et les cellules d'unités récurrentes à grille (GRU : *Gated Recurrent Unit*).

Une *unité LSTM* [Gers et collab., 2000; Hochreiter et Schmidhuber, 1997] se compose de quatre sous-unités, généralement appelées "portes", qui contrôlent la mise à jour de l'état caché et de la sortie de la couche : une porte d'entrée (*input gate*), une porte de sortie (*output gate*), une porte d'oubli (*forget gate*), et une porte de mémoire candidate (*candidate memory gate*). Chaque porte est respectivement calculée comme suit :

$$\begin{aligned} i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\ o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\ f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\ \tilde{c}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c) \end{aligned} \tag{5.8}$$

où $\sigma$ est la fonction sigmoïde.
A partir de ces portes, la cellule mémoire (*memory cell*) est mise à jour par :

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \tag{5.9}$$

où $\circ$ désigne le produit matriciel de Hadamard. L'état caché est mis à jour comme :

$$h_t = o_t \circ \tanh(c_t) \tag{5.10}$$

Une *unité GRU* [Cho et collab., 2014] est similaire à LSTM mais fonctionne avec moins de paramètres. Les performances des cellules GRU se sont révélées similaires à celles des LSTM, même si elles semblent plus limitées que ces derniers sur certaines tâches [Weiss et collab., 2018]. Une telle unité se compose de trois portes : une porte de mise à jour (*update gate*), une porte de réinitialisation (*reset gate*) et une porte candidat (*candidat gate*), respectivement calculées comme :

$$\begin{aligned} z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\ r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\ \hat{h}_t &= \tanh(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h) \end{aligned} \tag{5.11}$$

L'état caché est alors mis à jour par :

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \hat{h}_t \tag{5.12}$$

Pour tous ces types d'unités, l'état caché, après passage de toute la séquence $h_T$, constitue généralement la représentation apprise de la séquence.

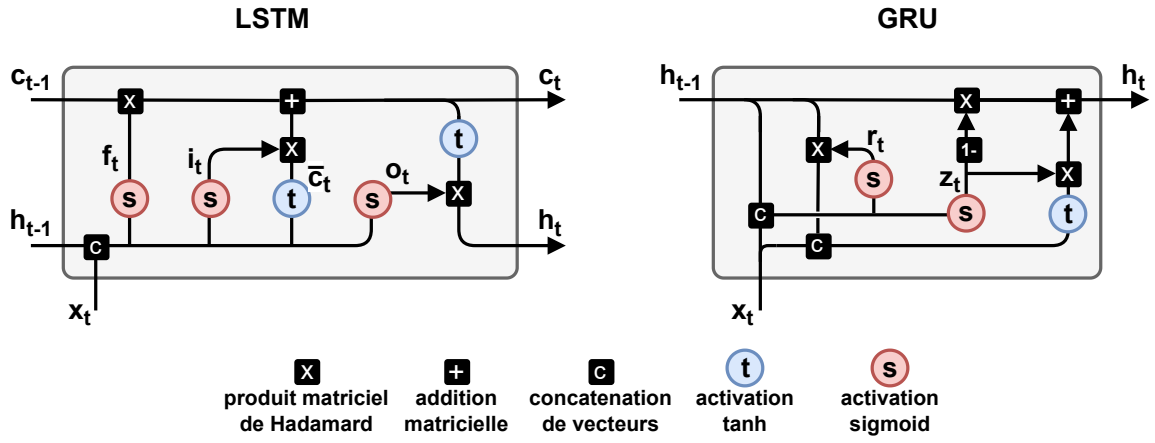



FIGURE 5.2 – Cellules LSTM et GRU avec $x_t$ en entrée et le calcul de l'état caché $h_t$, et pour LSTM le calcul additionnel de l'état de la cellule $c_t$.

D'autres options sont souvent utilisées au sein des RNNs. La plus courante est l'utilisation de couches bidirectionnelles. Elle consiste à entraîner en parallèle deux couches RNN identiques, mais l'une traite chaque pas de temps dans l'ordre chronologique (1 à $T$), et l'autre les traite dans l'ordre inverse ($T$ à 1), on parle alors de couches *forward* (ordre chronologique) et *backward* (ordre inverse). De même que pour les couches CNN, l'utilisation de la *dilatation* a été proposée [Chang et collab., 2017], même si elle est moins utilisée en pratique. L'utilisation des dilatations pour les couches RNN est plus simple que pour les CNN. Elle peut être résumée, pour un facteur de dilatation $d$, à l'équation suivante :

$$h_t = cell(x_t, h_{t-d}), \tag{5.13}$$

où *cell* est la fonction qui calcule de l'état caché de l'unité RNN. Il s'agit en fait de sauter les états cachés directement précèdant en calculant l'état caché au temps $t$ par rapport au $d^{\text{ème}}$ état caché précédent. Elle est également utilisée avec un facteur $d$ exponentiellement croissant. Enfin, on peut souligner que les RNN ont été initialement proposés avec une couche unique. Cependant, ils peuvent être empilés [Pascanu et collab., 2013]. Comme les couches RNN doivent être alimentées par une séquence, chaque couche RNN supplémentaire prend alors en entrée la séquence des états cachés de la couche RNN précédente.

### Mécanisme d'attention

Un mécanisme d'attention, plus particulièrement de *self-attention* dans notre cas, permet de quantifier l'interdépendance entre les éléments de l'objet en entrée afin de concentrer "l'attention" portée par le réseau principalement (voir uniquement) sur les éléments importants pour la tâche d'entraînement. Les mécanismes d'attention viennent souvent en complément ou comme base à de nouveaux modèles d'appentissage profond, avec les Transformeurs [Vaswani et collab., 2017]. Ces mécanismes ont suscité un fort intérêt récemment dû à leur très bonnes performances, dépassant celles des convolutions dans certains cas, mais aussi de bien plus faibles temps de calculs [Vaswani et collab., 2017]. Les mécanismes d'attention ont d'abord été utilisés en traitement automatique des langues [Bahdanau et collab., 2014; Vaswani et collab., 2017], puis en vision par ordinateur [Guan et collab., 2018; Jaderberg et collab., 2015; Woo et collab., 2018]. Quelques méthodes ont été proposées pour l'apprentissage non supervisé, mais principalement sur des cas spécifiques, comme le clustering de graphes [Wang et collab., 2019a] ou l'utilisation d'attention spatiale [Souza et Zanchettin, 2019]. Seul un petit nombre de travaux a été conduit pour le clustering de série temporelles.

On trouve notamment la méthode DeTSEC (Deep Time Series Embedding Clustering). Cette méthode, proposée par Ienco et Pensa [2019], repose sur l'utilisation d'un autoencodeur composé d'une couche GRU bidirectionnelle pour l'encodeur et le décodeur. Toutefois, un mécanisme d'attention est placé en sortie de la couche *forward* et de la *backward* de la couche bidirectionnelle. La couche d'attention $h^{att}$ est calculée de la manière suivante :

$$\begin{aligned} v_a &= \tanh(H.W_a + b_a) \\ \lambda &= SoftMax(v_a \circ u_a) \\ h^{att} &= \sum_{j=1}^{T} \lambda_j . h_{t_j} \end{aligned} \tag{5.14}$$

où $\circ$ désigne le produit matriciel de Hadamard, $H \in \mathbb{R}^{T,l}$ est une matrice construite en empilant verticalement l'ensemble des états cachés $h_{t_j}$ appris aux $T$ différents pas de temps par la couche GRU, avec $l$ la taille de l'état caché de la couche. La matrice $W_a \in \mathbb{R}^{l,l}$ et les vecteurs $b_a, u_a \in \mathbb{R}^l$ sont des paramètres appris par le réseau. La représentation latente est alors calculée pour un objet $x$ en entrée en utilisant la sortie du mécanisme d'attention de la couche *forward* ($h^{att}_{forw}$) et *backward* ($h^{att}_{back}$), comme :

$$\begin{aligned} f(x) &= gate(h^{att}_{back}) \circ h^{att}_{back} + gate(h^{att}_{forw}) \circ h^{att}_{forw} \\ gate(o) &= \sigma(W_g.o + b_g) \end{aligned} \tag{5.15}$$

où $\sigma$ est la fonction sigmoid et $W_g$ et $b_g$ sont des paramètres appris lors de l'entraînement du modèle. La fonction *gate* ajoute un niveau de décision supplémentaire afin de mieux discriminer les informations remontées par les couches *forward* et *backward*.

On peut encore évoquer la méthode proposée par Jiao et collab. [2020]. C'est une méthode générale pouvant traiter plusieurs tâches, telle que la détection d'anomalies ou le clustering. Elle repose sur un modèle général constitué des plusieurs modules, pouvant être activé ou non selon certains hyperparamètres, fixés par optimisation bayesienne [Shahriari et collab., 2015]. Toutefois, les auteurs ne donnent pas d'information précise sur l'implémentation utilisée pour le mécanisme d'attention.

### 5.1.2 Entraîner les paramètres de l'encodeur pour obtenir des caractéristiques représentatives

L'objectif des méthodes de deep clustering est d'entraîner l'encodeur à générer une représentation qui favorisera le regroupement des données en groupes pertinents. Comme les étiquettes sont inconnues en apprentissage non supervisé, l'encodeur doit être optimisé suivant un autre objectif. Pour ce faire, un objectif auto-supervisé (*self-supervised* en anglais), où les données fournissent la supervision, pour obtenir des caractéristiques représentatives est très souvent utilisé. Cet objectif est aussi appelé tâche "prétexte" ou "proxy" [Doersch et collab., 2015; Larsson et collab., 2017; Xu et collab., 2019]. Comme le terme "fonction de coût proxy" est souvent utilisé pour désigner une fonction de coût beaucoup plus facile à optimiser sur le plan informatique que la fonction de coût standard, nous utiliserons le terme "fonction de coût prétexte" pour les désigner dans la suite de cette thèse.

#### Autoencodeurs (AEs)

A l'origine les AEs ont été proposés comme des méthodes de réduction de dimensionnalité [Kramer, 1991; Rumelhart et collab., 1986] ou comme des méthodes de pré-entraînement [Ballard, 1987]. Ils ont aussi montré qu'ils pouvaient générer des représentations utiles pour des tâches de clustering [Becker, 1991]. Ce sont les premiers types de modèles de DNNs utilisés pour le clustering et ils restent à la base de la plupart des modèles existants.

Les AEs sont composés de deux parties, un encodeur, $f$, et un décodeur, $g$. L'encodeur est une fonction non linéaire $f : X \to Z$ qui projette les données dans un espace latent (comme décrit dans la section 4.1.2), et le décodeur une fonction non linéaire $g : Z \to X$ qui projette les données de l'espace latent dans l'espace des données. En conséquence, un objet $x_i$ peut être passé à l'encodeur pour obtenir sa représentation $z_i$, puis $z_i$ peut être passé au décodeur pour obtenir un nouvel objet $\hat{x_i}$, appelé la *reconstruction* d'$x_i$. Un AE est évalué par sa capacité à reconstruire fidèlement $x_i$ en $\hat{x_i}$. Ainsi, nous attendons de l'encodeur qu'il soit capable de conserver les caractéristiques importantes de l'espace des données dans l'espace latent pour permettre une bonne reconstruction. Pour entraîner les poids de l'AE, il suffit de minimiser l'erreur quadratique moyenne, également appelée *fonction de coût de reconstruction* :

$$L_r = \frac{1}{n}\sum_{i=1}^{n}\|x_i - g(f(x_i))\|^2 \tag{5.16}$$

Le décodeur est, en général, construit en miroir (des couches identiques mais ordonnées inversement) de l'encodeur [Guo et collab., 2017a; Kramer, 1991; Xie et collab., 2016] à l'exception de la couche latente $l_z$.

#### Autoencodeurs régularisés

Les premières alternatives aux AEs ont consisté en des formes régularisées de l'AE classique :

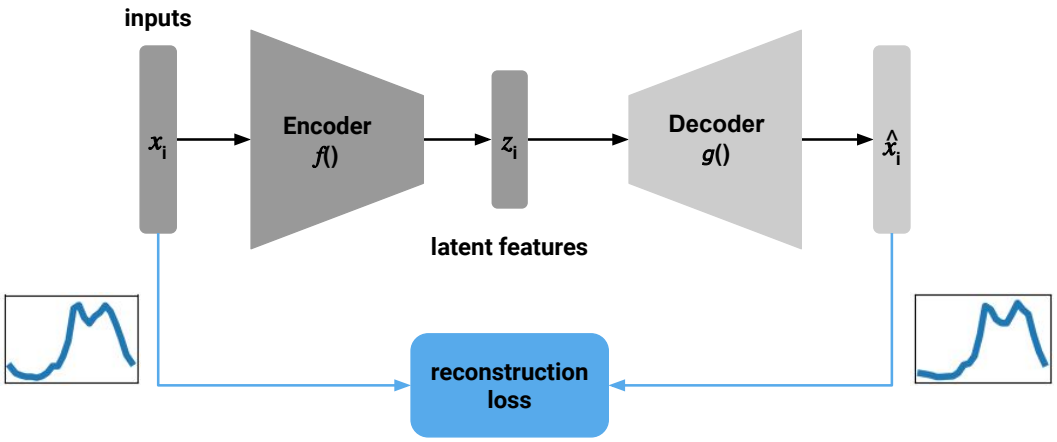



FIGURE 5.3 – L'architecture d'un Autoencodeur qui est entraîné à reconstruire les séries temporelles en entrée.

— Autoencodeur de débruitage (DAE : *Denoising Autoencoder*) [VINCENT et collab., 2008] : Au lieu de fournir en entrée l'objet initial ($x_i$) à l'AE, on utilise une version partiellement corrompue ($\tilde{x_i}$). L'objectif est de nettoyer l'entrée corrompue, ce qui permet d'obtenir une représentation robuste aux petites variations dans les données en entrée. La corruption, ou le bruit, est générée de manière aléatoire pour chaque objet et à chaque itération. Il existe différentes méthodes pour générer l'entrée corrompue. À l'origine, les auteurs ont utilisé un bruit par masquage (une fraction du vecteur en entrée est passée à 0), mais d'autres types, comme le bruit gaussien isotopique ou le bruit sel et poivre (une fraction de l'entrée est fixée à la valeur min ou max), peuvent être utilisés. Notez que la corruption n'est appliquée que pendant la phase d'entraînement du modèle. Aucune corruption n'est ajoutée lorsque la représentation est calculée pour la tâche de clustering. Le DAE est donc entraîné avec la fonction de coût suivante :

$$L_{dae} = \frac{1}{n}\sum_{i=1}^{n}\|x_i - g(f(\tilde{x_i}))\|^2 \tag{5.17}$$

Une forme dérivée des DAEs, les autoencodeurs de débruitage empilés (SDAE : *Stacked DAE*), a également été proposée et a donné lieu à de bons résultats [VINCENT et collab., 2008; XIE et collab., 2016]. Le concept est similaire à celui des DAEs, mais il diffère par l'inclusion d'une phase de pré-entraînement effectuée une couche à la fois. Pour chaque étape $i$ de 1 à $L$, l'AE est réduit aux $i$ premières couches de l'AE final et des $i$ dernières couches du décodeur final. Ensuite, seule la $i^{\text{ème}}$couche de l'encodeur et la $i^{\text{ème}}$ dernière couche du décodeur sont entraînées pour la tâche de débruitage. Cette phase de pré-entraînement est suivie d'une dernière phase d'entraînement dans laquelle les couches sont conjointement entraînées, qu'on appelle la phase de *fine-tuning*.

— Autoencodeur creux (SAE : *Sparse Autoencoder*) [MAKHZANI et FREY, 2013] : Les SAEs diffèrent des AEs en ne permettant qu'à un petit nombre de neurones d'être actifs en même temps dans la couche latente. Pour une *m-sparsité*, cela consiste simplement à sélectionner les $m$ plus grandes valeurs de sortie de l'encodeur et à mettre les autres à zéro avant de les transmettre au décodeur. D'autres versions expriment le terme de pénalité de *sparsité* directement dans la fonction de coût en tirant parti, par exemple, de la divergence de Kullback-Leiber [ZENG et collab., 2018].

— Autoencoder contractile (CAE : *Contractive Autoencoder* ) [RIFAI et collab., 2011] : Alors que les DAEs sont conçus pour augmenter la robustesse de la reconstruction à de petites modifications de l'entrée, les CAEs sont conçus pour augmenter la robustesse de la représentation elle-même. Le terme de régularisation correspond à la norme de

Frobenius ($\|.\|_F^2$) de la matrice jacobienne de l'activation de l'encodeur, $J_f$, par rapport à l'entrée $x_i$. Nous obtenons la fonction de coût suivante :

$$L_{cae} = L_r + \lambda \sum_{i=1}^{n} \|J_f(x_i)\|_F^2 \tag{5.18}$$

où $\lambda$ est un hyperparamètre qui contrôle la force de la régularisation. Cette régularisation permet aux CAEs d'ignorer les variations présentes dans les données (par exemple, la translation ou la rotation pour les images) mais aussi des variations plus petites et plus rares (présentes dans des exemples spécifiques), tandis que la fonction de coût de reconstruction assure que la reconstruction reste fidèle [Rifai et collab., 2011].

### Méthodes génératives

Contrairement aux AEs qui reposent sur la tâche de reconstruction, d'autres méthodes s'appuient sur la génération de données réalistes pour entraîner l'encodeur :

— Autoencodeur variationnel (VAE : Variational Autoencoder) [Kingma et Welling, 2013] : Basés sur l'architecture des AEs (un encodeur et un décodeur), les VAEs diffèrent significativement par la manière dont le modèle est entraîné. Pour cela les VAEs exploitent la capacité du décodeur à générer des données. Dans l'AE, le décodeur n'est utilisé que pour reconstruire une entrée précédemment encodée, mais il pourrait être envisagé de prendre un point aléatoire dans l'espace latent et de le décoder pour obtenir un nouvel objet. Cependant, cela suppose que l'espace latent soit suffisamment régulier. Les VAEs visent à introduire une telle régularisation en supposant que les données suivent une distribution, en pratique une distribution gaussienne isotopique.

Le concept des VAEs consiste à transmettre un ensemble d'objets $x$ à l'encodeur et le coupler avec une distribution gaussienne $q(z_g|x)$. Un échantillon $z_g$ est alors tiré de $q(z_g|x)$, puis passé à travers le décodeur pour obtenir la distribution $p(x|z_g)$. Pour entraîner le réseau, nous utilisons la fonction de coût suivante [Kingma et Welling, 2013], appelée *Evidence Lower BOund* (ELBO) :

$$L_{vae} = \sum_{i=1}^{n} - \mathop{\mathbb{E}}_{z_g \sim q(z_g|x_i)} [\log p(x_i|z_g)] + KL(q(z_g|x_i) \| p(z_g)) \tag{5.19}$$

Le premier terme est la fonction de coût de reconstruction pour assurer que le décodeur génère des données fidèles à celles d'entrée. Le second est un terme de régularisation qui vise à faire converger la distribution attendue $p(z_g)$ vers celle générée par l'encodeur. La distribution $p(z_g)$ est construite comme une distribution normale standard de moyenne zéro et de variance un.

Néanmoins, la représentation apprise par un VAE n'est pas la plus adaptée au clustering. Certains travaux ont proposé de légères modifications pour répondre à ce problème. Jiang et collab. [2016] proposent une méthode appelée Variational Deep Embedding (*VADE*), qui, au lieu d'essayer d'apprendre une distribution, apprend autant de distributions que de clusters attendus. Cela consiste, dans l'équation 5.19, à remplacer la distribution unique $q(z_g|x)$ par un ensemble de distributions $q(z_g, c|x)$ avec $c \in 1, .., K$ pour $K$ clusters et la distribution attendue $p(z_g)$ par $p(z_g, c)$. L'ensemble des distributions est initialisé à partir d'un modèle de mélange gaussien sur l'espace latent préentraîné avec une autre fonction de coût (par exemple un AE, ou un VAE). Li et collab. [2018] ont proposé une méthode basée sur les VAEs pour apprendre une structure de regroupement multi-facettes de l'espace latent au lieu d'une seule partition.

— Réseaux antagonistes génératifs (GAN : *Generative Adversarial Network*) [Goodfellow et collab., 2014] : Les GANs sont composés de deux éléments, un générateur $G_{\Theta_G}$ et un discriminateur $D_{\Theta_D}$, qui sont tous deux des DNNs. Le générateur, $G : Z \to X$,

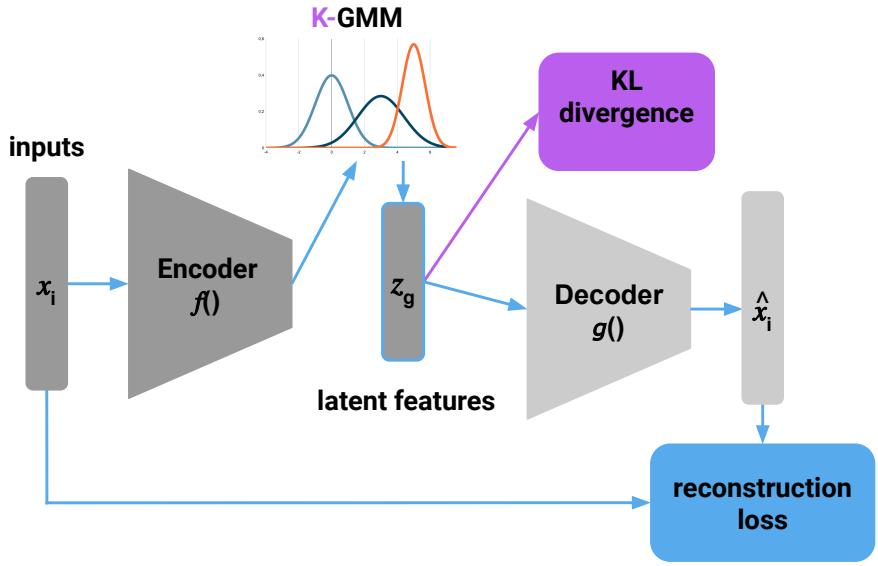



FIGURE 5.4 – La méthode *VADE* avec $K$ clusters : les représentations encodées sont utilisées pour générer $K$ distributions gaussiennes. Des échantillons sont générés pour chaque distribution. Le modèle est entraîné à reconstruire les données et à faire converger la distribution générée avec la distribution originale.

génère des données à partir de l'espace latent, de manière similaire au décodeur de l'AE. Le discriminateur, $D : X \to \mathbb{R}$, génère un scalaire, compris entre 0 et 1, à partir de l'espace de données qui peut être interprété comme la probabilité que les données soient réelles (en opposition aux données générées). Les deux réseaux sont entraînés dans une compétition à deux joueurs par l'objectif min-max suivant :

$$\min_{\Theta_G} \max_{\Theta_D} \mathop{\mathbb{E}}_{x \sim \mathbb{P}_x^r} q(D(x)) + \mathop{\mathbb{E}}_{z \sim \mathbb{P}_z} q(1 - D(G(z))) \tag{5.20}$$

où $\mathbb{P}_x^r$ est la distribution des échantillons de données réelles, $\mathbb{P}_z$ est la distribution a priori du bruit sur l'espace latent, et $q(x)$ est la fonction de qualité. Pour le GAN classique, $q(x) = \log x$, et pour le GAN de Wasserstein $q(x) = x$. Il est alors attendu qu'à la fin de l'apprentissage que $\mathbb{P}_z$ ait convergé vers $\mathbb{P}_x^r$.

Cependant, on peut remarquer que les GANs n'incluent aucun élément qui pourrait être assimilé à un encodeur. Par conséquent, ils ne peuvent pas être utilisés tels quels à des fins de clustering, car aucune représentation ne peut être extraite des données. Quelques travaux ont été proposés dans la littérature pour inclure un encodeur. Dans LIPTON et TRIPATHI [2017], les auteurs ont testé de rétropropager les données à travers le générateur $G$, mais la représentation obtenue n'était pas vraiment adaptée à des fins de clustering [MUKHERJEE et collab., 2019]. GHASEDI et collab. [2019] ont alors ajouté un troisième réseau, un *clustereur* $E : X \to Z$, et ils ont modifié le discriminateur pour non plus discriminer si l'exemple de l'espace de données est réel ou non, mais si la distribution conjointe des échantillons (E(x), x) et (z, G(z)) provient du générateur ou du *clustereur*. Il en résulte la fonction d'objectif suivante :

$$\min_{\Theta_G, \Theta_E} \max_{\Theta_D} \mathop{\mathbb{E}}_{x \sim \mathbb{P}_x^r} q(D(C(x), x)) + \mathop{\mathbb{E}}_{z \sim \mathbb{P}_z} q(1 - D(z, G(z))) \tag{5.21}$$

MUKHERJEE et collab. [2019] ont adopté une démarche similaire en incluant un encodeur, $E$. Mais dans ce cas, ils imposent aux $K$ dernières valeurs de la représentation latente d'être un vecteur de taille $K$ encodé en *one-hot* (une valeur du vecteur est égale à 1 les autres à 0) indiquant à quel cluster l'objet appartient. Par conséquent, la représentation latente est composée de $z = concat(z_n, z_c)$, où $z_n$ est échantillonné tiré à

partir d'une distribution normale et $z_c$ est un vecteur *one-hot* d'un cluster sélectionné au hasard. Les réseaux sont alors entraînés avec la fonction de coût suivante :

$$\min_{\Theta_G,\Theta_E} \max_{\Theta_D} \mathbb{E}_{x\sim\mathbb{P}_x^r} q(D(x)) + \mathop{\mathbb{E}}_{z\sim\mathbb{P}_z} q(1 - D(G(z)))+ \\ \beta_n \mathop{\mathbb{E}}_{z\sim\mathbb{P}_z} \|z_n - E(G(z_n))\|_2^2 + \beta_c \mathop{\mathbb{E}}_{z\sim\mathbb{P}_z} \mathcal{H}(z_c, E(G(z_c)) \quad (5.22)$$

où $\mathcal{H}()$ est la fonction de coût d'entropie croisée. Les deux premiers éléments de l'équation correspondent au GAN classique, le troisième assure la qualité de reconstruction des valeurs ré-encodées, et le dernier assure que la partie générée codée en *one-hot* est préservée par l'encodage. $\beta_n$ et $\beta_c$ sont des poids pour établir un équilibre entre l'entraînement des valeurs continues et discrètes de la représentation latente.

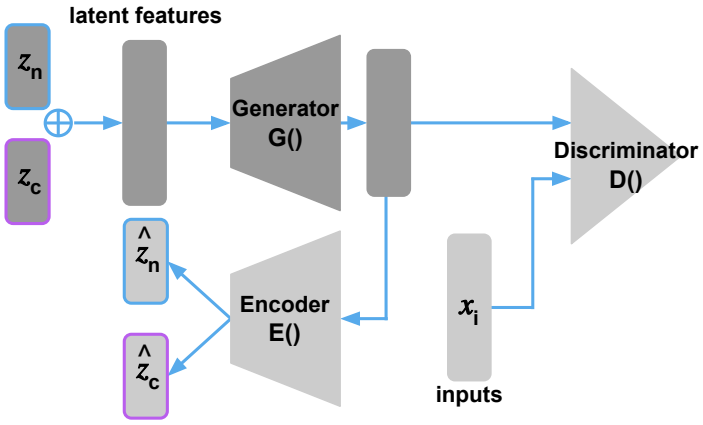



FIGURE 5.5 – La méthode *ClusterGAN* : la représentation latente est séparée en deux composantes, l'une provenant d'une distribution gaussienne, $z_n$, et l'autre pour coder à un coup l'affectation de clustering, $z_c$. Un encodeur est ajouté et entraîné pour préserver à la fois l'encodage $z_n$ et $z_c$.

### La fonction de coût de triplet pour les séries temporelles

Une fonction de coût pour entraîner un encodeur spécifique aux séries temporelles a été proposée, appelée *fonction de coût de triplet* (*triplet loss* en anglais) [FRANCESCHI et collab., 2019]. Cette fonction de coût a l'avantage de nécessiter une architecture constituée uniquement d'un encodeur. La suppression du décodeur présente l'avantage d'un gain de temps de calcul puisque nous n'avons pas besoin d'entraîner ses paramètres, mais aussi de supprimer le problème de conception de l'architecture du décodeur. Cette fonction de coût est basée sur des travaux antérieur [SCHROFF et collab., 2015] qui proposent une fonction de coût basée sur un triplé d'objets, le premier devant être plus proche du second que du troisième. Cependant, la version originale suppose une connaissance supervisée de la similarité entre objets pour construire ces triplets. FRANCESCHI et collab. [2019] ont proposé de résoudre ce problème en utilisant une stratégie d'échantillonnage basée sur le temps.

Ils font l'hypothèse que si nous sélectionnons une sous-série au hasard, $x^{ref}$, à partir d'une série temporelle $x_i$, alors il y a une forte probabilité que les deux affirmations suivantes soient vérifiées. D'une part, la représentation de $x^{ref}$ sera proche de la représentation de n'importe laquelle de ses propres sous-séries $x^{pos}$ (exemple positif), $x^{ref}$ peut être vu comme le *contexte* de $x^{pos}$. D'autre part, la représentation de $x^{ref}$ sera éloignée de toute sous-série $x^{neg}$ prise au hasard dans une autre série temporelle $x_j$, avec $j \neq i$. Ils ont également introduit un autre paramètre $K_{triplet}$ qui définit le nombre d'échantillons négatifs à utiliser pour chaque objet en entrée afin d'améliorer la stabilité de l'entraînement. La fonction de coût est calculée à l'aide de l'équation suivante :

$$L_{triplet} = -\log(\sigma(f(x^{ref})^T f(x^{pos})) - \sum_{l=1}^{K_{triplet}} \log(\sigma(-f(x^{ref})^T f(x_l^{neg})), \quad (5.23)$$

où $\sigma$ est la fonction sigmoïde. Le premier terme entraîne l'encodeur à minimiser la dissimilarité entre la représentation de $x^{ref}$ et de $x^{pos}$, tandis que le second terme entraîne l'encodeur à maximiser la dissimilarité entre la représentation de $x^{ref}$ et les représentations des $K_{triplet}$ $x^{neg}$.

### 5.1.3 Entraîner les paramètres de l'encodeur pour obtenir des caractéristiques adaptées au clustering

Dans la section précédente, nous avons décrit différentes fonctions de coût pour que entraîner le réseau à extraire des caractéristiques représentatives. Cependant, les espaces latents ainsi obtenus ne sont pas nécessairement adaptés au clustering, car même s'ils peuvent bien décrirent les séries temporelles en entrée, ils ne produisent généralement pas un espace latent discret. Dans ce cas, il peut être difficile pour la méthode de clustering de discriminer entre plusieurs choix de partitionnements et de bien délimiter la bordure des clusters.

A cette fin, de multiples méthodes ont proposé d'ajouter, soit en parallèle, soit en post-traitement, une fonction de coût complémentaire pour obtenir un espace latent plus facilement séparable. Nous les appellerons fonctions de coût de clustering.

#### Deep Embedded Clustering (*DEC*)

Proposée par Xie et collab. [2016], cette méthode est l'une des fonctions de coût de clustering les plus citées et elle a fait l'objet de nombreuses adaptations [Bo et collab., 2020; Guo et collab., 2017a; Ma et collab., 2019; Yang et collab., 2019]. L'idée est d'apprendre des clusters au fur et à mesure de l'entraînement des paramètres de l'encodeur. La méthode est divisée en deux étapes. D'abord, les paramètres de l'encodeur sont initialisés par un modèle entraîné d'AE. Ensuite, le décodeur est détaché et les paramètres de l'encodeur sont optimisés en calculant une distribution cible auxiliaire et en minimisant la divergence de Kullback-Leibler (KL) par rapport à celle-ci. Nous allons à présent décrire plus en détails ce processus.

Après la première étape, l'AE permet d'obtenir une forme initiale de l'espace latent final. Un clustering initial avec la méthode Kmeans est effectué sur les données encodées, ce qui produit un ensemble de $K$ centres $\{\mu_j\}_{k=1}^{K}$. Dans la deuxième étape, la méthode calcule une nouvelle distribution de l'espace latent $Q$, à partir de $Z$ en utilisant une t-distribution de Student comme noyau pour mesurer la similarité entre la représentation latente $z_i$ et le centre $\mu_k$ :

$$q_{ij} = \frac{(1 + ||z_i - \mu_k||^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{k=1}^{k}(1 + ||z_i - \mu_k||^2/\alpha)^{-\frac{\alpha+1}{2}}}, \tag{5.24}$$

où $\alpha$ est le degré de liberté de la t-distribution de Student. Les auteurs ont fixé cette valeur à $\alpha = 1$, car elle ne peut pas être directement calculée par validation croisée du fait du contexte non-supervisé [Xie et collab., 2016]. La valeur obtenue $q_{i,k}$ peut être considérée comme le degré de confiance du modèle en l'hypothèse que l'objet $x_i$ appartient au cluster $k$. Par conséquent, pour chaque valeur $x_i$, nous obtenons un vecteur d'affectation floue $q_i$. La fonction qui calcule $q_i$ à partir de $z_i$ est appelée la couche de clustering. L'objectif est maintenant de rendre cette affectation floue $q_i$ plus *dur* (c'est à dire avoir un degré d'appartenance le plus élevé possible sur un seul cluster) et, comme l'expliquent les auteurs, qui possède les propriétés suivantes : (1) renforcer les prédictions (c'est-à-dire améliorer la pureté des clusters), (2) se centrer davantage sur les points de données ayant une confiance élevée dans leur affectation, et (3) normaliser la contribution à la fonction de coût de chaque centre afin d'éviter que les grands clusters ne déforment trop fortement l'espace latent. Pour cela, ils ont utilisé la distribution $P$ définie comme :

$$p_{i,k} = \frac{q_{i,k}^2 / \sum_{i=1}^{N} q_{i,k}}{\sum_{k=1}^{k}(q_{i,k}^2 / \sum_{i=1}^{N} q_{i,k})} \tag{5.25}$$

L'encodeur est alors entraîné avec la KL divergence :

$$L_c = KL(P|Q) = \sum_{i=1}^{N}\sum_{k=1}^{k} p_{i,k} \log \frac{p_{i,k}}{q_{i,k}} \tag{5.26}$$

Afin de définir une méthode générique, les auteurs ont utilisé une architecture simple composée uniquement de couches entièrement connectées (FCNN).

Comme il s'agissait de l'une des premières méthodes de clustering basée sur l'apprentissage profond à donner de très bons résultats, de nombreux travaux ont proposé des extensions de cette fonction de coût de clustering.

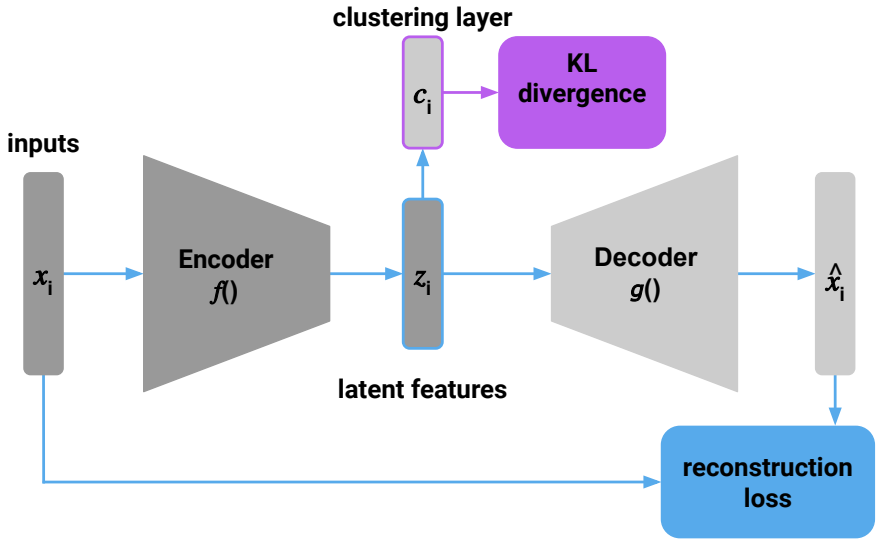



FIGURE 5.6 – La méthode *IDEC* : Le modèle est entraîné à la fois pour reconstruire les données et pour obtenir une représentation plus densément distribuée avec la KL divergence.

La première, Improved Deep Embedded Clustering (*IDEC*) [GUO et collab., 2017a], a proposé une modification simple qui consiste à conserver le décodeur et la fonction de coût de reconstruction pendant la deuxième phase. L'idée est de conserver la représentativité des caractéristiques apprises lors de la première phase. Il en résulte la fonction de coût suivante :

$$L_{IDEC} = (1-\gamma)L_c + \gamma L_r \tag{5.27}$$

Il peut être noté que *DEC* est une application particulière de *IDEC* avec $\gamma = 0$. Cependant, cette modification ne semble pas pertinente pour tous les jeux de données [GUO et collab., 2017a]. On peut également mentionner la version *IDEC* qui utilise un AE convolutionels [GUO et collab., 2017b] au lieu de couches FCNN. Cette dernière adaptation montre déjà l'importance du choix l'architecture de l'AE sur les résultats du modèle.

Des variantes plus sophistiquées ont également été proposées. Par exemple, la méthode Structural Deep Clustering Network (*SDCN*) [BO et collab., 2020] intègre des convolutions de graphes (GCN : *Graph Convolutional Network*) [KIPF et WELLING, 2016] qui sont entraînées en parallèle de l'encodeur d'*IDEC*. Les auteurs l'utilisent pour conserver les relations de voisinage de l'espace des données dans l'espace latent. Un graphe KNN est créé à partir de l'ensemble d'entraînement, où une arête est ajoutée entre chaque objet et ses $k_{knn}$ plus proches voisins. Chaque couche de l'encodeur est régularisée par une couche GCN.

### Les autres fonctions de coût de clustering notables

De nombreuses autres méthodes ont été proposées dans la littérature, voici les principales de façon non exhaustives :

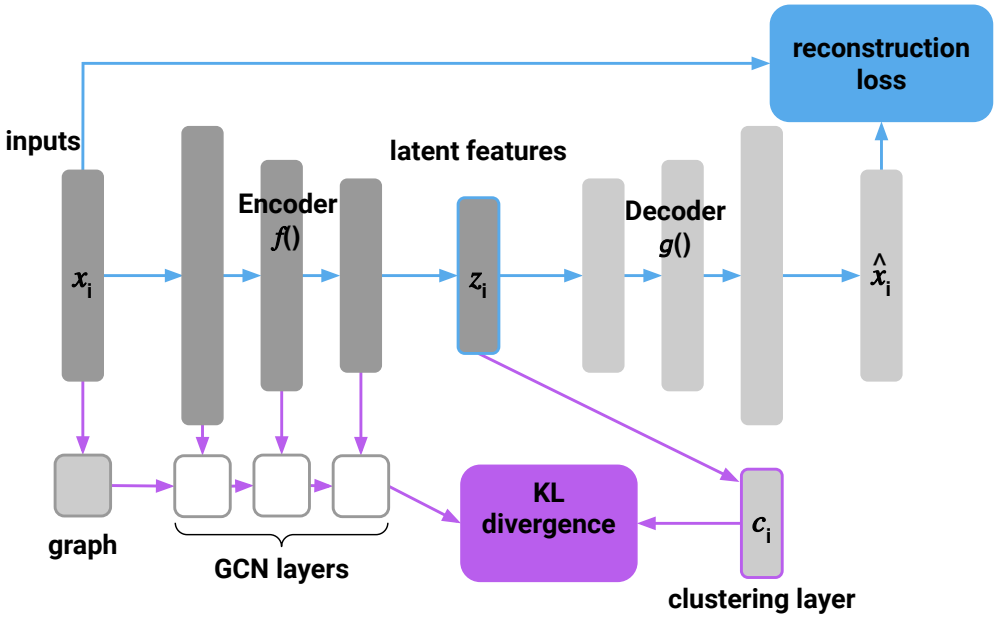



FIGURE 5.7 – La méthode *SDCN* : des couches GCN sont ajoutées au modèle IDEC pour préserver la structure locale de l'espace des données dans l'espace latent.

— *DeepCluster* [CARON et collab., 2018] : Les auteurs proposent une méthode d'entraînement de l'encodeur simple qui consiste à alterner deux phases. Dans la première, un clustering avec la méthode KMEANS est effectué sur le jeu d'entraînement dans l'espace latent afin d'obtenir une affectation pour chaque objet. Ensuite, dans la seconde phase, les paramètres de l'encodeur sont optimisés (avec l'entropie croisée) pour prédire ces pseudo-étiquettes pendant un petit nombre d'itérations. Les deux phases sont ensuite répétées sur plusieurs itérations. Toutefois, les auteurs n'ont utilisé cette méthode que comme méthode de pré-entraînement pour initialiser les poids d'un modèle de classification supervisée.

— Deep Embedded Regularized Clustering (*DEPICT*) [GHASEDI DIZAJI et collab., 2017] : La méthode *DEPICT* est similaire à celle d'*IDEC*, mais elle apporte des modifications supplémentaires à la méthode du *DEC*. Tout d'abord, cette méthode utilise une fonction de coût similaire à *DEC* et incorpore également une couche de regroupement composée d'une couche dense de taille $k$, mais cette fois elle est suivie d'une couche de softmax. En conséquence, la distribution $Q$ est définie comme :

$$q_{ij} = Q(y_i = j|z_i, \Theta_{soft}) = \frac{exp(\theta^T_{soft,k} z_i)}{\sum_{j=1}^{k} exp(\theta^T_{soft,k} z_i)}, \tag{5.28}$$

où $\Theta_{soft} = [\theta_{soft,1}, .., \theta_{soft,k}]$ sont les poids de la couche de clustering. $P$ est calculé de manière similaire à *DEC* par l'équation 5.25. Ensuite, la méthode utilise la KL divergence entre $P$ et $Q$ pour entraîner l'encodeur et ajoute également un terme de régularisation pour éviter d'obtenir une solution dégénérée avec seulement quelques grands clusters :

$$\begin{aligned} L &= KL(Q|P) + KL(f|u) \\ &= [\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{k} p_{ij} \log \frac{p_{ij}}{q_{ij}}] + [\frac{1}{N}\sum_{j=1}^{k} f_j \log \frac{f_j}{u_j}] \\ &= \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{k} p_{ij} \log \frac{p_{ij}}{q_{ij}} + p_{ij} \log \frac{f_j}{u_j}, \end{aligned} \tag{5.29}$$

où $f_j = \frac{1}{N}\sum_{i=1}^{N} q_{ij}$ est la distribution empirique des clusters, c'est-à-dire la fréquence d'apparition des clusters, et $u_j$ est une distribution uniforme. Cela implique l'hypothèse forte que les clusters sont également représentés dans l'ensemble d'apprentissage. Ensuite, ils montrent que cette régularisation peut être approximée par le calcul de l'entropie croisée standard entre $Q$ et $P$.

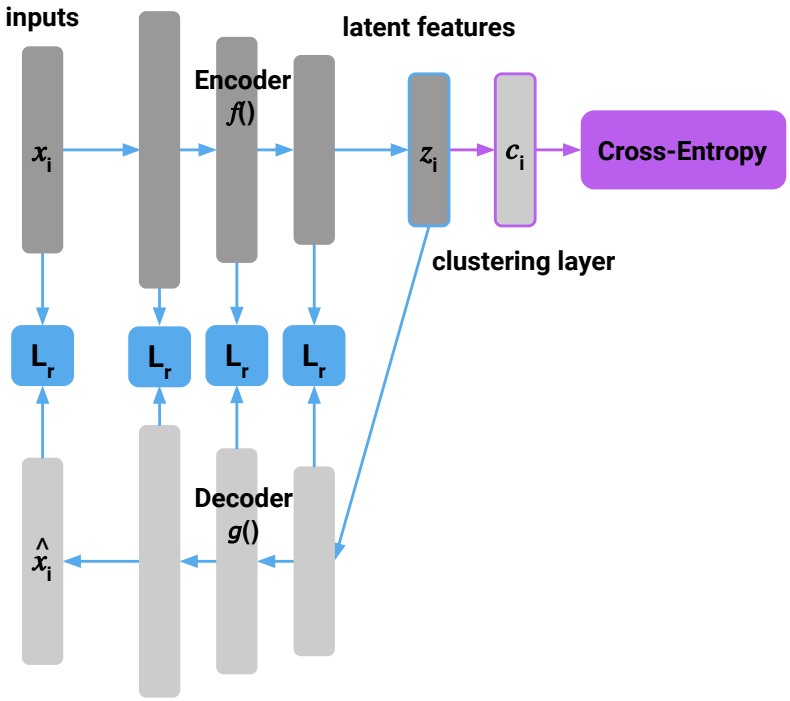



FIGURE 5.8 – La méthode *DEPICT* : L'AE est entraîné à la fois pour reconstruire les données à chaque profondeur de couche et pour obtenir une meilleure confiance dans la prédiction des pseudo-étiquettes des clusters avec l'entropie croisée.

Deuxièmement, ils ont également modifié l'AE pour utiliser une version de débruitage, avec un bruit de masquage, sur chaque couche (c'est-à-dire avec des couches de *Dropout*). Après une phase de pré-entraînement, l'AE est entraîné conjointement avec la fonction de coût de clustering comme dans la méthode *IDEC*. De plus, ils étendent la fonction de coût de reconstruction classique pour qu'elle soit calculée comme la somme des reconstructions à chaque profondeur de l'AE, par :

$$L_{multi\_rec} = \frac{1}{n}\sum_{i=1}^{n}\sum_{l=0}^{L-1}\frac{1}{|z_i^l|}(z_i^l - \hat{z}_i^l)^2 \tag{5.30}$$

où $z_i^l$ est la sortie de la $l^{\text{ème}}$ couche de l'encodeur (la série temporelle en entrée pour $l = 0$), $|z_i^l|$ sa taille de sortie, et $\hat{z}_i^l$ la sortie de la $l^{\text{ème}}$ couche du décodeur en partant de la fin. Cette fonction de coût permet de s'assurer que la reconstruction est conservée à toutes les étapes d'apprentissage de l'autoencodeur.

— Joint Unsupervised LEearning (JULE) [YANG et collab., 2019] : Cette approche est très différente des autres, car elle n'utilise pas de fonction de coût de clustering séparée. Elle est tout de même incluse dans cette section, car elle est souvent mentionnée dans la littérature. C'est une des premières méthodes à avoir obtenu de bons résultats en clustering par apprentissage profond. Cette méthode est une approche de clustering agglomératif, où à chaque étape deux clusters sont fusionnés jusqu'à atteindre le nombre de clusters désiré. Cette fusion se fait par rapport à une matrice d'affinité calculée dans l'espace latent. Ce choix est pondéré par la prise en compte de la structure locale des

données, pour favoriser la fusion des couples de clusters qui sont, à la fois, proches l'un de l'autre mais éloignés des autres clusters voisins.

L'encodeur $f$ est mis à jour tous les $p$ étapes par la fonction de coût suivante :

$$L_{JULE} = -\frac{1}{K_c - 1} \sum_{i,j,k} (\gamma \mathcal{A}(f(x_i), f(x_j)) - \mathcal{A}(f(x_i), f(x_k))), \tag{5.31}$$

où $\gamma$ est un poids défini en paramètre, $\mathcal{A}()$ est la mesure d'affinité entre deux objets. $x_i$ et $x_j$ sont issus du même cluster, tandis que $x_k$ est issu des $K_c$ clusters voisins les plus proches. Le poids de la matrice d'affinité $W$ du sommet $x_i$ au sommet $x_j$ défini par :

$$W(i,j) = \begin{cases} exp(-\frac{\|f(x_i) - f(x_j)\|_2^2}{\delta}), & \text{si } x_i \in \mathcal{N}_i^{K_s} \\ 0, & \text{autrement} \end{cases} \tag{5.32}$$

où $\mathcal{N}_i^{K_s}$ est l'ensemble des plus proches voisins de $K_s$ $x_i$ et $\delta$ l'erreur quadratique moyenne entre $x_i$ et son voisinage $\mathcal{N}_i^{K_s}$.

**Les méthodes de clustering par apprentissage profond pour les séries temporelles**

Certaines méthodes de clustering profond ont également été proposées dans le contexte spécifique des séries temporelles :

— Deep Temporal Clustering ($DTC$) [Madiraju et collab., 2018] : Cette méthode est organisée de la même manière que la méthode *IDEC*. Cependant, les auteurs ont modifié l'architecture de l'encodeur en le remplaçant par une couche de convolution 1D suivie d'une couche de MaxPooling et de deux couches bi-LSTM empilées. Le décodeur consiste en un simple sur-échantillonnage suivi d'une couche de conversion 1D pour reconstruire les données. De plus, au lieu d'utiliser l'état de la dernière couche bi-LSTM pour générer l'espace latent, le modèle utilise la séquence reconstruite par la dernière couche bi-LSTM. Ils conservent donc la dimension temporelle dans l'espace latent. Sur cette base, ils ont modifié le calcul de la distribution de *DEC* $Q$ de l'équation 5.24 par :

$$q_{ij} = \frac{(1 + sim(z_i, \mu_j)/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{j=1}^{k} (1 + sim(z_i, \mu_j)/\alpha)^{-\frac{\alpha+1}{2}}}, \tag{5.33}$$

où $sim(x_i, x_j)$ est une mesure de similarité entre les objets $x_i$ et $x_j$. Ils ont utilisé différentes mesures de similarité telles que la distance euclidienne ou la Distance Invariante de Complexité, cette dernière donnant les meilleurs résultats.

— Deep Temporal Clustering Representation ($DTCR$) Ma et collab. [2019] : Comme pour $DTC$, les auteurs ont utilisé des réseaux récurrents comme base de l'AE. L'encodeur est composé de l'empilement de trois couches d'RNN bidirectionnels. Cependant, ils ajoutent également un paramètre de dilatation exponentielle à ces couches. Le décodeur est une couche RNN unique, son état caché est initialisé par la concaténation de l'état caché final des couches de l'encodeur. Pour l'objectif d'apprentissage, les auteurs ont proposé une nouvelle fonction de coût composée de trois parties :

  — Une fonction de coût de reconstruction classique $L_r$.

  — Une fonction de coût réelle/fausse $L_{classif}$ : l'encodeur est entraîné à discriminer si la série temporelle est réelle ou fausse, les échantillons faux étant générés en mélangeant aléatoirement 20 % des pas de temps. À cette fin, une couche de softmax à deux dimensions est attachée à la fin de l'encodeur et entraînée à prédire un vecteur *one-hot* de taille deux indiquant s'il s'agit d'un vrai ou d'un faux exemple en utilisant l'entropie croisée catégorielle pour l'entraînement du réseau.

— KMEANS $L_{K-Means}$ : cette fonction de coût est basée sur la relaxation spectrale pour le clustering KMEANS proposée dans ZHA et collab. [2002]. Elle consiste à minimiser la fonction suivante :

$$\min_{F} Trace(H^T H) - Trace(F^T H^T H F), s.t. F^T F = I, \tag{5.34}$$

où $H \in \mathbb{R}^{m \times N}$ est la matrice de données, avec $m$ la dimension de l'espace latent. Et $F \in \mathbb{R}^{N \times k}$ est la matrice de l'indicateur de cluster (l'appartenance de chaque objet à chaque cluster, 1 indiquant une appartenance au cluster, 0 une non-appartenance). F est fixé lors de l'apprentissage de l'AE, mais elle est mise à jour toutes les 10 itérations en calculant la décomposition en $k$ valeurs singulières tronquées de $H$.

La fonction de coût combinée est alors définie par :

$$L_{DTCR} = L_r + L_{classif} + \lambda L_{K-Means}, \tag{5.35}$$

où $\lambda$ est un coefficient de régularisation. On peut remarquer que cette fonction de coût n'est pas spécifique aux séries temporelles et pourrait être utilisée pour d'autres types de données.

## 5.2 Évaluation comparative

Toutes les méthodes présentées dans la section 5.1.3 ont leur propre cadre, avec leur propre **architecture** (les types et nombre de couches, l'optimiseur, etc.), leur propre **fonction de coût de clustering** et souvent leur propre **fonction de coût de prétexte**. Ainsi, la comparaison est rendue difficile à faire, car nous ne pouvons déterminer exactement le rôle de chaque élément ou de la combinaison d'éléments sur les performances d'une méthode particulière. Les articles présentant ces méthodes proposent généralement une étude d'ablation, qui consiste à retirer une composante de la fonction de coût ou un pré-traitement des donnés particulier, pour évaluer l'impact de la nouvelle proposition sur le résultat du nouveau modèle [GUO et collab., 2017a; MA et collab., 2019]. Cependant, ces études sont souvent limitées, et ne concernent qu'une composante particulière de la méthode. Or, le nombre de paramètres et de variations possibles est souvent très élevé. En outre, la comparaison d'une méthode générique capable de traiter divers types de données avec une méthode adaptée à un ensemble spécifique de données peut être considérée comme injuste, surtout lorsque la méthode générique peut être facilement adaptée. Il nous a donc paru important de mieux analyser l'impact des principaux choix possibles, afin de mieux comprendre le rôle de chaque élément sur les performances du modèle final.

L'étude présentée dans cette section vise à couvrir les différents éléments disponibles pour le clustering de séries temporelles en apprentissage profond et à mettre en évidence l'influence de chaque élément sur les performances du clustering. Cependant, étant donné le nombre élevé d'approches proposées dans la littérature, nous n'avons étudié qu'une sélection de méthodes. Dans un premier temps nous expliquons comment nous avons décomposé l'analyse des méthodes de clustering et nous donnons plus de détails sur les éléments sélectionnés (section 5.2.1). Dans un second temps, nous présentons le protocole d'évaluation (section 5.2.3), puis nous présentons les résultats de notre évaluation (section 5.2.4).

### 5.2.1 Les méthodes comparées

#### Décomposition des méthodes de clustering par apprentissage profond

Comme nous l'avons présenté dans la section présédente (section 5.1), les méthodes de clustering par apprentissage profond sont généralement décomposées en deux phases. Une phase de pré-entraînement, où le réseau est entraîné à retenir les caractéristiques significatives

TABLEAU 5.1 – Récapitulatif de la compatibilité des fonctions de coût et des architectures. Pour chaque fonction de coût (f.c.) de prétexte et de clustering, nous listons les types d'architecture compatibles. * : indique que c'est à l'exclusion de l'architecture Dilated-RNN

| | | f.c. clustering | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | DEPICT | SDCN | DTCR | DEC | IDEC | ClusterGAN | VADE |
| f.c. prétexte | Multi_rec | ✓FCNN<br>✓CNN<br>✗RNN | ✓FCNN<br>✓CNN<br>✗RNN | ✓FCNN<br>✓CNN<br>✗RNN | ✓FCNN<br>✓CNN<br>✗RNN | ✓FCNN<br>✓CNN<br>✗RNN | ✗All | ✗All |
| | Reconstr. | ✓All | ✓FCNN<br>✓CNN<br>✗RNN | ✓All | ✓All | ✓All | ✗All | ✗All |
| | Triplet | ✓All | ✓FCNN<br>✓CNN<br>✗RNN | ✓All | ✓All | ✓All | ✗All | ✗All |
| | VAE | ✓All | ✓FCNN<br>✓CNN<br>✗RNN | ✓All | ✓All | ✓All | ✗All | ✓All* |
| | GAN | ✗All | ✗All | ✗All | ✗All | ✗All | ✓All* | ✗All |

avec une fonction de coût de prétexte. Et une phase de clustering, où le réseau est entraîné à produire des caractéristiques adaptées à la tâche de clustering. Par conséquent, les réseaux de neurones (DNNs) pour le clustering peuvent souvent être résumés à la combinaison de trois éléments : une architecture (c'est-à-dire l'ensemble des couches dans cette thèse), une fonction de coût de clustering et une fonction de coût de prétexte.

Par exemple, la méthode *DEC* présentée dans XIE et collab. [2016] peut être décomposée comme suit :

1. Un auto-encodeur FCNN comme architecture du DNN.
2. La fonction de coût de reconstruction comme fonction d'optimisation dans la phase de pré-entraînement.
3. La fonction de coût de clustering basée sur la KL divergence comme fonction d'optimisation dans la phase de clustering.

Dans le cadre de cette étude, nous avons donc testé un ensemble d'éléments séparément, en testant les modèles résultant des différentes combinaisons possibles. La liste de toutes les combinaisons utilisées est résumée dans le Tableau 5.1.

### Architectures

Comme expliqué dans la section 5.1.1, trois familles d'architecture de DNNs peuvent être utilisées pour les séries temporelles : les réseaux de neurones entièrement connectés (FCNN), les réseaux de neurones convolutifs (CNN) et les réseaux de neurones récurrents (RNN). Cependant, différentes configurations peuvent être réalisées pour chaque type en fonction de nombreux hyperparamètres, comme le nombre de couches, la taille de chaque couche (nombre de neurones pour les FCNN, nombre de filtres pour les CNN et nombre de cellules pour les RNN), l'ajout de couches spécifiques (par exemple, les couches de *Pooling* pour les CNN, l'utilisation de couches bidirectionnelles pour les RNN). Par conséquent, nous avons décidé de suivre les configurations utilisées dans des articles existants pour nos expériences, comme détaillé ci-dessous. Par défaut, nous avons fixé la taille de la couche d'intégration à $D_{ls} = 320$, l'effet de ce paramètre sera étudié dans la section 5.2.3.

**FCNN :** pour ce type de couches, nous avons utilisé la configuration proposée dans les articles des méthodes *DEC* et *IDEC* [GUO et collab., 2017a; XIE et collab., 2016]. L'encodeur

est composé de trois couches FCNN de dimensions $d/500/500/2000/D_{ls}$, où $d$ est la dimension de l'espace de données. Le décodeur est construit comme un miroir de l'architecture de l'encodeur, à l'exclusion de la couche d'intégration.

**CNN :** pour les réseaux convolutifs, nous avons utilisé trois configurations :

— *ResNet* : Cette architecture a été proposée dans Wang et collab. [2017]. Les DNNs résiduels utilisent des connexions supplémentaires entre les différentes couches du DNN, appelées *skip connections*, qui permettent au réseau d'éviter une ou plusieurs couches (ce qui constitue un bloc résiduel). Les *skip connections* sont principalement utilisée dans le but d'éviter le problème de disparation du gradient lorsque le nombre de couches augmente [He et collab., 2016].

L'implémentation de l'encodeur utilisée est composée de trois blocs résiduels suivis d'une couche de *Global Average Pooling* et d'une couche FCNN comme couche latente. La couche de *Global Average Pooling* consiste à calculer et à créer un vecteur composé des moyennes de chacun des $m$ filtres de la sortie de la dernière couche de convolution $out_l \in \mathbb{R}^{T \times m}$, ce qui peut se résumer à la formule suivante :

$$out_{GAP} = \frac{1}{T} \sum_{t=1}^{T} out_l[t] \tag{5.36}$$

Pour l'architecture *ResNet*, chaque bloc résiduel est composé de trois couches convolutionnelles avec un nombre de filtres identique fixé à 64. La taille des filtres est fixée à 8, 5 et 3 respectivement pour la première, la deuxième et la troisième couche de convolution. Une fonction d'activation ReLU, précédée d'une opération de normalisation par batch, est ensuite ajoutée à la fin du bloc. Enfin, une couche FCNN, comme couche latente, est ajoutée à la fin et a pour taille $D_{ls}$.

— Simple-CNN (*SCNN*) : Il s'agit d'une version simplifiée de l'architecture *ResNet* composée d'un seul bloc résiduel, sans la *skip connection*. Cette configuration permet d'évaluer s'il est justifié d'utiliser l'architecture *ResNet* qui possède beaucoup de paramètres et qui est donc plus coûteuse à entraîner.

— Dilated-CNN (*DCNN*) : Cette architecture a été proposée dans Franceschi et collab. [2019]. Elle utilise deux hyperparamètres particuliers, le padding causal et les convolutions dilatées exponentielles.

L'encodeur est composé d'un ensemble de couches convolutionnelles causales dilatées suivies de la couche FCNN comme couche latente. Le nombre de filtres est fixé à 40 pour toutes les couches avec une taille de filtre de 3. Dans la version originale, le nombre de couches était arbitrairement fixé à 10 avec un facteur exponentiel pour un taux de dilatation de 2. Cependant, nous avons modifié ce paramètre en calculant le nombre de couches et le taux de dilatation avec la fonction décrite dans l'algorithme 4. A noter que cette modification a donné de meilleurs résultats expérimentaux et est plus rapide à calculer. Enfin, une couche FCNN de taille $D_{ls}$ est ajoutée à la fin comme couche latente. Cette architecture est basée sur le code des auteurs [1].

Pour ces trois architectures, le décodeur est construit en miroir de l'architecture de l'encodeur, à l'exclusion de la couche latente.

*RNN* **:** pour les réseaux récurrents, nous avons utilisé trois configurations :

— Deep Temporal Clustering (*DTC*) : Cette architecture a été proposée dans Madiraju et collab. [2018]. L'encodeur est composé d'une couche convolutive suivie d'une couche de *Max Pooling* de taille 10 pour réduire le nombre de pas de temps, notamment pour les longues séries. Ensuite, la sortie est envoyée à deux couches LSTM bidirectionnelles

1. https://github.com/White-Link/UnsupervisedScalableRepresentationLearningTimeSeries

**Algorithm 4** Calcule de la taille de dilation d'une couche

```
procedure (Longueur de la série temporelle ts_length)
    last_dilation = 1
    dilation_list = []
    if ts_length > 50 then
        rate = 2
    else
        rate = 4
    while last_dilation < ts_length/2 do
        last_dilation* = rate
        dilation_list+ = last_dilation
    return dilation_list
```

empilées (Bi-LSTM) de taille 50. La séquence des états cachés du Bi-LSTM est alors utilisée comme espace latent. Ceci est motivé par le maintien de la dimension temporelle dans l'espace latent. Par conséquent, la taille de l'espace latent n'est pas fixe, car elle dépend des dimensions des séries en entrée.

Le décodeur est composé d'une seule couche de déconvolution (une couche d'*Upsampling* suivie d'une couche de convolution) avec un noyau de taille 10 et un nombre de filtres égal au nombre de caractéristiques de la série temporelle d'entrée.

— LSTM bidirectionnel (*BLSTM*) : Il s'agit d'une architecture simple composée de deux couches Bi-LSTM empilées. La première couche a une taille fixe de 50 et la seconde de $\lfloor D_{ls}/2 \rfloor$. Pour l'espace latent, nous utilisons l'état caché final de la dernière couche Bi-LSTM. Le décodeur est construit comme le miroir de l'architecture de l'encodeur.

— GRU Bidirectionnel (*BGRU*) : Cette architecture est identique à l'architecture *BLSTM* et utilise des cellules GRU à la place des cellules LSTM.

— Dilated-RNN (*DRNN*) : Cette architecture a été proposée dans Ma et collab. [2019]. Elle est composée de trois encodeurs RNN dilatés bidirectionnels empilés, avec respectivement un taux de dilatation de 1, 4 et 16. Ce modèle utilise les cellules GRU dans les couches du réseau. Dans l'article, le nombre d'unités de chaque couche est soit 100/50/50, soit 50/30/30. Cependant, comme aucune indication n'est donnée sur la manière de faire ce choix, nous l'avons fixé à 100-50-50 pour tous les jeux de données, cette configuration donnant de meilleurs résultats en moyenne. L'état caché final de la dernière couche (de taille $50 \times 2 = 100$) est utilisé comme espace latent.

Le décodeur est composé d'une seule couche RNN avec des unités GRU de taille $(100 + 50 + 50) \times 2 = 400$. Son état caché est initialisé par la concaténation des états cachés finaux de toutes les couches de l'encodeur. Ensuite, le décodeur prédit itérativement la séquence reconstruite à partir de la sortie à $t - 1$, avec la sortie au temps $t = 0$ est un vecteur nul. Cette architecture est basée sur le code des auteurs [2].

### Les fonctions de coût de prétexte

Pour les fonctions de coût de prétexte, nous utilisons quatre méthodes :

— La fonction de coût de reconstruction (*rec*) : Il s'agit de la fonction de coût de reconstruction de l'autoencodeur classique qui consiste à calculer l'erreur quadratique moyenne entre la séquence en entrée et sa reconstruction (voir équation 5.16).

— La fonction de coût de reconstruction de *DEPICT* (*multi_rec*) : Cette fonction de coût de prétexte est celle utilisée dans Ghasedi Dizaji et collab. [2017]. Elle étend la fonction de coût de reconstruction en calculant l'erreur quadratique moyenne entre chaque

2. https://github.com/qianlima-lab/DTCR

couche de l'encodeur et la couche correspondante du décodeur (voir équation 5.30). On peut remarquer que cette fonction de coût nécessite que le décodeur soit construit en miroir de l'encodeur, elle exclut toute architecture RNN dans notre cas.

— La fonction de coût des VAE (*vae*) : Il s'agit de la fonction de coût des VAE classique qui contient la reconstruction et la normalisation de la distribution de l'espace latent (voir équation 5.19) :

— La fonction de coût de triplet (*triplet*) : Il s'agit de la fonction de coût proposée dans Franceschi et collab. [2019] qui vise à obtenir une représentation similaire entre une sous-série temporelle et son voisinage (voir équation 5.23). Pour cette fonction de coût, nous avons utilisé quatre valeurs de $K_{triplet}$, 1, 2, 5, et 10. Nous avons également calculé le résultat avec la version combinée (la concaténation de la représentation sur les quatre différentes valeurs de $K_{triplet}$). L'implémentation est basée sur le code des auteurs [3].

Nous avons également utilisé la fonction de coût de prétexte du GAN, mais comme le GAN n'implique pas d'encodeur, il ne peut être utilisé qu'avec des modèles spécifiquement conçus pour le clustering. Elle n'est donc utilisée qu'en combinaison avec la fonction de coût de clustering *ClusterGAN* (voir section suivante).

#### Les fonctions de coût de clustering

De nombreuses fonctions de coût ont été proposées pour simplifier la tâche de clustering dans l'espace latent, notamment pour les images [Caron et collab., 2018; Ghasedi Dizaji et collab., 2017; Guo et collab., 2017b; Xie et collab., 2016; Yang et collab., 2019]. Nous en avons sélectionné un sous-ensemble couvrant les différents types d'approches. Nous avons privilégié celles dont le code était disponible afin de nous assurer de la validité de nos implémentations. Les fonctions de coût sélectionnées sont présentées ci-dessous, mais ont déjà été expliquées plus en détail dans la section 5.1.3 :

— *DEC* [Xie et collab., 2016] et *IDEC* [Guo et collab., 2017a] : La fonction de coût *DEC* utilise la divergence KL pour améliorer la confiance de l'affectation d'un objet à son cluster. La fonction de coût *IDEC* étend cette dernière en conservant la fonction de coût de prétexte (la fonction de coût de reconstruction dans l'article) dans la phase de clustering (voir équation 5.27). Pour le paramètre $\gamma$, nous avons gardé la valeur par défaut proposée dans [Guo et collab., 2017a] de 0,1. L'implémentation est basée sur le code des auteurs [4].

— *DEPICT* [Ghasedi Dizaji et collab., 2017] : Cette fonction de coût est similaire à la fonction de coût *IDEC*. Cependant, les auteurs utilisent l'entropie croisée standard pour entraîner les paramètres du modèle. Nous utilisons également une valeur $\gamma$ de 0,1.

— *SDCN* [Bo et collab., 2020] : Cette fonction de coût est également basée sur la fonction de coût *IDEC*. Les auteurs utilisent des réseaux convolutifs de graphes pour régulariser l'espace latent appris afin de conserver la structure locale des données en se basant sur un graphe KNN. Nous avons utilisé un nombre de voisins égal à 3 vu la petite taille de certains jeux de données. Nous avons également modifié l'algorithme KNN pour utiliser la mesure DTW au lieu de la distance euclidienne, car DTW donne de meilleurs résultats sur l'archive UCR [Dau et collab., 2019]. L'implémentation est basée sur le code des auteurs [5].

— *VADE* [Jiang et collab., 2016] : Cette fonction de coût est basée sur la fonction de coût ELBO et est donc compatible uniquement avec les VAEs. Elle étend la fonction de coût ELBO en générant des exemples d'entraînement dans $K$ (le nombre de clusters

3. https://github.com/White-Link/UnsupervisedScalableRepresentationLearningTimeSeries
4. https://github.com/XifengGuo/IDEC
5. https://github.com/bdy9527/SDCN

attendus) distributions différentes au lieu d'une seule. L'implémentation est basée sur le code des auteurs [6].

— *ClusterGAN* [Ghasedi et collab., 2019] : Cette fonction de coût est basée sur la fonction de coût des GAN, et n'est donc compatible qu'avec ce type de modèles. Elle consiste en l'ajout d'un encodeur qui est entraîné à reproduire la représentation latente à partir des séries temporelles générées. L'implémentation est basée sur le code des auteurs [7].

— *DTCR* [Ma et collab., 2019] : Cette fonction de coût est la seule proposée spécifiquement pour les séries temporelles, même si elle est générique. Elle combine trois composants, la fonction de coût de prétexte, une fonction de coût K-Means (voir équation 5.34), et la fonction de coût vrai/faux. Nous utilisons une valeur $\lambda$ de $0,5$. L'implémentation est basée sur le code des auteurs [8].

— None : Nous évaluons également le résultat obtenu sans utiliser de fonction de coût de clustering.

Pour toutes les configurations, nous utilisons l'optimiseur Adam avec un taux d'apprentissage de 0.001 [Bo et collab., 2020; Franceschi et collab., 2019; Madiraju et collab., 2018; Mukherjee et collab., 2019] à l'exception de l'architecture *DRNN* où nous utilisons l'optimiseur SGD avec une décroissance exponentielle, un taux d'apprentissage de 0.1, et un taux de décroissance de 0.1 pour suivre les préconisations des auteurs. La taille de *batch* (nombre de séries temporelles fournies au modèle pour chaque descente de gradient) est fixée à 10 pour toutes les configurations à l'exception des fonctions de coût *SDCN* et *DTCR* qui nécessitent une taille de *batch* égale à la taille de l'ensemble du jeu d'entraînement.

### 5.2.2 Protocole d'évaluation

#### Les archives UCR et UEA

Pour valider nos résultats, nous avons utilisé deux ensembles de jeux de données, l'archive univariée de l'University of California Riverside (UCR) [Dau et collab., 2019] et l'archive multivariée de l'University of East Anglia (UEA) [Bagnall et collab., 2018]. Bien que ces archives aient été conçues pour évaluer les méthodes de classification supervisée, aucun ensemble de jeux de données n'est encore disponible pour évaluer spécifiquement les méthodes de clustering. De plus, ces deux archives sont déjà souvent utilisées dans le contexte du clustering de séries temporelles [Ma et collab., 2019; Madiraju et collab., 2018; Paparrizos et Gravano, 2015].

Nous avons utilisé la version étendue de l'archive de l'UCR avec 128 jeux de données et l'archive de l'UEA avec 30 jeux de données, toutes deux disponibles en ligne. Les jeux de données, provenant de différents domaines, sont regroupés en différentes catégories, les principales étant les contours d'images, les enregistrements de capteurs, la capture de mouvements, la spectrographie, les électrocardiogrammes, les dispositifs électriques, les données audio et les données simulées. Tous les jeux de données sont divisés en jeux d'entraînement et de test. Le nombre de séquences dans les jeux d'entraînement va de 3000 à 12 séquences et de 20000 à 15 dans les jeux de test. Certains jeux de données ont des séries temporelles de longueurs différentes, mais nous utilisons les versions avec des longueurs égalisées fournies dans l'archive. Dans cette version, toutes les séries sont complétées par des zéros à la fin pour atteindre la longueur de la série la plus longue. La longueur des séries varie de 3000 à 2 mais avec une longueur médiane de 218. De plus, pour chaque jeu de données, un jeu de données de référence est fourni. Le nombre de classes par jeu de données varie de 2 à 60, avec une médiane à 4 (49 d'entre elles n'ont que 2 classes). Pour les jeux de données multivariées, le nombre de caractéristiques varie entre 2 et 1345. Auparavant, tous les jeux de données UCR

6. https://github.com/slim1017/VaDE
7. https://github.com/sudiptodip15/ClusterGAN
8. https://github.com/qianlima-lab/DTCR

étaient z-normalisés, mais certains sont maintenant fournis sans aucun prétraitement. Pour simplifier l'évaluation, nous avons décidé d'effectuer une z-normalisation sur tous les jeux de données. La z-normalisation est calculée individuellement pour chaque série telle que pour une série $x$ de longueur $l$ on obtient la version normalisée $x_z$ par :

$$x_z = \frac{x - \mu(x)}{std(x)} \tag{5.37}$$

où $\mu(x)$ est la valeur moyenne de la série sur chaque caractéristique et $std(x)$ l'écart-type.

En plus d'offrir une variété de jeux de données, ces deux archives ont fait l'objet de nombreuses utilisations [Fawaz et collab., 2019; Franceschi et collab., 2019; Ma et collab., 2019; Madiraju et collab., 2018; Xiao et collab., 2020; Zhang et collab., 2018]. Elle permet ainsi une meilleure comparaison avec les autres méthodes que ce soit de classification ou de clustering.

### Méthodologie d'évaluation

Toutes les combinaisons sont entraînées avec 1000 itérations par batch pour la phase d'entraînement et 1000 pour la phase de clustering. Après la phase d'apprentissage, un premier modèle est obtenu (désigné par la fonction de coût de clustering $None$). Ce modèle est ensuite utilisé comme initialisation pour toutes les autres fonctions de coût de clustering, à l'exception de *ClusterGAN* et *VADE* qui sont entraînés à partir de zéro, ces deux méthodes n'ayant pas de phase de pré-entraînement.

Les modèles sont entraînés (phase d'entraînement + phase de clustering) sur le jeu d'entraînement, et le clustering est effectué sur le jeu de test, ce qui suit le choix fait dans Ma et collab. [2019] et Xiao et collab. [2020] pour le clustering profond de séries temporelles. Ce protocole permet de s'assurer que l'espace latent appris peut être généralisé. Pour clusteriser le jeu de test, la plupart des modèles de fonction de coût de clustering fournissent directement l'affectation aux clusters. Pour les autres, le clustering est effectué avec un Kmeans sur le jeu de test projeté dans l'espace latent (c'est-à-dire les séries temporelles encodées). Pour évaluer la performance du clustering, nous utilisons l'information mutuelle normalisée (NMI : *Normalized Mutual Information*), car c'est la métrique la plus couramment utilisée pour le clustering profond [Ghasedi Dizaji et collab., 2017; Guo et collab., 2017b; Ma et collab., 2019; Xie et collab., 2016; Yang et collab., 2019; Zhang et collab., 2018] et aussi car elle prend également en considération la distribution attendue contrairement à des mesures telles que l'indice de Rand ou la précision de clustering.
Le score de NMI est calculé entre une partition de $M$ groupes $A = \{A_1, \ldots, A_M\}$ et une partition de $M'$ groupes $B = \{B_1, \ldots, B_{M'}\}$ par la formule suivante :

$$NMI = \frac{\sum_{i=1}^{M}\sum_{j=1}^{M'} N_{ij} log\frac{N.N_{ij}}{|A_i||B_j|}}{\sqrt{(\sum_{i=1}^{M}|A_i| log\frac{|A_i|}{N})(\sum_{j=1}^{M'}|B_j| log\frac{|B_j|}{N})}}, \tag{5.38}$$

où $N_{ij} = |G_i \bigcap A_j|$. La valeur varie entre 0 et 1, une valeur de 1 indiquant que les distributions sont identiques.

Chaque combinaison de DNN est entraînée 5 fois. Nous utilisons la moyenne du score NMI de ces 5 exécutions pour l'évaluation. Ces processus ont été exécutés sur un cluster de calcul de plus de 60 GPU de types GTX 1080Ti, Tesla P100, K20, K40 ou K80. En outre, il convient de mentionner que certaines combinaisons présentent des problèmes de convergence qui ont conduit à la disparition ou à l'explosion des gradients. Ces combinaisons conduisent essentiellement à de très mauvais résultats. Cependant, certaines combinaisons ont été exclues de l'évaluation lorsque trop de jeux de données sans résultat ont été obtenus. Une limite de 10 jeux de données sans résultats pour l'archive univariée et de 3 pour l'archive multivariée a été fixée. Cela ne s'est produit que pour les combinaisons avec l'architecture *FCNN* et les

fonctions de coût de prétexte *triplet* dans l'archive multivariée (une note est ajoutée dans les tableaux de résultat le cas échéant). Tous les résultats détaillés avec le NMI mais aussi l'Adjusted Rand Index (ARI) et la mesure de la Clustering Accuracy sont également rapportés sur notre dépôt git [9].

Enfin, nous voulons évaluer la performance globale de chaque combinaison et déterminer si la différence avec les autres combinaisons est significative ou non. Pour la comparaison, nous utilisons le rang moyen des gains et des pertes. En suivant les recommandations de Dau et collab. [2019], nous utilisons les tests de rang signés de Wilcoxon par paire [Wilcoxon, 1992] et formons des cliques en utilisant la correction de Holm [Holm, 1979] pour déterminer la différence critique entre chaque combinaison avec un niveau significatif de $\alpha = 0.05$. Pour visualiser ces comparaisons, nous utilisons un diagramme de différences critiques proposé par Demšar [2006], où une ligne horizontale épaisse montre la clique calculée précédemment. Cette méthode de comparaison présente certaines limites, car le rang peut ne pas refléter la robustesse globale d'une méthode, surtout lorsque le nombre de jeux de données et de méthodes comparées est élevé. Cependant, dans nos expériences, les résultats ont donné des informations qui étaient corrélées aux résultats individuels observés. Il est également à noter que le classement et le test de Wilcoxon traitent tous deux les valeurs manquantes sans fausser le résultat.

### 5.2.3 Résultats

Dans cette section, nous présentons les résultats obtenus en exécutant toutes les combinaisons valides. Nous présentons d'abord l'évaluation entre toutes les combinaisons dans la section 5.2.3, puis nous montrons l'effet des autres paramètres sur les performances dans la section 5.2.3. Enfin, nous comparons les meilleures combinaisons aux approches standards de clustering sans apprentissage profond dans la section 5.2.3.

#### Comparaison croisée des différentes combinaisons

Afin de mieux comparer les performances de chaque choix, nous comparons chaque élément (architecture, fonction de coût de prétexte et fonction de coût de clustering) séparément. Pour chaque élément, nous choisissons la meilleure combinaison qui l'inclut (par exemple, la meilleure combinaison qui utilise l'architecture *FCNN*) en fonction du rang moyen de gain/perte sur le jeu d'entraînement. Ensuite, nous comparons les meilleurs candidats pour l'architecture, la fonction de coût de prétexte et la fonction de coût de clustering séparément sur le jeu de test.

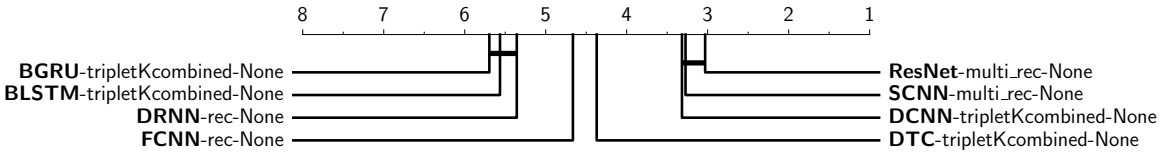
Les résultats de l'archive univariée sont affichés dans les figures 5.9 et 5.10 pour le multivarié. Le meilleur score de NMI moyen est obtenu par la combinaison *ResNet-mulit_rec-None* avec un score de NMI moyen de 0,349 pour l'archive univariée et par la combinaison *DCNN-mulit_rec-SDCN* pour l'archive multivariée avec 0,356 (cette dernière valeur est approximative, car deux jeux de données n'ont pu être clusterisés avec cette combinaison).

Le premier constat qui ressort de ces résultats est qu'aucune méthode ne surpasse les autres avec une différence critique, surtout si l'on tient compte des deux archives. Cependant, les observations suivantes peuvent être faites :

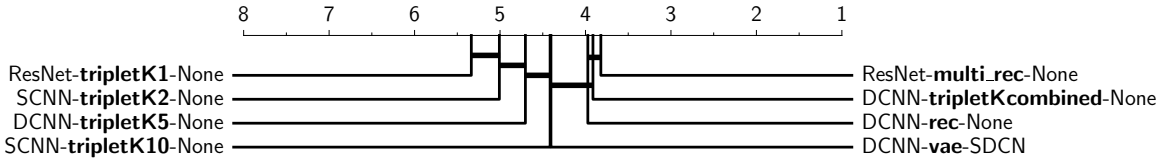
— Sur les architectures : en regardant les résultats univariés, les architectures basées sur des CNN surpassent toutes les autres types. Sur les jeux de données multivariés, l'architecture *DRNN* donne également de bons résultats, mais les architectures *DCNN* et *SCNN* sont également performantes.

— Sur les fonctions de coût de prétexte : les fonctions de coût basées sur la reconstruction (*rec* et *multi_rec*) et les fonctions de coût *triplet* combinée sont celles qui donnent les meilleurs résultats sur les archives univariées. Cependant, la fonction de coût *triplet*

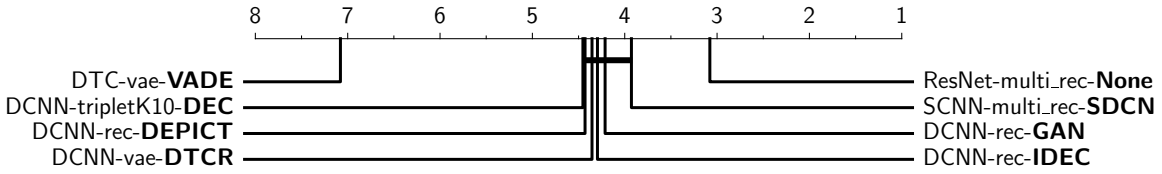
9. https://github.com/blafabregue/TimeSeriesDeepClustering




(a) Meilleures combinaisons pour chaque architecture

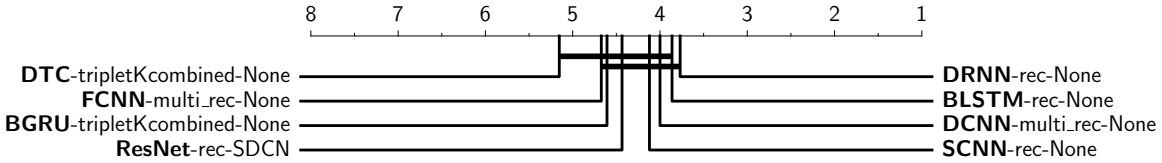



(b) Meilleures combinaisons pour chaque fonction de coût de prétexte

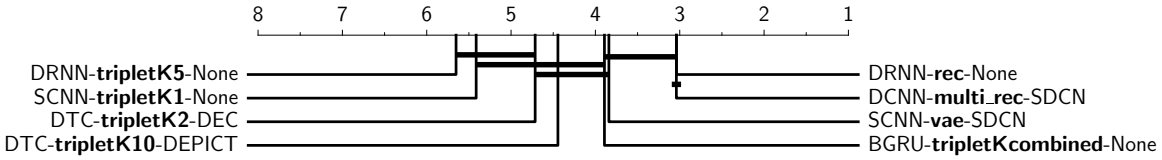



(c) Meilleures combinaisons pour chaque fonction de coût de clustering

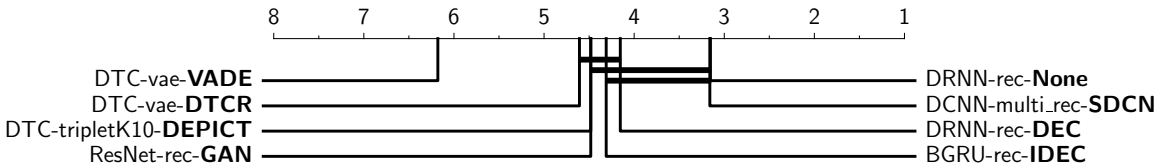
FIGURE 5.9 – Résultats pour les séries temporelles univariées avec la mesure NMI.




(a) Meilleures combinaisons pour chaque architecture




(b) Meilleures combinaisons pour chaque fonction de coût de prétexte*




(c) Meilleures combinaisons pour chaque fonction de coût de clustering

FIGURE 5.10 – Résultats pour les séries temporelles multivariées avec la mesure NMI (* : les fonctions de coût basées sur *FCNN* combiné avec *triplet* ont été exclues, car elles entraînent trop d'erreurs de calcul).

semble ne pas être cohérente sur les archives multivariées. De plus, si nous rapportons le score de NMI moyen sur tous les jeux de données univariées, nous obtenons pour l'architecture *DCNN* respectivement 0,328, 0,329 et 0,339 pour la fonction de coût *triplet_combiné*, *multi_rec* et *rec*. Mais la différence augmente considérablement avec 0,137, 0,343 et 0,339 pour les fonctions de coût multivariées.

— les fonctions de coût de clustering : étonnamment, l'ajout de fonction de coût de clustering n'entraîne pas de gain de performance à l'exception de la fonction de coût *SDCN* pour les archives multivariées. Par conséquent, les résultats obtenus tendent à ne pas justifier le temps de calcul et la complexité supplémentaires requis par l'utilisation d'une fonction de coût de clustering.

On peut également souligner que chaque élément, pris séparément, parvient à obtenir le meilleur score de NMI sur au moins un jeu de données. Par exemple, l'architecture *BGRU* obtient le meilleur score sur le jeu de données CBF univarié (avec 0,71 NMI moyen), même si cette architecture obtient le plus faible rang moyen. Il peut donc être intéressant de rechercher des relations entre certains types d'éléments et leurs performances sur les jeux de données. Malheureusement, l'analyse que nous avons menée n'a conduit à aucune corrélation, plus de détails sont rapportés dans la section 5.2.4.

Enfin, il convient de mentionner que le clustering est souvent considéré comme un problème mal posé, tel qu'explicité dans la section 1.2. D'ailleurs nous y avons évoqué le cas de certains jeux de données de l'archive UCR qui contiennent les mêmes données mais un étiquetage différent (par exemples les jeux de données DodgerLoopGame et DodgerLoopWeekend). Par conséquent, aucune méthode non supervisée ne peut obtenir de bons résultats sur un jeu de données sans avoir de mauvais résultats sur l'autre ou les autres. De plus, l'écart-type observé sur toutes les méthodes est non négligeable. L'écart-type de NMI va de 0,010 à 0,170 avec une médiane à 0,060. Ceci peut être observé avec toutes les combinaisons. Cela peut suggérer que toutes les méthodes ont tendance à tomber dans des minima locaux.

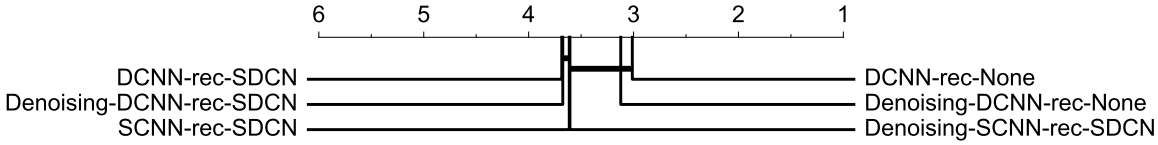
Plus de détails seront donnés dans la section 5.2.4 où nous discuterons de l'évolution du score NMI à travers le processus d'entraînement.

#### Influence des autres paramètres

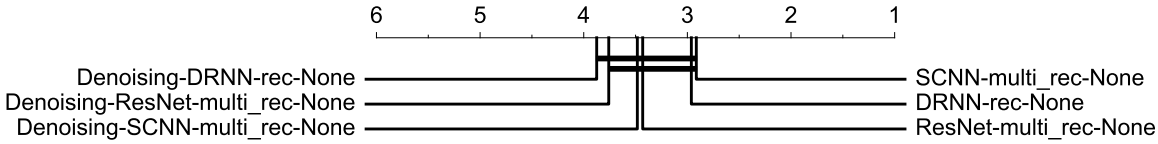
Même si les résultats précédents couvrent différentes variations de DNNs, d'autres paramètres sont utilisés dans la littérature lors de l'entraînement de DNNs pour le clustering. Il est classique en classification supervisée d'ajuster les choix de ces paramètres, par *grid-search* par exemple, notamment pour les hyperparamètres (par exemple la taille et le nombre de couches, l'optimiseur, le taux d'apprentissage, etc). Cependant, il est impossible de mener une telle optimisation pour chaque jeu de données dans un contexte non supervisé, car aucun jeu d'entraînement ne peut être utilisé.

Dans cette section, étant donné le nombre élevé de méthodes comparées, nous avons décidé de rapporter l'effet d'une petite sélection de ces paramètres. Nous avons sélectionné des paramètres ou des traitements supplémentaires qui sont souvent utilisés dans les méthodes de clustering.

**Débruitage :** Les autoencodeurs de débruitage (DAE) sont souvent utilisés comme fonction de coût de prétexte [Ghasedi Dizaji et collab., 2017; Guo et collab., 2017b; Xie et collab., 2016]. (voir section 5.1.2 pour plus de détails). Nous avons utilisé le bruit de masquage car il est le plus utilisé dans les méthodes sélectionnées [Ghasedi Dizaji et collab., 2017; Guo et collab., 2017a; Xie et collab., 2016]. Le bruit de masquage est généré avec des couches de *Dropout*. Ces couches fonctionnent en passant une portion des caractéristiques de la couche précédente à zéro, les caractéristiques modifiées étant sélectionnées au hasard à chaque passage dans la couche. Nous avons fixé le taux de *dropout* à 20%. Notez que cette méthode est conçue pour les autoencodeurs et qu'elle n'est donc applicable qu'aux fonctions de coût *rec* et




(a) Meilleures combinaisons avec débruitage sur l'archive univariée




(b) Meilleures combinaisons avec débruitage sur l'archive multivariée

FIGURE 5.11 – Comparaison des performances des DNNs formés avec débruitage et sans débruitage avec la mesure NMI.

*multi_rec*. Nous avons sélectionné les trois combinaisons les mieux classées avec débruitage et les avons comparées à leur version sans débruitage dans la figure 5.11.

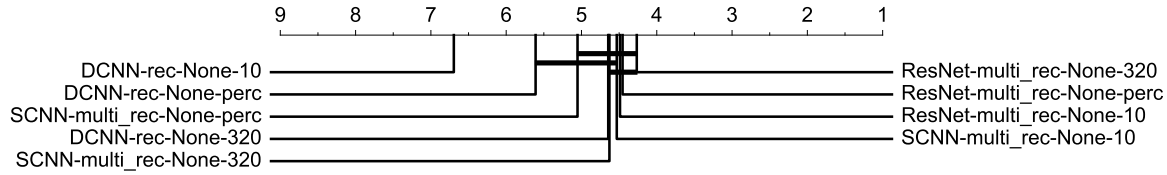
Les résultats rapportés montrent que le débruitage dégrade les résultats ou a un effet très faible sur les performances. En regardant les résultats individuels sur chaque jeu de données, on peut également remarquer que les améliorations, quand elles ont lieu, ont tendance à être très faibles. Mais à l'inverse, cela peut aussi conduire à une dégradation importante des performances. Cela peut s'expliquer par l'effet du bruit sur les séries temporelles, où une forte variation du signal peut plus facilement créer une confusion entre les classes. De plus, pour les DNNs à convolution 1D, l'information sur le voisinage peut être plus limitée que pour les convolutions 2D en raison de la taille des filtres et de leur nature 1D. En effet, nous n'avons que 2 voisins immédiats en 1D pour 8 en 2D. Cela peut altérer la capacité des DNNs à discriminer entre le bruit et le vrai signal.

**Dimension de l'espace latent** Il est souvent recommandé d'avoir un espace latent avec un nombre de dimensions significativement plus petit que celui de l'espace de données original. Ceci a pour but de forcer le DNN à ne retenir que les caractéristiques significatives pour la reconstruction ou la tâche de prétexte sélectionnée. Par conséquent, nous voulons tester l'effet sur les données de séries temporelles.

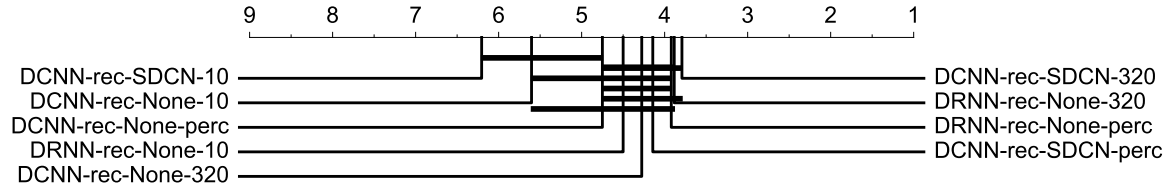
Compte tenu du temps de calcul nécessaire pour lancer toutes les combinaisons, nous n'avons lancé que trois options différentes de taille de clustering. Le nombre de caractéristiques de l'espace latent est fixé à 10, 320 ou « 10% de la longueur de la série temporelle » (noté *perc*). Ainsi, nous pouvons comparer l'effet d'un petit espace latent, d'un grand espace latent ou d'un espace latent adapté au jeu de données. Les résultats sont affichés dans la figure 5.12.

Le choix d'une grande dimension latente semble plus pertinent, notamment pour l'architecture *DCNN-rec-None* sur l'archive univariée et *DCNN-rec-SDCN* sur la multivariée. Pour cette dernière combinaison, elle parvient à obtenir le meilleur classement moyen pour la version « 320 caractéristiques » et le pire avec sa version « 10 caractéristiques » avec une différence significative. La différence entre les versions à 10 et 10% semble plus relative, car la version à 10% donne lieu à une dimension latente médiane de 20. Par conséquent, cette version est davantage proche du choix d'une petite taille de dimension latente.

Cependant, ces observations générales cachent un comportement très variable entre les jeux de données. Dans le cas de l'architecture *DCNN-rec-None* et du jeu de données Chinatown, la version à « 10 caractéristiques » obtient le meilleur score avec 0,69 contre 0,53 pour la taille 320. Ceci peut également être observé pour d'autres jeux de données comme GesturePebbleZ2 ou Trace, avec respectivement un gain de NMI de 0,13 et 0,10. Par conséquent,




(a) Meilleures combinaisons par tailles d'espace latent sur l'archive univariée




(b) Meilleures combinaisons par tailles d'espace latent sur l'archive multivariée

FIGURE 5.12 – Comparaison des performances des DNNs entraînés avec différentes tailles de dimension latente, 10, 320 ou perc (10% de la longueur de la série temporelle) avec la mesure NMI.

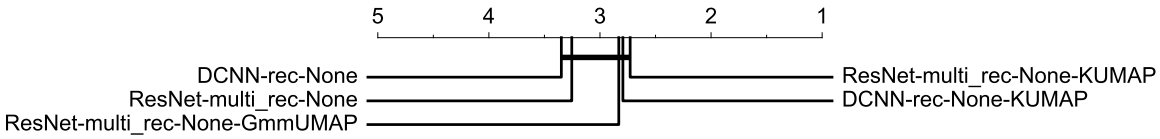
même si une grande taille de dimension latente semble être un bon choix par défaut, elle peut fortement minimiser la performance du DNN dans certains cas.

Cette comparaison limitée souligne déjà que la modification des hyperparamètres peut entraîner une modification importante de la capacité des DNNs à extraire de bonnes caractéristiques. Dans un contexte non supervisé, cela peut fortement limiter l'application de ces méthodes. C'est encore plus problématique si l'on prend en considération le nombre de combinaisons d'hyperparamètres possibles pour les DNNs (nombre de couches, taille et nombre de filtres pour les CNNs, options de taux d'apprentissage, ...).

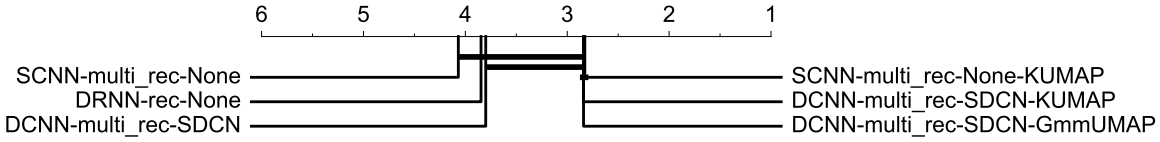
**Réduction de dimension** Cette section est motivée par le travail présenté par McConville et collab. [2019]. Les auteurs proposent d'utiliser une méthode de réduction de dimension sur l'espace latent avant d'appliquer la méthode de clustering. Les méthodes de réduction de dimension proposées sont Isomap [Tenenbaum et collab., 2000], t-SNE [Maaten et Hinton, 2008], Linear Embedding (LLE) [Roweis et Saul, 2000] et UMAP [McInnes et collab., 2018]. Ils fixent le nombre de dimensions de la sortie égale au nombre de clusters recherchés $K$. Ils proposent également de remplacer la méthode Kmeans par des approches de clustering spectral ou de modèle de mélange gaussien (GMM). Notez que dans les résultats rapportés, la combinaison de la méthode Kmeans avec la méthode UMAP est notée comme KUMAP. Nous avons testé les différentes combinaisons et indiqué les trois combinaisons les mieux classées avec la dimension de réduction et les avons comparées à leur version sans réduction de dimension dans la figure 5.13. Nous avons également ajouté la combinaison la mieux classée sans réduction de dimension pour l'archive multivariée (*DRNN-rec-None*) dans les deux diagrammes pour comparaison. Pour l'archive univariée, la combinaison *DRNN-rec-None* est déjà dans le top 3 des versions avec réduction de dimension. Les résultats ont été obtenus en se basant sur le code des auteurs [10].

Dans McConville et collab. [2019], les auteurs ont observé que la méthode de réduction de dimension UMAP, en combinaison avec la méthode de clustering GMM, atteint la meilleure performance. Dans notre cas, UMAP améliore également la performance du clustering. Pour toutes les autres méthodes de réduction de dimension, nous observons une dégradation des résultats. Dans notre cas, Kmeans a donné des résultats légèrement meilleurs que GMM mais sans une différence critique. Cependant, cela tend toujours à confirmer les observations de McConville et collab. [2019]. Il faut également mentionner qu'aucune différence critique

10. https://github.com/rymc/n2d




(a) Meilleures versions de réduction de dimension en comparaison a un simple KMEANS sur l'archive univariée




(b) Meilleures versions de réduction de dimension en comparaison a un simple KMEANS sur l'archive multivariée

FIGURE 5.13 – Comparaison des performances de clustering effectuées soit directement sur la représentation apprise, soit après une réduction de dimension (UMAP) avec différentes méthodes de clustering (GMM et K-Means) avec la mesure NMI.

n'est rapportée entre les résultats avec et sans dimension de réduction avec la correction de Holms. Mais le test de Wilcoxon rapporte une différence avec $p < 0.02$ entre le clustering reportés avec et sans UMAP lorsqu'ils sont comparés individuellement. Sur la moyenne globale du NMI, la combinaison *ResNet-multi_rec-None-KUMAP* obtient 0,399 contre 0,356 sans UMAP. Pour le multivarié, cela passe de 0,417 pour *DRNN-rec-None-KUMAP* à 0,348 sans UMAP. Dans l'ensemble, l'utilisation d'UMAP semble être un outil cohérent pour améliorer la performance du clustering.

### Comparaison aux méthodes sans apprentissage profond

Même si cet article a pour but d'évaluer différentes méthodes de clustering profond les unes par rapport aux autres, nous souhaitons également positionner ces méthodes parmi les autres méthodes classiques de clustering pour les séries temporelles. Pour ce faire, nous avons sélectionné les méthodes suivantes (sans volonté aucune d'exhaustivité) :

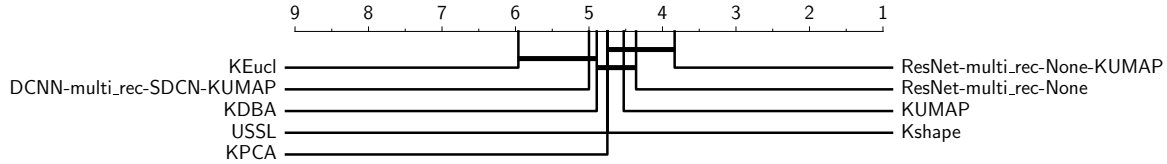
— *KEucl* : La méthode KMEANS avec la distance euclidienne et la moyenne arithmétique pour calculer les centres. Nous utilisons l'implémentation dans `tslearn`[11] avec une itération maximale de 200.

— *KDBA* : La méthode KMEANS avec la mesure DTW (Dynamic Time Warping) [SAKOE et CHIBA, 1978] et DBA (DTW Barycenter Averaging) PETITJEAN et collab. [2011] pour calculer les centres. Nous avons utilisé l'implémentation dans `tslearn` avec une itération maximale de 200.

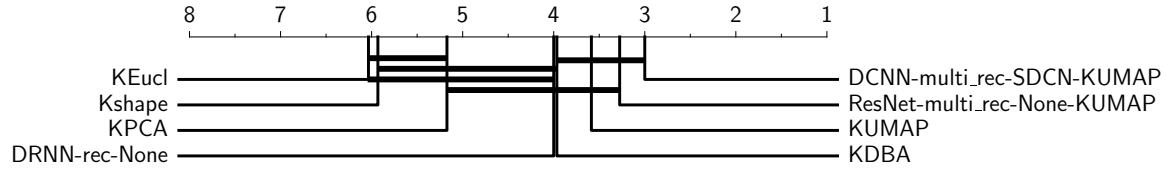
— *KPCA* : La méthode KMEANS avec la distance euclidienne et la moyenne arithmétique pour calculer les centres. Cependant, dans ce cas, nous effectuons une analyse en composantes principales (ACP) avec une réduction à $K$ dimensions avant d'appliquer l'algorithme KMEANS. Nous avons utilisé l'implémentation `tslearn` pour le KMEANS, et `sklearn` pour l'ACP[12].

— *KUMAP* : La méthode KMEANS avec la distance euclidienne et la moyenne arithmétique pour calculer les centres. Cependant, dans ce cas, nous effectuons avec une réduction à $K$ dimensions avec la méthode UMAP avant d'appliquer l'algorithme KMEANS. Nous avons utilisé l'implémentation dans `tslearn` pour KMEANS avec une itération maximale

11. https://tslearn.readthedocs.io/en/stable/gen_modules/clustering/tslearn.clustering.TimeSeriesKMeans.html
12. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html




(a) Comparaison avec les méthodes non-profondes sur l'archive univariée




(b) Comparaison avec les méthodes non-profondes sur l'archive multivariée

FIGURE 5.14 – Résultats des méthodes de clustering par apprentissage profond comparées aux méthodes non-profondes avec la mesure NMI.

de 200, et `umap-learn` pour UMAP [13].

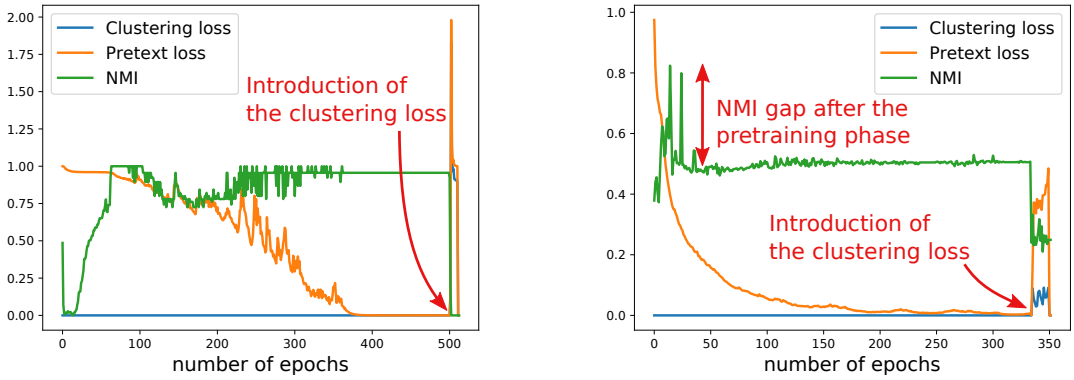
— *Kshape* [PAPARRIZOS et GRAVANO, 2015] : k-shape est une méthode qui s'appuie sur une procédure de raffinement itérative évolutive pour extraire des clusters et qui se base sur la mesure de corrélation croisée. Nous avons utilisé l'implémentation Python de l'auteur [14].

— *USSL* [ZHANG et collab., 2018] : Unsupervised Salient Subsequence Learning est une méthode qui extrait des motifs temporels pour obtenir extraire des caractéristiques représentatives. Pour cette méthode, nous utilisons les résultats rapportés dans les matériaux supplémentaires de MA et collab. [2020] sur la version à 85 jeux de données de l'archive UCR.

La comparaison est présentée dans la figure. 5.14. Pour les deux archives, nous avons sélectionné le meilleur candidat sur la base des résultats NMI sur le jeu d'entraînement avec et sans UMAP. Le diagramme affiché est calculé sur le jeu de test et donne un classement similaire. Nous avons également ajouté le meilleur candidat UMAP de chaque archive dans le diagramme de l'autre archive pour évaluer la robustesse de ces méthodes sur les deux archives. Pour les méthodes *USSL* et *Kshape*, nous n'avons pas rapporté les résultats multivariés, car ils n'étaient pas inclus dans MA et collab. [2020] pour *USSL* et parce que le code des auteurs de *Kshape* ne supporte pas les séries temporelles multivariées.

Pour les deux archives, le meilleur candidat des méthodes de clustering par apprentissage profond avec l'utilisation d'UMAP est classé premier. Il convient également de noter que la méthode *KUMAP* figure parmi les trois premières méthodes classées pour les deux archives. Ceci met en évidence l'avantage d'utiliser UMAP avant la tâche de clustering indépendamment de l'utilisation ou non d'une représentation par apprentissage profonde. Cependant, l'espace latent "profond" semble bénéficier davantage d'UMAP que l'espace original. Même si la méthode *KUMAP* est classée avant les autres méthodes non-profondes, elle reste inférieure au meilleur candidat de clustering profond avec une confiance $p < 0.02$ au test de Wilcoxon pour l'archive univariée. Cependant, pour l'archive multivariée, la différence ne passe pas le test de Wilcoxon (avec $p = 0.10$). De plus, pour l'archive multivariée, la combinaison d'apprentissage profond seul (sans UMAP) est significativement moins performante que la méthode *KUMAP*, ce qui confirme que la combinaison d'UMAP et de l'apprentissage profond permet d'obtenir ce score sur l'archive multivariée. Notez que le score de NMI moyen

13. https://pypi.org/project/umap-learn/#description
14. https://github.com/johnpaparrizos/kshape




(a) *ResNet-rec-IDEC* sur ShapletSim (b) *DCNN-rec-DEC* sur BME

FIGURE 5.15 – Évolution du score de NMI et des fonctions de coût lors de l'apprentissage des paramètres (les dernières époques correspondent à l'ajout de la fonction de coût de clustering).

pour *KUMAP* est de 0,352 pour l'archive univariée et de 0,390 pour l'archive multivariée.

Globalement, la combinaison *ResNet-multi_rec-None-KUMAP* obtient le meilleur classement si l'on prend en compte les deux archives. Mais le gain obtenu grâce à l'utilisation des DNNs reste faible et difficile à généraliser.

### 5.2.4 Comportement des modèles et tendances

Dans cette section, nous cherchons à mieux comprendre la formation des DNNs et l'espace latent obtenu à travers différents aspects.

#### Entraînement des DNNs et tâche de clustering

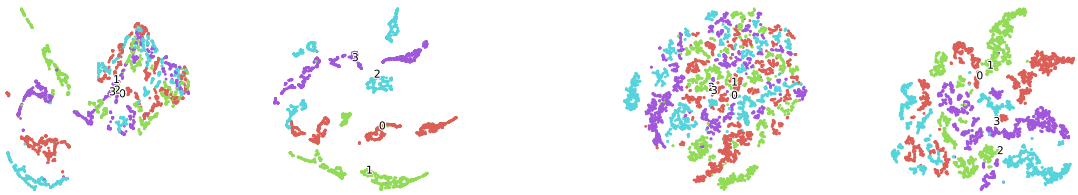
Comme nous n'entraînons pas les DNNs à prédire directement les classes, la corrélation entre la fonction de coût et l'évolution des performances de clustering peut ne pas être vérifiée. Dans la figure 5.15, nous avons tracé l'évolution du score de NMI sur le jeu de test comparé à l'évolution de la valeur de la fonction de coût de prétexte et de clustering.

Pour rappel, les 1000 premiers *batchs* sont lancés sans la fonction de coût de clustering, pour le jeu de données ShapeletSim ce qui correspond à 500 époques et 334 pour CBF. La combinaison des fonctions de coût de clustering et de prétexte est également lancée sur 1000 *batchs*, pour DEC et IDEC. Ils incluent un critère d'arrêt qui se déclenche lorsque le clustering ne change pas d'une itération à l'autre, ce qui explique l'arrêt précoce.

Sur les deux graphiques, on peut observer que le score de NMI n'est pas stable et peut finir sous sa valeur maximale. De plus, nous pouvons voir que l'ajout de la fonction de coût de clustering a tendance à fortement perturber la représentation latente. Cela entraîne souvent une baisse des performances de la représentation apprise, ce qui se traduit par un score de 0.0 de NMI dans la figure 5.15a. Cette observation peut être généralisée à presque tous les jeux de données et combinaisons de DNNs. Elle peut être confirmée en mesurant l'écart entre le score de NMI maximal et final à chaque exécution. Pour les deux archives, cet écart est significatif. Pour l'archive univariée, l'écart moyen entre toutes les combinaisons varie de 0,029 à 0,256 points de score de NMI. Pour les deux meilleures méthodes, il est de 0,078 pour *ResNet-mulit_rec-None* et de 0,073 pour *DCNN-rec-None*. Pour la première, cela implique une baisse du score moyen de NMI de 0,434 à 0,356. Même s'il ne semble pas réaliste de trouver un moyen d'atteindre ce score maximum, cela montre tout de même la capacité potentielle des DNNs à créer une représentation adaptée à la tâche de clustering. De plus, nous pouvons voir dans la figure 5.15a que si nous arrêtons le processus autour de l'époque 250, nous pouvons avoir un résultat très différent d'une époque à l'autre.

(a) UMAP (b) t-SNE

FIGURE 5.16 – Projection en deux dimensions du jeu de données CBF en utilisant différentes méthodes entre l'espace brut (à gauche) et l'espace latent du DNN (à droite) avec UMAP ou t-SNE. L'espace latent est obtenu avec la combinaison *DCNN-rec-None*.



(a) UMAP (b) t-SNE

FIGURE 5.17 – Projection en deux dimensions du jeu de données TwoPatterns en utilisant différentes méthodes entre l'espace brut (à gauche) et l'espace latent du DNN (à droite) avec UMAP ou t-SNE. L'espace latent est obtenu avec la combinaison *DCNN-rec-None*.

Sur certains jeux de données, cette capacité peut être confirmée visuellement. Pour ce faire, nous projetons les jeux de données en 2 dimensions. Pour effectuer la projection, nous utilisons deux méthodes de réduction de dimension, UMAP [McInnes et collab., 2018], et t-SNE [Maaten et Hinton, 2008]. Pour chaque méthode, nous avons tracé la projection avec les données originales (sans aucun prétraitement) et les données dans l'espace latent appris par les DNNs. Les projections sont présentées dans les figures 5.16, 5.17 et 5.18.

La représentation apprise sur les jeux de données CBF et ShapletSim illustre clairement le gain de l'utilisation des DNNs sur ces jeux de données. C'est encore plus clair pour ce dernier pour lequel les clusters ne pouvaient pas être distingués avec les données brutes. Cependant, les DNNs ne conduisent pas toujours à d'aussi bonnes représentations. Pour le jeu de données TwoPatterns, même si les données forment des groupes plus faciles à distinguer, chaque classe reste divisée en plusieurs groupes, ce qui rend la tâche de clustering difficile. Le clustering par Kmeans sur l'espace original et l'espace latent a donné un résultat médiocre (environ 0,02

(a) UMAP (b) t-SNE

FIGURE 5.18 – Projection en deux dimensions du jeu de données ShapeletSim en utilisant différentes méthodes entre l'espace brut (à gauche) et l'espace latent du DNN (à droite) avec UMAP ou t-SNE. L'espace latent est obtenu avec la combinaison *ResNet-multi_rec-None*.

score NMI). D'après les graphiques que nous avons analysés, l'espace latent généré par les DNNs tend à créer des groupes plus denses et séparables dans les données, mais ces groupes ne correspondent pas nécessairement à la partition attendue.

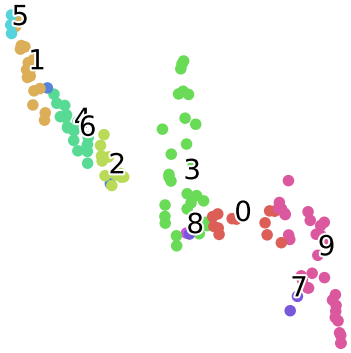
### Corrélation entre types de modèles et jeux de données

Trouver des corrélations entre un type d'élément (par exemple l'utilisation des réseaux récurrents) et les caractéristiques de certains jeux de données aiderait grandement l'utilisateur à sélectionner la meilleure combinaison pour analyser un nouveau jeu de données.

Pour ce faire, nous avons représenté les jeux de données de toutes les archives dans une projection à deux dimensions. Chaque jeu de données est représenté par un vecteur dont la valeur est la performance obtenue par un ensemble de combinaisons de réseaux de neurones sur ce jeu de données (indiqué par "*Données :*" dans la figure 5.19). Nous avons testé différentes combinaisons (par exemple, les meilleures combinaisons de chaque architecture/fonction de coût de prétexte/clustering, toutes les combinaisons, toutes les architectures avec une fonction de coût de prétexte spécifique et sans fonction de coût de clustering). Cette représentation est ensuite projetée avec la méthode UMAP en deux dimensions et est colorée avec une autre propriété du jeu de données (indiquée par "*Couleur :*" dans la figure 5.19). Avec ce processus, nous cherchons à identifier des groupes/clusters avec une coloration homogène et donc un comportement identique. Nous avons testé les propriétés des jeux de données suivants : la longueur de la série temporelle, la catégorie du jeu de données (par exemple ECG, Spectrographie, Image, ...), la taille de jeu d'entraînement, la taille du jeu de test, et le nombre de classes. Cependant, les résultats ne sont pas concluants. Certains des graphiques sont présentés dans la figure 5.19. Par exemple, dans la figure 5.19b, les jeux de données sont représentés par l'ensemble des performances obtenues sur toutes les architectures combinées avec la fonction de coût de reconstruction et sans fonction de coût de clustering et colorées avec la longueur de la série temporelle (plus clair indiquant une longueur plus longue, et plus sombre une longueur plus courte).

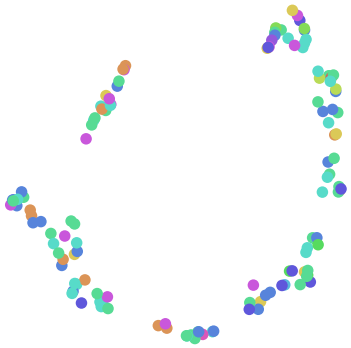
Dans l'ensemble, nous pouvons constater qu'aucune tendance ne peut vraiment être observée dans les graphiques. Pour tous les graphiques, le principal facteur de séparation correspond à la performance moyenne de toutes les combinaisons. Dans la figure 5.19a, nous avons appliqué un clustering avec la méthode Kmeans sur l'ensemble des jeux de données pour essayer d'identifier un comportement similaire entre certains jeux de données. La plupart des clusters regroupent des résultats homogènes, avec des performances bonnes, mauvaises ou moyennes sur toutes les combinaisons sélectionnées. Il est donc difficile de trouver des tendances et de décider quelle combinaison sera la plus adaptée pour un nouvel ensemble de données. Certains clusters, comme les numéros 8 et 6 regroupent des jeux de données où les architectures CNN ont des performances significativement meilleures que les autres. Cependant, ils ne totalisent que 5 jeux de données (sur 128) sans similarité particulière.

La conclusion ressortant de l'analyse de ces données est la difficulté de trouver une corrélation cohérente entre les performance des modèles et les propriétés des jeux de données. Cependant, il faut rappeler que ces deux archives regroupent une grande variété de jeux de données qui peuvent avoir des types de caractéristiques/motifs temporels très dissemblables.

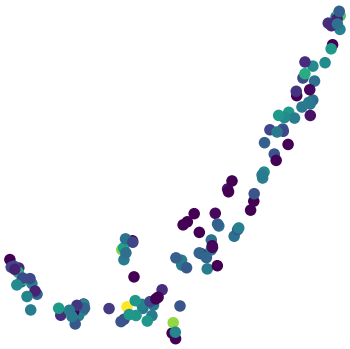



(a) *Couleur* : clustering, *Données* : toutes les architectures avec la fonction de coût de reconstruction et aucune fonction de coût de clustering

(b) *Couleur* : longueur des séries temporelles, *Données* : toutes les architectures avec la fonction de coût de reconstruction et aucune fonction de coût de clustering



(c) *Couleur* : catégorie du jeu de données, *Données* : top 3 des combinaisons de chaque architecture avec et sans UMAP



(d) *Couleur* : taille du jeu d'entraînement, *Données* : *DCNN* et *ResNet* avec toutes les fonction de coût de prétexte mais sans fonction de coût de clustering

FIGURE 5.19 – Chaque graphique montre des jeux de données univariées projetés en deux dimensions obtenues avec UMAP. La valeur de chaque jeu de données est calculée à partir de la performance d'un ensemble de combinaisons décrit dans *Data* et colorée à l'aide du critère décrit dans *Coloring*.

## 5.3 Comprendre les caractéristiques apprises par les réseaux

L'analyse conduit dans la section 5.2.4 permet de mieux comprendre l'intérêt d'utiliser l'apprentissage profond pour obtenir un clustering de qualité. Toutefois, lors de l'analyse d'un résultat de clustering, l'utilisateur souhaite également pouvoir comprendre la décision prise par la méthode de clustering. Or, les DNNs, manquent d'interprétabilité car les modèles produits ne peuvent pas être compris intuitivement par un humain Lipton [2018]. Il est donc difficile, voire impossible, d'affirmer sur quelles caractéristiques repose la décision du modèle.

L'interprétabilité d'un modèle est un problème majeur à différents niveaux. Lorsque le modèle ne parvient pas à obtenir les prédictions attendues, nous voulons être en mesure d'identifier les principales limitations pour améliorer le modèle existant. D'autre part, lorsqu'un modèle atteint des performances élevées, nous voulons vérifier que la décision est basée sur les bonnes caractéristiques. Une bonne interprétabilité résulte en une meilleure fiabilité et une plus grande confiance de la part des utilisateurs. Cela leur permet également d'en tirer des informations sur les données, par exemple des motifs temporels importants, des similitudes entre objets, etc.

Malheureusement, il n'existe pas de méthode générique et efficaces pour l'ensemble des DNNs. Toutefois, dans le domaine de l'analyse d'images, les performances importantes des modèles à base de convolutions He et collab. [2016]; LeCun et collab. [2010] ont donné lieu à la proposition de méthodes d'interprétabilité basées les cartes d'activation (*activation maps* en anglais) qui sont les sorties des filtres de la couche de convolution. La première méthode proposée, Class Activation Mapping (CAM), consiste à calculer un filtre d'activation (*carte de chaleur*) basée sur le degré d'activation des sorties de la dernière couche de convolution pour une classe prédite donnée. D'abord proposée par Zhou et collab. [2016], cette version n'est applicable qu'aux modèles CNN qui ne contiennent pas de couches entièrement connectées et nécessitait une architecture spécifique, ou une modification de celle-ci avec un entraînement supplémentaire. Selvaraju et collab. [2017] ont proposé une autre version, grad-CAM, qui peut être utilisée sans aucun processus supplémentaire et qui se généralise à tous les modèles de CNNs.

Cependant, CAM et grad-CAM s'appuient tous deux sur les couches de softmax utilisées pour la prédiction des classes afin de calculer la carte de chaleur pour chaque classe. Par conséquent, ils ne peuvent pas être appliqués directement aux modèles CNNs dédiés au clustering. En effet, comme nous l'avons vu précédemment, les modèles de clustering n'incluent pas nécessairement de couches de softmax, notamment dans le cas de l'utilisation d'un simple autoencodeur. Certaines méthodes utilisent une couche de clustering (*DEC*,*IDEC*, *ClusterGan*, ...), mais à l'exception de quelques méthodes (uniquement *DEPICT* dans notre étude), elles ne peuvent être directement considérées comme l'équivalent d'une couche de softmax.

Dans la suite nous discuterons d'abord des travaux connexes et donnerons plus de détails sur l'utilisation des cartes d'activation (section 5.3.1). Nous présenterons ensuite notre méthode (section 5.3.2), puis comment nous allons évaluer la pertinence de son utilisation (section 5.3.3) et les résultats obtenus (section 5.3.4), sur sa capacité à localiser (section 5.3.4) et discriminer (section 5.3.4) les caractéristiques des clusters. Enfin, nous conclurons par une discussion, d'une part, sur les limites actuelles de la méthode et les perspectives dans la section 5.3.5 et d'autre part, sur comment cela peut nous aider à analyser les modèles obtenue dans l'étude comparative de la section précédente 5.3.6.

### 5.3.1 Connaissances de base et travaux connexes

Dans cette section, nous présentons dans un premier temps les approches CAM et Grad-CAM plus en détail et, dans un second temps, les travaux existants pour les appliquer dans un contexte de clustering. Cependant, avant d'aller plus loin, nous voulons mentionner que même si nous nous plaçons dans le contexte des séries temporelles, un raisonnement similaire peut être fait pour d'autres types de données où les modèles CNN sont utilisés, comme les

images. Dans le cas des images, la dimension temporelle est remplacée par les dimensions spatiales.

### CAM et Grad-CAM

La plupart du temps, la représentation latente apprise par un modèle de clustering profond est un vecteur unidimensionnel qui permet d'ignorer les dimensions spécifiques de la donnée en entrée (la temporalité dans notre cas). Cela peut être un avantage dans de multiples tâches, car nous voulons être invariants par rapport à un décalage temporel ou un changement d'amplitude, mais la dimension latente perd le lien avec la localisation temporelle de chaque caractéristique latente dans la série en entrée.

Cependant, l'encodeur est composé d'un ensemble de couches de différents types. Parmi elles, les couches convolutives préservent la nature dimensionnelle des données (c'est-à-dire spatiale pour les images et temporelle pour les séries temporelles). Comme présenté plus en détail dans la section 5.1.1, les couches convolutives sont constituées d'un ensemble de filtres. La sortie est obtenue en passant les filtres sur l'objet en entrée pour obtenir un ensemble de cartes d'activation qui seront transmises à la couche suivante. Pour une couche convolutive $l$, la sortie de l'encodeur après la couche $l$ peut être vue comme une nouvelle série temporelle, notée $A \in \mathbb{R}^{T' \times f}$, où $T'$ la longueur de la série, et $f$ le nombre de caractéristiques/filtres de la série. Ainsi, comme la convolution est une opération locale, il existe une correspondance temporelle entre la série originale et la sortie de la couche. Par conséquent, nous pouvons l'utiliser pour localiser le degré d'activation dans sortie de la couche précédente et par transitivité dans la série originale en entrée.

**Class Activation Mapping (CAM) :** L'idée derrière cette méthode est de trouver la combinaison linéaire des cartes d'activation générées, $A$, qui a abouti à la prédiction par le réseau de la classe $c$. Leur méthode est simple mais nécessite que le réseau soit composé de couches convolutives suivi par une couche de GAP (Global Average Pooling) et une couche de softmax, que nous noterons $S$. Ainsi, la sortie de la couche de softmax pour la classe $c$, $S^c$ est une combinaison linéaire de la sortie GAP, qui est, par construction, l'activation moyenne de chaque filtre de la dernière couche convolutive $A$, donc :

$$S^c = \sum_{k=1}^{f} W_k * \sum_{i=0}^{w'} \sum_{j=0}^{h'} A_{i,j}^k = \sum_{k=1}^{f} W_{c,k}, \tag{5.39}$$

où $A^k$ est le $k^{\text{ième}}$ filtre de $A$, et $W_k$ est le poids de la couche de softmax. Comme illustré dans la figure 5.20, la CAM est alors calculée comme suit :

$$CAM = \sum_{k=1}^{f} A^k * W_{c,k} \tag{5.40}$$

**grad-CAM :** Dans cette approche, la carte de chaleur est également calculée par rapport à une classe, $c$. Elle utilise la $c^{\text{ème}}$ valeur de la couche de softmax, $S^c$, pour calculer la valeur du gradient au niveau de la carte d'activation $A$ de la dernière couche convolutive. Notez que dans ce cas, cette couche n'est pas nécessairement suivie uniquement par la combinaison d'une couche de GAP et d'une couche de softmax. Chaque poids de filtre est calculé pour chaque poids de $A$ $W_{c,k}$ puis les filtres sont additionnés de manière pondérée :

$$W_{c,k} = \frac{1}{D} \sum_{i=0}^{w'} \sum_{j=0}^{h'} \frac{\partial S^c}{\partial A_{i,j}^k} \tag{5.41}$$

où $D = T$ est la dimension d'entrée. Notez que, selon les données entrées, elle peut avoir une ou plusieurs dimensions (par exemple, la dimension temporelle pour les séries chronologiques),

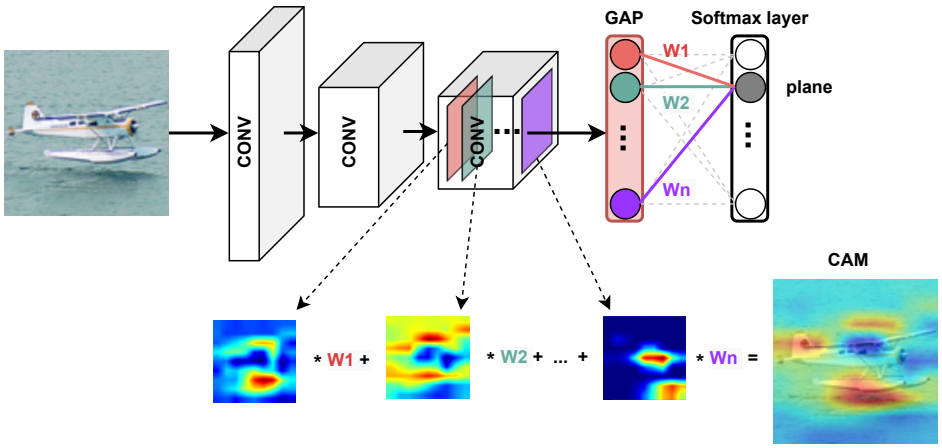



FIGURE 5.20 – La méthode CAM qui s'appuie sur la couche de Global Average Pooling pour déterminer les poids des filtres de la dernière couche convolutive dans la décision finale.

mais l'équation est donnée pour les convolutions en 2D.
Nous obtenons à partir de ces poids le degré d'activation pour le filtre de chaque couche. La carte de chaleur est constituée de la somme pondérée de toutes les sorties des filtres avec leur degré d'activation :

$$L^c_{grad-CAM} = ReLU(\sum_k W_{c,k} A^k) \tag{5.42}$$

Ainsi, cela met en évidence les filtres de convolution qui sont à la fois fortement activés par l'objet en entrée et qui contribuent au calcul du gradient. Notez que la fonction $ReLU$ est ajoutée pour ne garder que les contributions positives à $S^c$. Par conséquent, en restreignant le calcul du gradient à $S^c$, nous ne sélectionnons que les résultats de convolution qui contribuent à cette classe spécifique.

#### Les applications au clustering

L'adaptation de CAM aux méthodes de clustering reste très limitée. Ceci s'explique par l'absence d'une couche de softmax dans ces méthodes. Il est donc difficile de trouver un moyen de calculer la contribution de chaque filtre au choix d'un cluster séparément.

RYAN et collab. [2020] ont proposé la première adaptation de CAM pour le clustering, appelée CLustering Activation Mapping (CLAM). Ils ont utilisé une variation de grad-CAM, Score-CAM WANG et collab. [2019b], qui utilise une prédiction supplémentaire par le modèle de DNN avec une entrée masquée basée sur chaque filtre de la dernière couche convolutive $A$. Cela permet de mieux fixer le poids de chaque filtre dans la décision basée sur la sortie de la couche de softmax de la classe évaluée. Cependant, pour l'appliquer à une tâche de clustering, RYAN et collab. [2020] ont résolu le problème de l'abscence de couche softmax en utilisant la méthode Deep Embedded Clustering (DEC) XIE et collab. [2016] (voir section 5.1.3). En se basant sur la similarité entre la couche de clustering et une couche softmax, les auteurs ont simplement transposé la méthode Score-CAM en remplaçant le $S^c$ par $q_k$.

Cependant, cette approche nécessite toujours d'avoir un DNN de clustering qui inclut une couche dont l'objectif est similaire à celui d'une couche softmax. De plus, dans notre cas, les méthodes les plus robustes (celles sans fonction de coût de clustering) sont celles qui n'en utilisent pas.

### 5.3.2 Grad-CeAM : une adaptation de grad-CAM pour le clustering basé sur le calcul de centres

Dans le cas du clustering, nous n'avons pas de couche de softmax avec un neurone correspondant à chaque classe comme dans le cas de la classification. Ainsi, nous voulons remplacer

l'utilisation de la valeur de la classe en sortie de la couche de softmax $S^c$ par une valeur qui aidera à sélectionner les convolutions qui contribuent à faire appartenir l'objet au cluster analysé. Cependant, au lieu des classes, nous avons des clusters qui sont souvent représentés par leurs centres. La méthode proposée se concentre sur les méthodes de clustering qui impliquent l'utilisation de centres (par exemple, KMEANS, EM, DEC, etc.) en plus de la représentation apprise par le CNN. Par conséquent, la dérivation utilisée pour calculer le gradient doit se faire sur une fonction de coût qui est plus élevée lorsque l'entrée est proche du centre analysé mais également plus faible pour les autres centroïdes. Pour ce faire, nous identifions les caractéristiques de la représentation latente $z$ qui contribuent le plus pour obtenir les poids $weight_{z,c}$ par rapport au cluster $c$. Ensuite, nous calculons une valeur pondérée de la représentation latente de l'objet en entrée. Les poids $weight_{z,c}$ doivent mettre l'accent sur les caractéristiques qui contribuent le plus à rendre $z$ proche du centre $\mu^c$ du cluster $c$. Pour ce faire, les poids doivent :

1. discriminer entre tous les clusters. Par exemple, certaines des caractéristiques de l'espace latent peuvent être faibles ou élevées pour tous les clusters et ne contribuent donc pas beaucoup au choix du cluster. Ceci est calculé indépendamment de $z$ par la formule suivante :

$$centroids\_weights^c = \left\| \sum_{\mu^k \in M \setminus \{\mu^c\}} |\mu_k - \mu_c| \right\|_{0-1}, \tag{5.43}$$

   où $M$ est l'ensemble de tous les centres, $\|.\|_{p-q}$ est la normalisation entre $p$ et $q$, et $|.|$ est la valeur absolue.

2. contribuer à rendre $z$ plus proche de $\mu^c$ que les autres centres. En effet, nous ne voulons que les caractéristiques qui contribuent positivement à ce que $z$ soit affecté au cluster $c$. Ceci est fait sur la base de la représentation $z$ par la formule suivante :

$$representation\_weights^c(z) = \| |\mu^c - z| \|_{min_M - max_M}, \tag{5.44}$$

   où $min_M = reduce\_min(\sum_{\mu^k \in M \setminus \{\mu^c\}} |\mu^k - z)|$
   et $max_M = reduce\_max(\sum_{\mu^k \in M \setminus \{\mu^c\}} |\mu^k - z)|$.

A partir de ces deux éléments, on obtient $weight_{z,c}$ et la fonction de coût finale :

$$weight_{z,c} = representation\_weights^c(z) * centroids\_weights^c, \tag{5.45}$$

$$L^c_{centroid}(z) = z * weight_{z,c}, \tag{5.46}$$

Les cartes d'activations sont obtenues par :

$$W^c_k = \frac{1}{D} \sum_{i=0}^{w'} \sum_{j=0}^{h'} \frac{\partial L^c_{centroid}(z)}{\partial A^k} \tag{5.47}$$

La carte de chaleur est alors caculée comme :

$$L^c_{grad-CeAM} = \sum_k ReLU(W^c_k) A^k, \tag{5.48}$$

Pour la grad-CAM classique, la fonction ReLU est également appliquée à $A^k$, mais dans notre cas, certaines valeurs négatives peuvent encore contribuer au choix du centre (par le masquage de certaines partie de l'objet en entrée). Le méthode complète est résumée dans la figure. 5.21. Dans la suite, nous évaluons notre méthode sur deux aspects : la localisation et la discriminabilité.

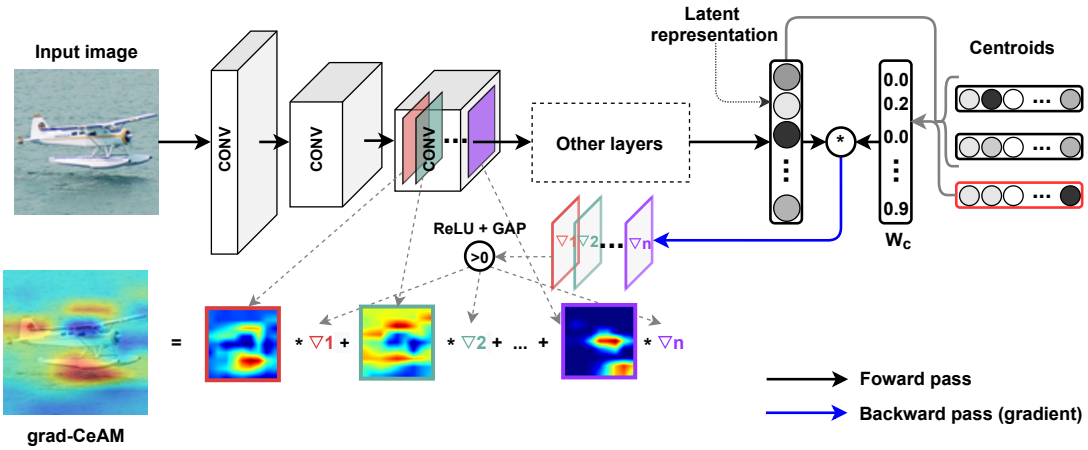



FIGURE 5.21 – Grad-CeAM combine la sortie de la dernière couche convolutive avec le gradient rétro-propagé et seuillé.

### 5.3.3 Paramètres expérimentaux

#### Jeux de données utilisés

Pour valider notre méthode, nous avons décidé de ne pas nous limiter aux séries temporelles mais également d'intégrer des jeux de données d'images, afin de montrer que notre méthode n'est pas limitée à un seul type de données.

Pour le domaine des images, nous avons sélectionné :

— MNIST : Un jeu de données d'images en niveaux de gris de 28 par 28. Il est composé de 70 000 chiffres manuscrits, 10 000 dans l'ensemble d'entraînement et 60 000 dans l'ensemble de test, répartis de manière égale. Les chiffres sont centrés et leur taille est normalisée LeCun et collab. [1998]. Des exemples des classes sont présentés dans la figure 5.22.

FIGURE 5.22 – Exemples des classes du jeu de données MNIST.

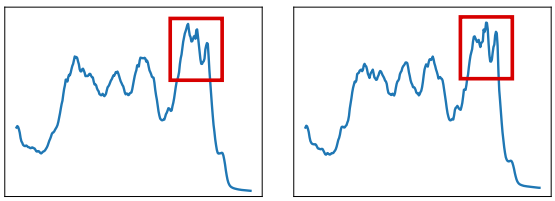
— STL-10 : Un jeu de données d'images en couleur de 96 par 96. Il y a 10 classes, avec 13000 exemples, 5000 dans l'ensemble d'entraînement et 8000 dans l'ensemble de test répartis de manière égale. Il s'agit de classer les images des éléments suivants : avion, oiseau, voiture, chat, cerf, chien, cheval, singe, bateau, camion Coates et collab. [2011]. Des exemples de ces classes sont présentés dans la figure 5.23.

FIGURE 5.23 – Exemples des classes du jeu de données STL-10.

Et pour les séries temporelles nous avons utilisé les jeux de données de l'archive de l'UCR suivants :

— Coffee : Un ensemble de données de séries temporelles univariées de longueur 286. Il est composé de 56 enregistrements de spectrographes alimentaires, 28 dans l'ensemble d'entraînement et 28 dans l'ensemble de test répartis de manière égale. Il est composé

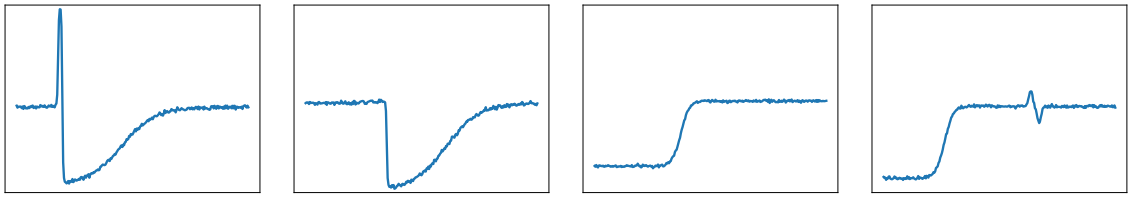
de deux classes, chacune correspondant à un type de grains de café Briandet et collab. [1996]. Des exemples de ces deux classes sont présentés dans la figure 5.24.



(a) classe 1 (b) classe 2

Figure 5.24 – Exemples des classes du jeu de données Coffee, les carrés rouges indiquent la zone discriminante entre les deux classes.

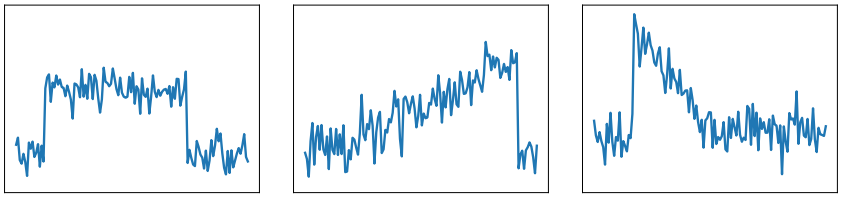
— Trace : Un ensemble de données de séries temporelles univariées de longueur 286. Il s'agit d'un ensemble de données synthétiques à 4 classes conçu pour simuler des défaillances d'instruments dans une centrale nucléaire qui sont différenciées par leurs modèles temporels. Des exemples des quatre classes sont présentés dans la figure 5.25.



(a) classe 1 (b) classe 2 (c) classe 3 (d) classe 4

Figure 5.25 – Exemples des classes du jeu de données Trace.

— CBF : Un jeu de données de séries temporelles univariées de longueur 128. Il s'agit d'un ensemble de données synthétiques à trois classes conçu pour distinguer trois formes : cylindre (classe 1), cloche (classe 2) et entonnoir (classe 3) Saito [1994]. Des exemples de ces trois classes sont présentés dans la figure 5.26.



(a) classe 1 (b) classe 2 (c) classe 3

Figure 5.26 – Exemples des classes du jeu de données CBF.

### Modèles de réseaux neuronaux convolutifs utilisés

Pour apprendre la représentation latente, nous avons utilisé des autoencodeurs convolutifs (CAE).

Pour les jeux de données d'images, nous avons utilisé une architecture simple d'encodeur-décodeur. L'encodeur est composé de $d$ couches convolutionnelles 2D, suivies d'une couche de GAP et d'une couche *FCNN* de taille 10 pour la couche d'intégration. Le décodeur est construit comme un miroir des $d$ couches convolutionnelles 2D et se termine par une couche convolutionnelle finale pour s'adapter à la dimension d'entrée. Pour STL-10, nous avons utilisé $d = 3$ avec des paramètres fixés à [32, 64, 64] pour les filtres, avec [3, 5, 5] pour les tailles de

noyau, et [2, 2, 2] pour les strides. Pour MNIST, nous avons utilisé $d = 2$ avec des paramètres fixés à [32, 64] pour les filtres, avec [3, 5] pour les tailles de noyau, et [2, 2] pour les strides.

Pour les jeux de données de séries temporelles, nous avons utilisé l'architecture *ResNet* (voir section 5.2.1). Tous les CAE sont entraînés avec une fonction de coût de reconstruction classique (voir équation 5.16). Les centres sont calculés avec la méthode KMEANS exécutée sur la représentation latente produite par l'encodeur.

Avant d'aborder l'évaluation elle même, il convient de mentionner que pour toutes les expériences, nous voulons évaluer l'interprétabilité et l'exactitude des résultats fournis par notre méthode de visualisation. Ainsi, que le modèle CNN réel obtienne de bonnes ou de mauvaises performances de clustering, nous voulons que cela soit reflété dans la visualisation.

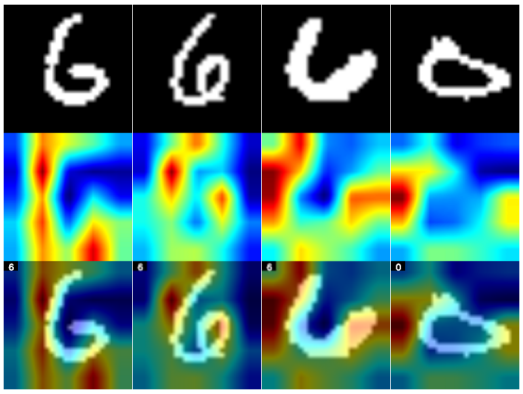
### 5.3.4 Evaluation

#### Localisation et analyse de modèle

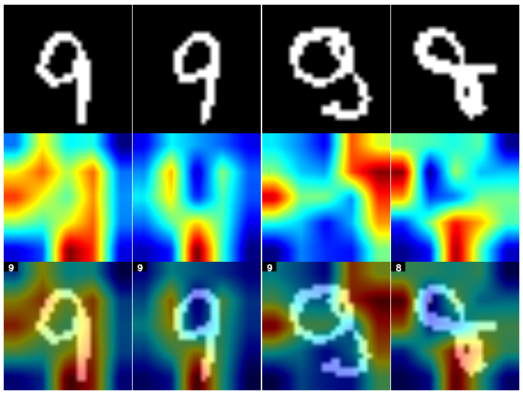
La localisation dénote la capacité de la méthode à identifier correctement les caractéristiques et les régions de l'objet en entrée qui ont été utilisées dans le processus de décision. Pour les méthodes supervisées, il est habituel d'utiliser l'évaluation des boîtes de délimitation (*bounding boxes* en anglais) sur le réseau formé pour la classification afin d'évaluer si la localisation, obtenue avec les outils de CAM, parvient à mettre en évidence la zone appropriée dans l'image. Cependant, cela implique d'avoir les étiquettes des données et les boîtes de délimitation correspondantes, et en outre, que le modèle CNN prenne une décision cohérente. Ceci n'est pas réaliste dans une configuration non supervisée, où les jeux de données utilisés sont souvent simplifiés avec seulement un élément centré par image ou un modèle général pour les séries temporelles, ce qui rend inutile l'utilisation de boîtes de délimitation. De plus, cela suppose également que l'algorithme base sa décision sur la partie appropriée de l'objet en entrée, mais étant donné les résultats obtenus par les méthodes de clustering, nous pouvons supposer que ce n'est pas nécessairement le cas. En résumé, nous ne voulons pas évaluer la capacité de notre méthode à mettre en évidence la zone d'intérêt attendue mais celle utilisée par le modèle CNN. Par conséquent, nous voulons que la localisation proposée par notre méthode corresponde aux motifs fréquents parmi les objets regroupés au sein du même cluster. Cela permettra également d'évaluer la qualité des informations fournies par notre méthode.

Pour le jeu de données MNIST (figure 5.27), nous avons pris deux exemples, l'un du cluster correspondant à la classe 6 et l'autre à la classe 9. Pour la classe 6, la Grad-CeAM montre en fait une forte activation sur la barre verticale des chiffres 6 et également en bas à droite de la boucle. Par conséquent, l'affectation de la dernière image à droite, qui devrait être regroupée avec les autres chiffres 0, peut être mieux comprise car il n'est pas complètement arrondi sur le côté gauche et a une forme étroite sur le côté droit. Pour la classe 9, nous pouvons observer une activation symétrique (c'est-à-dire sur la barre de droite et à gauche de la boucle). Les deux figures les plus à gauche sont en fait plus proches du centre du cluster (donc plus représentatives) que les deux autres. Pour la troisième image, on peut observer que le bas du chiffre 9 n'est pas mis en évidence car il ne correspond pas à la barre droite des autres échantillons. Pour la dernière image, qui est un chiffre 8, la localisation de l'activation est similaire aux deux premiers échantillons, car la boucle inférieure du chiffre 8 est confondue avec la barre inférieure du chiffre 9 sans prendre en compte le trou inférieur. Ceci montre partiellement que le modèle appris est toujours sensible à une petite rotation.

Pour le jeu de données STL-10, nous avons affiché deux exemples de clusters dans la figure 5.28. On peut observer sur les deux jeux de données que l'affectation est en fait basée sur la disposition générale de l'image et non sur l'objet lui-même. Le cluster 6 regroupe les images où l'horizon divise l'image en deux parties et le cluster 9 regroupe les objets centrés dans un fond monochrome (ici le ciel). Les Grad-CeAM valident clairement cette interprétation. Pour cet ensemble de données, l'interprétation des résultats n'est pas simple. Cependant, les cartes de chaleur sur STL-10 montrent clairement que le réseau ne se concentre pas sur les objets

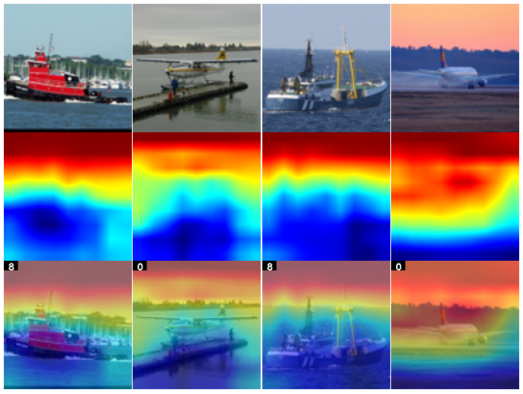


(a) cluster 6

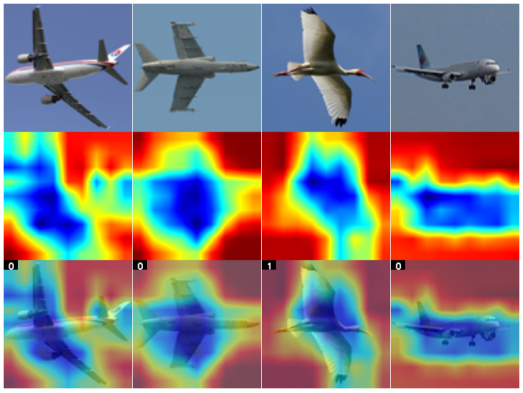


(b) cluster 9

FIGURE 5.27 – Cartes de chaleur de Grad-CeAM sur le jeu de données MNIST. La figure 5.27a contient les échantillons du cluster correspondant à la classe 6, et la figure 5.27b celui correspondant à la classe 9. Les régions rouges correspondent à une forte contribution et les régions bleues à une contribution quasi nulle à la correspondance avec le centre.



(a) cluster 6



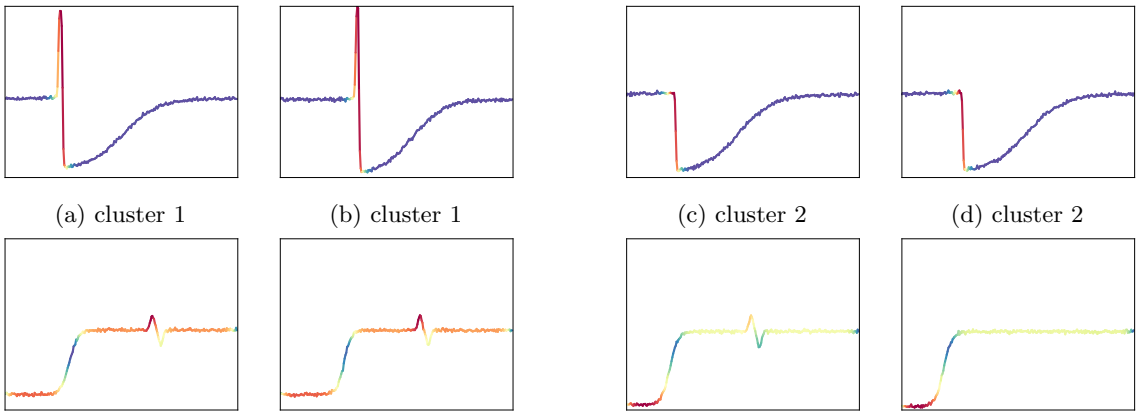
(b) cluster 9

FIGURE 5.28 – Cartes de chaleur de Grad-CeAM sur le jeu de données STL-10. La figure 5.28a contient les échantillons du cluster correspondant le plus à la classe *bateau*, et la figure 5.28b celui correspondant le plus à la classe *avion*. Les régions rouges correspondent à une forte contribution et les régions bleues à une contribution quasi nulle à la correspondance avec le centre.

eux-mêmes, en particulier pour le cluster 6, ce qui montre que les caractéristiques apprises par le réseau ne sont pas significatives pour cette tâche de clustering.

Pour le jeu de données Trace, dans la figure 5.29, les clusters 1 et 2 correspondent parfaitement aux classes 1 et 2 et les cartes de chaleur mettent précisément en évidence les pas de temps discriminants entre les deux classes. Cependant, il y a une confusion des classes 3 et 4. Pour ces deux classes, le clustering semble basé sur le fait que le saut se produit tôt ou tard dans la série temporelle et non sur la présence de la petite variation au niveau du plateau supérieur. Ceci est validé par le Grad-CeAM où la petite variation n'est pas mise en évidence.

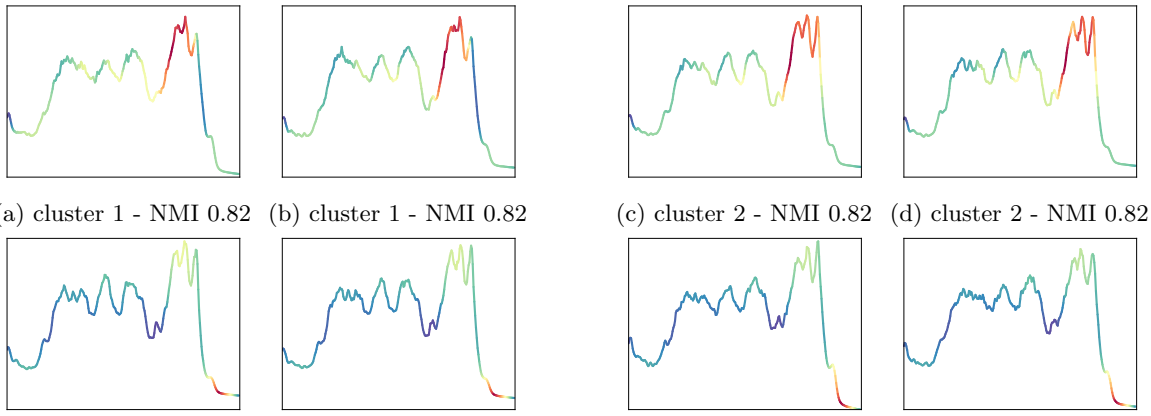
Pour le jeu de données Coffee, comme nous n'avons que deux classes et que nous obtenons de bons résultats (0,82 du score NMI), peu de conclusions peuvent être faite en comparant uniquement les deux clusters. Par conséquent, nous comparons la localisation des Grad-CeAM avec un modèle avec une *basse* performance de clustering, mesurée par leur score d'information mutuelle normalisée (NMI). Le modèle *low* est obtenu en sélectionnant le plus mauvais modèle parmi plusieurs exécutions. Dans la figure 5.30, nous pouvons facilement constater la différence entre les deux modèles. Le modèle *high* identifie parfaitement la zone de discrimination, tandis que le modèle *low* se concentre sur la fin de la série temporelle (bosse basse ou haute) qui n'a aucune implication dans la discrimination des classes.



(a) cluster 1 (b) cluster 1 (c) cluster 2 (d) cluster 2

(e) cluster 3 (f) cluster 3 (g) cluster 4 (h) cluster 4

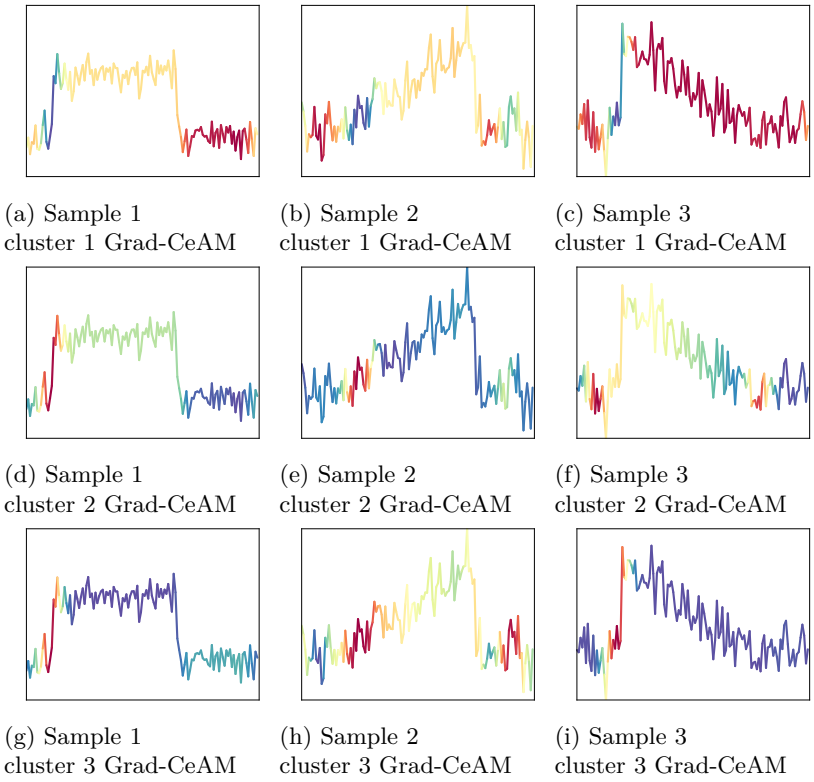
FIGURE 5.29 – Cartes de chaleur de Grad-CeAM sur le jeu de données Trace. Deux échantillons sont affichés par cluster. Les régions rouges correspondent à une forte contribution et les régions bleues à une contribution quasi nulle à la correspondance avec le centre (les couleurs sont lissées pour des raisons de clarté visuelle et pour mieux refléter la taille des filtres).



(a) cluster 1 - NMI 0.82 (b) cluster 1 - NMI 0.82 (c) cluster 2 - NMI 0.82 (d) cluster 2 - NMI 0.82

(e) cluster 1 - NMI 0.03 (f) cluster 1 - NMI 0.03 (g) cluster 2 - NMI 0.03 (h) cluster 2 - NMI 0.03

FIGURE 5.30 – Cartes de chaleur de Grad-CeAM sur le jeu de données Coffee, où les figures 5.30a, 5.30b, 5.30c et 5.30d sont des échantillons de chaque cluster issus d'un clustering avec un score de NMI de 0.82 et les figures 5.30e, 5.30f, 5.30g et 5.30h sont des échantillons de chaque cluster issus d'un clustering avec un score de NMI de 0.03. Les régions rouges correspondent à une forte contribution et les régions bleues à une contribution quasi nulle à la correspondance avec le centre (les couleurs sont lissées pour des raisons de clarté visuelle et pour mieux refléter la taille des filtres).



(a) Sample 1 cluster 1 Grad-CeAM
(b) Sample 2 cluster 1 Grad-CeAM
(c) Sample 3 cluster 1 Grad-CeAM
(d) Sample 1 cluster 2 Grad-CeAM
(e) Sample 2 cluster 2 Grad-CeAM
(f) Sample 3 cluster 2 Grad-CeAM
(g) Sample 1 cluster 3 Grad-CeAM
(h) Sample 2 cluster 3 Grad-CeAM
(i) Sample 3 cluster 3 Grad-CeAM

FIGURE 5.31 – Cartes de chaleur de Grad-CeAM sur le jeu de données CBF, où 5.31a,5.31b, et 5.31c sont des échantillons provenant respectivement des clusters 1, 2 et 3 colorisés avec le Grad-CeAM du centre du cluster 1, 5.31d,5.31e, et 5.31f sont les mêmes échantillons mais avec la colorisation Grad-CeAM du centre du cluster 2 et 5.31g,5.31h, et 5.31i pour le cluster 3. Les régions rouges correspondent à une forte contribution et les régions bleues à une contribution quasi nulle à la correspondance avec le centre (les couleurs sont lissées pour des raisons de clarté visuelle et pour mieux refléter la taille des filtres).

Pour le jeu de données CBF, dans la figure 5.31, nous avons affiché la comparaison des cartes de chaleur générées à partir des trois centres sur la même série temporelle pour montrer les différences entre les Grad-CeAM des trois centres. Pour les Grad-CeAM du cluster 1 (figure 5.31a à 5.31c), l'activation est forte lorsque le signal est stable, et particulièrement faible lorsque le signal augmente fortement. À l'inverse, les Grad-CeAM du cluster 3 sont clairement focalisées sur une forte augmentation du signal. Il est à noter que les couleurs des cartes de chaleur sont normalisées, donc la partie fortement rouge dans la figure 5.31c indique simplement que cette partie de la série temporelle contribue davantage à la décision, mais globalement le degré d'activation/contribution est plus faible que dans la partie rouge de la figure 5.31a.

Les résultats obtenus sur ces 5 jeux de données illustrent que la localisation de notre méthode aide à comprendre l'affectation des objets aux clusters et semble donc cohérente avec les clusters obtenus.

### Discrimination entre les clusters

Dans cette section, nous voulons évaluer si les caractéristiques mises en évidence par la méthode Grad-CeAM sont spécifiques à chaque cluster et permettent de comprendre le choix entre un cluster ou un autre.

Si la proposition précédente est vraie, amplifier (ou réduire) les valeurs des caractéristiques sélectionnées devrait respectivement avoir un effet positif (ou négatif) sur le choix du cluster analysé. Pour évaluer cela, nous devons utiliser le gradient calculé lors du calcul de Grad-CeAM pour pondérer la sortie de la dernière couche convolutive $A$. La nouvelle sortie est

calculée comme :

$$A_{c-graded} = \|ReLU(a_k^c)\|_{0-2} A^k \tag{5.49}$$

Les poids de $a_k^c$ sont normalisés entre 0 et 2 pour amplifier les valeurs élevées. Ensuite, $A_{c-graded}$ est utilisé comme entrée de la couche suivante du CAE pour obtenir la représentation modifiée $Z^c = \{z_1^c, ..., z_N^c\}$. Enfin, un nouveau clustering est calculé sur $Z^c$ mais basé sur les centres originaux, $M$, utilisés pour calculer le modèle Grad-CeAM. Ainsi, en mesurant la proportion d'objets affectés au cluster $c$, on peut estimer si la modification ajoutée a amélioré la probabilité qu'un objet soit affectée au cluster $c$.

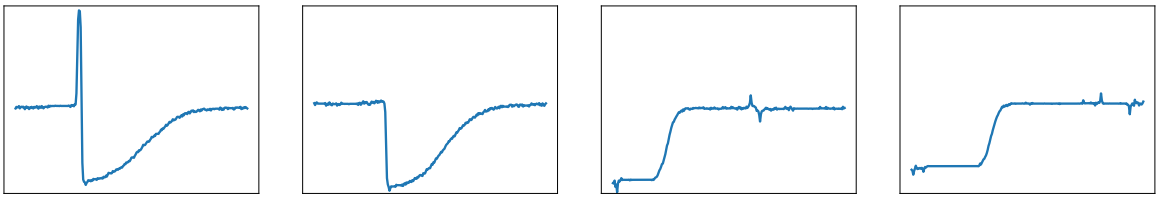
TABLEAU 5.2 – Comparaison de la cardinalité des clusters avant et après l'application des poids des gradients à la dernière couche convolutive.

| Jeu de données | STL-10 | MNIST | Trace | Coffee | CBF |
|---|---|---|---|---|---|
| Médiane | 1.1 | 1.4 | 1.5 | 1.2 | 1.7 |
| Moyenne | 1.2 | 1.4 | 1.7 | 1.2 | 1.4 |
| Minimum | 0.0 | 0.5 | 1.0 | 0.0 | 0.8 |
| Maximum | 3.1 | 5.8 | 2.8 | 2.3 | 1.8 |

Dans le tableau 5.2, nous avons rapporté le changement de proportion médian, moyen, minimal et maximal des affectations aux clusters. Les résultats que le cluster utilisé pour fixer les poids bénéficie largement de l'amplification. Cela montre donc qu'en moyenne, Grad-CeAM sélectionne les bonnes caractéristiques pour chaque centre. Cependant, nous pouvons également souligner que, même si le gain médian est positif (1,3 en moyenne, soit 30% de gain de cardinalité), certains clusters ne bénéficient pas de la modification. Pour tous les jeux de données, à l'exception de Trace, il y a au moins un cluster qui perd en cardinalité ou même disparaît (c'est-à-dire aucun objet n'est assigné à ce cluster). Pour MNIST et CBF, cela semble marginal, car cela se limite à un cluster, tous les autres ayant au moins une cardinalité stable. Pour Coffee, un cluster absorbe l'autre quel que soit le centre utilisé par la méthode Grad-CeAM. Cela s'explique par le fait qu'un cluster est défini par une valeur plus faible sur certaines caractéristiques discriminantes. Par conséquent, lorsque nous amplifions ces caractéristiques, sa valeur augmente au lieu d'être diminuée (pour être plus proche du centre), ce qui favorise l'autre cluster. Ces faibles valeurs signifient généralement une faible activation de la couche CNN provenant soit d'une faible amplitude du signal, soit de son absence, par exemple un pic de moindre amplitude pour le jeu de données Coffee. Pour la STL-10, l'explication est plus difficile à trouver en observant la représentation apprise. Cependant, cela peut être partiellement dû à deux éléments. Premièrement, comme pour le jeu de données Coffee, nous avons un groupe défini par des valeurs plus faibles pour certaines caractéristiques. Deuxièmement, dans un cas de représentation menant à de faibles résultats (seulement 0,2 du score NMI), cela peut supposer que les centres ne sont pas bien définis ou qu'ils reposent sur des caractéristiques peu représentatives. Un comportement similaire a également été observé pour le jeu de données MNIST lorsque le score de NMI était trop faible. Néanmoins, cela montre que notre méthode a du mal à identifier tous les éléments utilisés par le DNN pour générer la prédiction, en particulier pour traduire l'absence d'un motif particulier dans la donnée (c'est-à-dire de faibles valeurs dans las cartes d'activation).

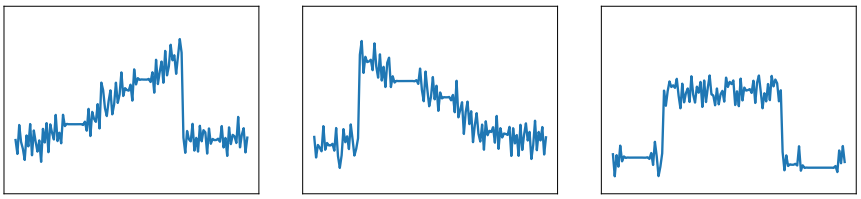
### 5.3.5 Conclusion

L'évaluation de notre méthode, Grad-Centroid Activation Mapping (Grad-CeAM), a montrée qu'elle permet d'obtenir une bonne représentation de l'importance des différents éléments de l'objet en entrée dans l'affectation à un cluster, que ce soit sur la localisation ou la discriminantes entre chaque cluster. Nous avons également montré que notre méthode peut fournir des informations utiles sur les motifs appris par le réseau de neurones, ce qui peut aider le chercheur à analyser et mieux comprendre le modèle de clustering obtenu. Cependant,



(a) Centre du cluster 1 (b) Centre du cluster 2 (c) Centre du cluster 3 (d) Centre du cluster 4

FIGURE 5.32 – Centres appris par KMEANS avec DTW et DBA sur le jeu de données Trace.



(a) Centre du cluster 1 (b) Centre du cluster 2 (c) Centre du cluster 3

FIGURE 5.33 – Centres appris par KMEANS avec DTW et DBA sur le jeu de données CBF.

même si nous obtenons de bons résultats globaux, nous montrons également que notre méthode peine à capter complètement les caractéristiques discriminantes pour certains centres, surtout si elles résultent d'une faible activation dans les couches du CNN.

Il faut également rappeler que Grad-CeAM n'est applicable que pour des réseaux de neurones à base de convolutions (CNNs). Toutefois, même si nous ne pouvons utiliser cette méthode que sur les combinaisons *ResNet*, *DCNN* et *SCNN*, nous l'avons utilisé pour mieux comprendre les écarts de qualité entre les résultats de différentes combinaisons dans la section suivante.

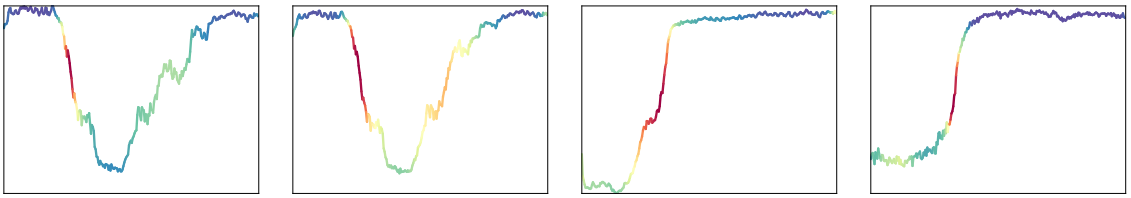
### 5.3.6 Application à l'étude comparative

Comme expliqué dans la section 1.1.2, nous voulons également évaluer la capacité des DNNs à prendre en compte la spécificité de la dimension temporelle lors du clustering des données. Cela signifie reconnaître les motifs temporels, même lorsqu'ils sont décalés ou étirés dans le temps.

Même si nous souhaitons utiliser les Grad-CeAMs, nous avons également utilisé le décodeur (pour les méthodes à base de fonction de coût de reconstruction) pour reconstruire les centres des clusters. À titre de comparaison, nous avons tracé les centres appris par KMEANS avec la métrique DTW et DTW Barycenter Averaging (DBA) dans la figure 5.32 pour l'ensemble de données Trace et dans la figure 5.33 pour l'ensemble de données CBF. DBA et DTW sont conçues pour être moins sensibles aux distorsions temporelles car elles réalignent les séries temporelles pour minimiser la distance entre elles. Ces deux méthodes se sont avérées efficaces pour capturer les motifs temporels [PETITJEAN et collab., 2011]. Pour le jeu de données CBF, nous pouvons observer que la méthode DBA extrait clairement les trois modèles. Pour le jeu de données Trace, les centres calculés avec DBA identifient également clairement les principaux motifs. Cependant, une confusion peut être observée entre la 3$^{\text{ème}}$ et la 4$^{\text{ème}}$ classe. Dans les données de référence, les deux classes sont distinguées par la présence ou non d'une perturbation finale. Le clustering discrimine en fait les deux classes par le niveau du premier plateau, inférieur à 1,5 pour la classe 3 et supérieur pour la classe 4.

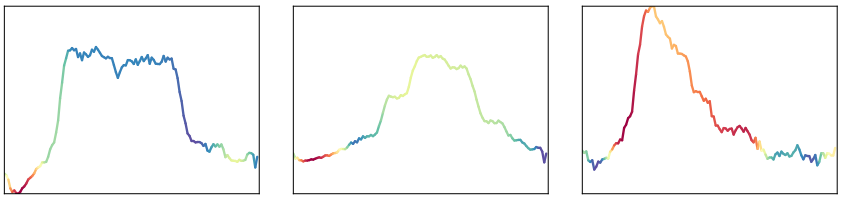
Dans la figure 5.34 nous avons affiché la reconstruction des centres des clusters utilisés dans l'évaluation précédente (figure 5.29) pour le jeu de données Trace, et la figure 5.35 pour CBF. Les séries temporelles ont également été colorées avec la méthode Grad-CeAM.

Pour le jeu de données Trace, même si le modèle obtient de bons résultats (avec un score de NMI similaire d'environ 0.75), les centres reconstruits ne rendent pas compte des



(a) Centre du cluster 1 (b) Centre du cluster 2 (c) Centre du cluster 3 (d) Centre du cluster 4

FIGURE 5.34 – Séries temporelles reconstruites à partir des centres obtenus par un clustering avec l'architecture *ResNet-multi_rec-None* sur le jeu de données Trace. Les régions rouges correspondent à une forte contribution et les régions bleues à une contribution quasi nulle à la correspondance avec le centre (les couleurs sont lissées pour des raisons de clarté visuelle et pour mieux refléter la taille des filtres).



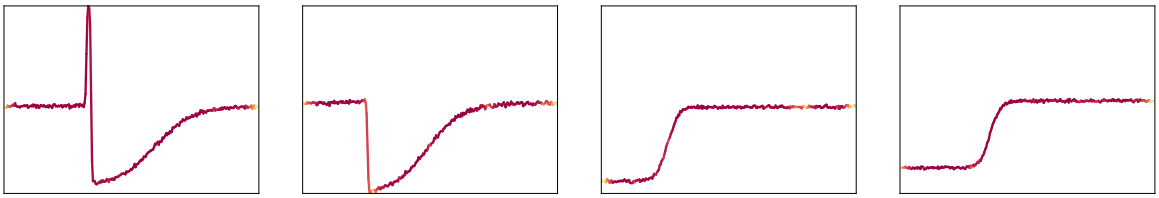
(a) Centre du cluster 1 (b) Centre du cluster 2 (c) Centre du cluster 3

FIGURE 5.35 – Séries temporelles reconstruites à partir des centres obtenus par un clustering avec l'architecture *ResNet-multi_rec-None* sur le jeu de données CBF. Les régions rouges correspondent à une forte contribution et les régions bleues à une contribution quasi nulle à la correspondance avec le centre (les couleurs sont lissées pour des raisons de clarté visuelle et pour mieux refléter la taille des filtres).

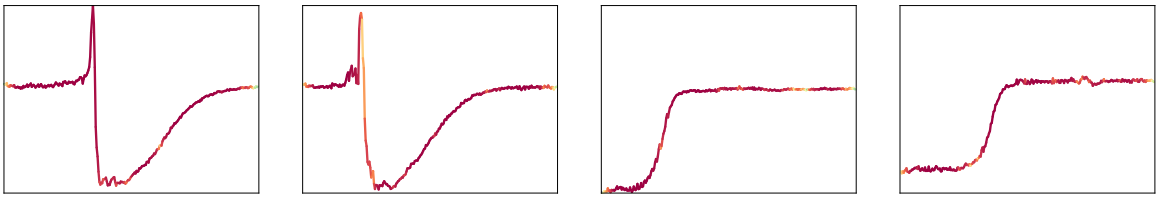
motifs temporels de la donnée original. C'est particulièrement le cas pour la classe 1, même si l'encodeur distingue parfaitement les classes 1 et 2 car le clustering obtient un score parfait sur ces deux classes. Par conséquent, il est possible que le décodeur ne parvienne pas à obtenir une bonne reconstruction. Pour le décodeur, il peut être plus optimal de ne pas reconstruire le pic de la classe 1. En effet, lors du calcul de l'erreur quadratique moyenne, omettre le pic dans la reconstruction peut donner lieu à un meilleur résultat que de le reconstruire mais de manière décalée. Dans le premier cas, le pic affectera l'erreur une fois, alors qu'il est probable qu'il l'affectera deux fois dans le second cas. De même, entre les classes 3 et 4, il pourrait être plus efficace de se concentrer sur le rendu correct des plateaux que sur le rendu de la petite variation à la fin.

Dans la figure 5.36, nous avons également affiché les cartes de chaleur obtenues avec Grad-CeAM sur le jeu de données Trace mais avec une autre combinaison, *DCNN-rec-None*. Cette combinaison obtient un score NMI plus faible (0,55). La zone utilisée pour l'assignation au cluster est, dans ce cas, répartie sur l'ensemble de la série temporelle. Une observation plus poussée des résultats de clustering, nous a permis de remarquer que tous les clusters sont basés sur le moment où la variation se produit mais pas sur sa forme. Le cluster 1 regroupe les classes 1 et 2 lorsque la variation se produit tardivement (autour du pas de temps 100 sur 286) et le cluster 2 lorsqu'elle se produit plus tôt (autour du pas de temps 60 sur 286). Il en est de même pour les classes 3 et 4 avec les clusters 3 et 4. Sur cet ensemble de données, cette combinaison se comporte comme la distance euclidienne (avec un score NMI similaire de 0,52). Elle ne prend alors pas du tout en compte les motifs temporels. On peut remarquer que, dans ce cas, le rendu du pic peut sembler plus pertinent pour la reconstruction, ce qui peut expliquer pourquoi il est visible ici.

Nous avons suivi le même procédé pour le jeu de données CBF avec la figure 5.37. Cette fois ci nous avons utilisé la combinaison *DCNN-rec-None* avec un score de NMI de 0.35, contre 0.93 pour le résultat des figures 5.29 et 5.32. Le clustering obtenu résulte en une confusion



(a) Échantillon du cluster 1 (b) Échantillon du cluster 2 (c) Échantillon du cluster 3 (d) Échantillon du cluster 4

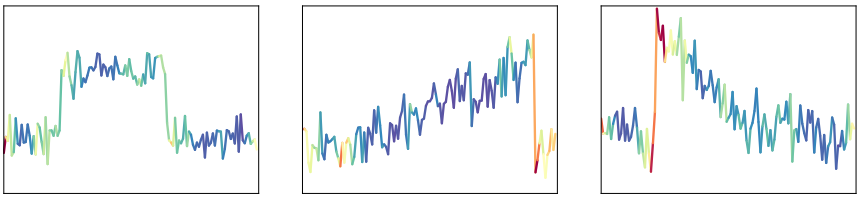


(e) Reconstruction du centre 1 (f) Reconstruction du centre 2 (g) Reconstruction du centre 3 (h) Reconstruction du centre 4

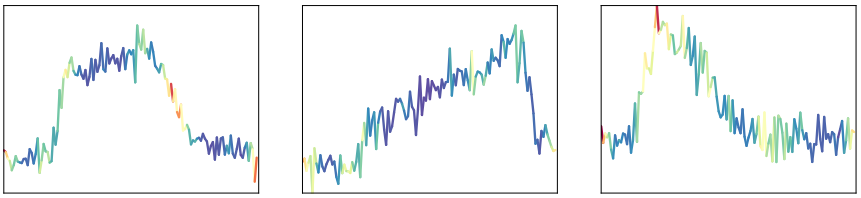
FIGURE 5.36 – Cartes de chaleur de Grad-CeAM sur le jeu de données Trace avvec la combinaison *DCNN-rec-None*, où 5.36a, 5.36b,5.36c et 5.36d sont des échantillons de chaque cluster 5.36e, 5.36f, 5.36g et 5.36h sont la construction des centres de chaque clusters. Les régions rouges correspondent à une forte contribution et les régions bleues à une contribution quasi nulle à la correspondance avec le centre (les couleurs sont lissées pour des raisons de clarté visuelle et pour mieux refléter la taille des filtres).

des classes 1 et 3. Cependant, contrairement à la reconstruction obtenue dans la figure 5.32 qui n'était que peut ressemblante, pour le modèle *DCNN-rec-None* la reconstruction est assez fidèle, notamment pour la classe 3. Ce cas particulier montre qu'afin d'obtenir une bonne reconstruction, l'encodeur peut avoir besoin d'enregistrer des informations inutiles qui interfèrent avec la tâche de clustering.

D'ailleurs, que ce soit pour Trace ou CBF, les derniers résultat montrent qu'une reconstruction plus fidèle n'aboutie pas nécessairement à une amélioration des résultats. Ces deux cas peuvent donc laisser penser que la tâche de reconstruction classique, qui est basée sur la distance euclidienne, n'est pas complètement appropriée à l'apprentissage de représentation pour les séries temporelles. Or, c'est la fonction de coût de prétexte qui a donné les résultats les plus robustes, avec son autre version *multi_rec*.



(a) Échantillon du cluster 1 (b) Échantillon du cluster 2 (c) Échantillon du cluster 3



(d) Reconstruction du centre 1 (e) Reconstruction du centre 2 (f) Reconstruction du centre 3

FIGURE 5.37 – Cartes de chaleur de Grad-CeAM sur le jeu de données CBF avec la combinaison *DCNN-rec-None*, où 5.37a, 5.37b et5.37c sont des échantillons de chaque cluster 5.37d, 5.37e et 5.37f sont la construction des centres de chaque clusters. Les régions rouges correspondent à une forte contribution et les régions bleues à une contribution quasi nulle à la correspondance avec le centre (les couleurs sont lissées pour des raisons de clarté visuelle et pour mieux refléter la taille des filtres).

## 5.4 Discussion : Avantages et limitations

L'étude que nous avons mené peut se résumer aux observations suivantes :

1. L'utilisation de la représentation par apprentissage profond conduit en moyenne à des améliorations de la performance du clustering.
2. Il existe une grande variabilité des résultats en fonction des différentes architectures, des fonctions de coût de prétexte et des fonctions de coût de clustering utilisées.
3. Les architectures basées sur l'utilisation de convolutions (CNN) semblent les plus appropriées pour l'apprentissage de caractéristiques pertinentes.
4. L'utilisation des fonctions de coût de clustering existantes ne semble pas pertinente pour les séries temporelles.
5. La fonction de coût de reconstruction classique, et sa variation *multi_rec*, restent les moyens les plus robustes d'obtenir une représentation appropriée pour la tâche de clustering.
6. Malgré l'observation précédente, la fonction de coût de reconstruction ne semble pas complètement adaptée à l'extraction des motifs temporels.
7. De multiples paramètres peuvent influencer les performances de clustering de la représentation apprise par les réseaux de neurones. Parmi eux, l'utilisation de la méthode de réduction de la dimension UMAP entraîne un gain de performance significatif et constant.
8. Aucune corrélation significative n'a pu être établie entre les caractéristiques des jeux de données, les paramètres ou les éléments des DNNs (architecture, type de fonction de coût).

Toutes les méthodes de clustering proposés dans la littérature pour l'analyse d'image se sont avérées moins efficaces qu'un autoencodeur classique sur les séries temporelles. Ces méthodes semblent perturber fortement les modèles, ce qui peut conduire à un clustering aberrant, par exemple composé d'un seul cluster. Il semble clairement impossible de les appliquer telles quelles sur des séries temporelles. Cela nécessiterait des adaptations supplémentaires ou

de nouvelles propositions pour prendre en compte les particularités de la dimension temporelle. La fonction de coût de reconstruction ne semble pas particulièrement adaptée aux séries temporelles, alors que c'est celle qui permet d'obtenir les meilleurs résultats de clustering.

Malgré ces observations, le meilleur modèle de clustering profond parvient à obtenir de bons résultats et à être classé premier parmi toutes les méthodes lorsqu'elle est combinée avec UMAP avec une réduction à $K$ (nombre de clusters) dimensions. Toutefois, il est important de noter que la méthode UMAP, lorsqu'elle est appliquée aux données originales avec la méthode Kmeans, se classe avant le meilleur modèle de clustering profond (sans UMAP). Comme UMAP recherche une projection à faible dimension des données qui préserve à la fois la structure locale et globale, elle préserve indirectement les caractéristiques les plus discriminantes. Par conséquent, cela peut suggérer que les DNNs parviennent à capturer des caractéristiques ou des motifs temporels importants, mais aussi une part du bruit, ce qui expliquerait la différence de performance. La méthode UMAP permet alors d'extraire les caractéristiques pertinentes, du moins les plus discriminantes. Les cartes de chaleur obtenues par la méthode Grad-CeAM renforce cette hypothèse, car elles montrent que les DNNs ont la capacité d'extraire des motifs temporels mais qu'ils peuvent aussi être partiellement perturbés par la fonction de coût d'entraînement.

Par conséquent, cette analyse nous a amené à la conclusion que le clustering profond pour les séries temporelles manque de fonctions de coût de prétextes appropriées. Les nouvelles fonctions de coût devraient mieux prendre en compte les spécificités de la dimension temporelle (c'est-à-dire les étirements ou décalages dans la série) et être moins sensibles aux variations faibles ou rares.

# Conclusion générale

## Contributions de cette thèse

Le clustering de séries temporelles, que celles-ci proviennent de la télédétection ou d'autres domaines, reste une tâche complexe. Cela est dû en partie à la complexité de l'analyse de la dimension temporelle. En effet, en plus d'apporter une dimension supplémentaire, les séries temporelles peuvent contenir des motifs décalés ou étirés dans le temps. Ainsi, cela peut générer une difficulté d'analyse par des méthodes classiques de clustering, difficulté amplifiée de par la nature mal-posée du problème de clustering.

Dans cette thèse, nous nous sommes attachés à répondre à ce problème par l'intégration de connaissances de l'expert sous la forme de contraintes dans le processus de clustering. Même si l'utilisation de méthodes classiques de clustering sous contraintes permet d'améliorer la qualité des résultats obtenus, notre première étude a montré que l'ajout de contraintes n'impliquait pas toujours une augmentation de la qualité et/ou robustesse des résultats. L'intégration de contraintes peut même amener une dégradation des résultats dans certains cas. Il est ressorti de notre étude que deux facteurs principaux influent sur l'impact des contraintes : une forte consistance des méthodes de clustering et une faible cohérence des contraintes. Nous avons décidé de répondre à ces deux verrous scientifiques par deux approches distinctes.

### L'incrémentalité, une réponse à la consistance

Une forte consistance d'une méthode de clustering vis à vis d'un jeu de contraintes indique la capacité de cette méthode à satisfaire un grand nombre des contraintes sans les utiliser explicitement dans le processus de clustering. Notre étude a montré que plus la consistance est élevée plus le gain à utiliser des contraintes est faible, voire négatif. Or, il ressort également de cette étude que la consistance liée à l'utilisation de contraintes générées aléatoirement est souvent forte. De plus, nous avons pu observer empiriquement que la consistance était également élevée lorsqu'on demande à un expert de donner des contraintes en se limitant à l'observation des données brutes.

Pour répondre à ce premier problème, nous avons proposé une méthode, nommée I-Samarah, basée sur le principe d'incrémentalité. La méthode repose sur un aller-retour entre la méthode de clustering et l'expert. L'expert utilise le résultat de clustering courant pour ajouter les contraintes qu'il juge pertinentes. La méthode prend en compte les nouvelles contraintes pour modifier le résultat courant en s'appuyant sur le processus collaboratif de la méthode de clustering sous contraintes Samarah.

Notre étude expérimentale sur des données de télédétection a montré les avantages de notre proposition, d'une part en une augmentation forte de la qualité des résultats et d'autre part, en une réduction importante de l'investissement de l'expert dans le processus de clustering.

Elle a aussi permis de mettre en évidence que c'est la conjonction de deux éléments qui permet ce gain significatif en qualité des résultats sur ces deux aspects. En effet, nous avons montré, dans un premier temps, que cette approche permet l'acquisition de contraintes plus discriminantes, les objets contraints se trouvant entre les couples de classes générant le plus de confusion. Dans un second temps, nous avons montré que la prise en compte du résultat

précédent permet d'assurer le plein potentiel des contraintes obtenues de cette manière.

Enfin, notre expérience sur les coupes franches d'arbres a permis de valider l'utilisation de notre approche sur un cas d'application réel. Notre approche permet l'obtention de résultats exploitables par l'expert pour un effort d'annotation raisonnable, malgré la présence de certaines limitations. En effet, nous avons constaté que les résultats avaient tendance à stagner voir se dégrader après un trop grand nombre d'incrémentations.

**L'apprentissage de représentation, une réponse à la cohérence**

Une faible cohérence de contraintes vis à vis d'une méthode de clustering indique la propension de cette méthode à satisfaire le maximum de ces contraintes. Cette notion est intrinsèquement liée à la manière dont la (dis)similarité entre les différents objets du jeu de données est mesurée. En effet, il est difficile d'assurer la satisfaction de contraintes qui se contredisent ou qui ne sont pas en accord avec la structure de l'espace des données. Notre étude a montré que plus la cohérence est élevée plus les gains en qualité des résultats de clustering seront faibles, voir négatifs.

Pour répondre à ce second problème, nous avons décidé de nous intéresser à l'apprentissage de représentation par apprentissage profond (les méthodes de clustering profond). Dans un premier temps nous avons testé les méthodes existantes permettant l'intégration de contraintes dans l'apprentissage de représentation. L'objectif est alors d'apprendre une représentation qui permette de maximiser la satisfaction des contraintes et donc indirectement leur cohérence. Cependant, nos premières expériences ont fait ressortir des résultats mitigés et non-exploitables en pratique. Toutefois, nous avons également montré que ces mauvais résultats ne sont pas liés uniquement à la prise en compte des contraintes, mais également à la méthode de clustering profond sous-jacente.

Cette dernière observation nous a motivé à conduire une analyse plus poussée sur les capacités des méthodes de clustering profond à capturer des caractéristiques représentatives et utiles à la tâche de clustering de séries temporelles. En conséquence, nous avons mené une étude comparative sur différentes méthodes existantes. Pour cela, nous avons caractérisé chaque méthode par trois éléments constituants de telles méthodes : l'architecture, la fonction de coût de prétexte et la fonction de coût de clustering. Nous avons évalué un ensemble de 300 combinaisons sur un large ensemble de jeux de données (128 univariés et 30 multivariés) afin de mieux faire ressortir les influences de chaque élément pris séparément. Il ressort de cette étude que même si certaines combinaisons permettent d'obtenir des résultats compétitifs, les gains en qualité restent limités. Néanmoins, de manière plus générale, nous avons constaté que les fonctions de coût de clustering existantes ne semblent pas adaptées à l'analyse de séries temporelles, car elles mènent en moyenne à une dégradation de la qualité des résultats. Nous avons également constaté de meilleurs résultats pour les combinaisons utilisant des convolutions et celles utilisant des fonctions de coût de prétexte à base de reconstruction. Cependant, il ressort également une grande difficulté à mettre en évidence des corrélations entre l'utilisation de certains paramètres et les caractéristiques des jeu de données utilisés. De plus, même si certaines tendances ressortent, nous avons pu constater une grande influence du choix de certains hyperparamètres sur la qualité finale des résultats, rendant la conception de ces modèles complexe en pratique.

Afin de mieux comprendre ces résultats et de répondre au problème d'interprétabilité, liée à l'utilisation de réseau de neurones, nous avons proposé une nouvelle méthode appelée Grad-CeAM (grad-Centroid Activation Mapping). Cette méthode permet de visualiser les zones ayant servi dans la décision d'affectation à un cluster particulier. Or, il ressort de l'analyse des visualisations obtenues que les méthodes de reconstruction ne sont pas toujours aptes à bien prendre en compte les décalages et étirements d'un même motif temporel, malgré le fait que ce soient celles qui donnent les résultats les plus robustes. Cela nous mène donc à penser que les méthodes de deep clustering pour les séries temporelles bénéficierait fortement d'une fonction de coût de prétexte adaptée.

## Discussion et perspectives

Dans cette thèse, nous avons abordé le problème de consistance, et de cohérence des contraintes de manière séparée. Toutefois, nous nous sommes orientés sur des méthodes suffisamment génériques pour pouvoir être intégrées au sein d'un processus général de clustering. Nous avions comme perspective de pouvoir évaluer le gain à combiner ces deux approches. Malheureusement, même si les travaux de cette thèse proposent une réponse au problème de consistance, le problème de cohérence, quant-à-lui, reste ouvert.

Ainsi, même si le clustering par apprentissage profond suscite un intérêt croissant, les travaux portant sur les spécificités du clustering de séries temporelles restent très restreint. Nos dernières recherches se sont portées sur la conception d'une fonction de coût de prétexte apte à combler ce manque. Malheureusement, nos premières expériences portant sur une modification de la fonction de coût de triplet par génération d'exemples positifs (en utilisant des transformations par décalage ou étirement) ne se sont pas avérées concluantes. Cependant, d'autres pistes restent à explorer, telle que l'utilisation d'une version modifiée de la fonction de coût de reconstruction avec une série alignée. Ainsi, l'objectif est de pouvoir modifier les fonctions de coût prétextes existantes pour qu'elles puissent générer des modèles plus invariants aux déformations temporelles.

Enfin, nous faisons l'hypothèse qu'un manque de cohérence des contraintes est à l'origine des limitations de notre méthode I-Samarah lorsque le nombre de contraintes ou d'incrémentation augmente. Toutefois, nous pensons que ce n'est pas le seul vecteur d'amélioration de notre méthode. En effet, même si l'utilisation I-Samarah permet de faciliter le processus d'annotation pour l'expert, un mécanisme de suggestion de contraintes ou à minima d'évaluation (donner une note sur l'efficacité potentielle d'une contrainte) pourrait faciliter cette tâche. Cela permettrait de faire passer notre méthode, d'une approche purement incrémentale à une approche active. Cependant, le problème d'évaluer a priori l'informativité d'une contrainte n'est pas trivial. Nos premières études, impliquant notamment des critères tels que le score de silhouette ou de distance au centre, ne sont pas probantes. La difficulté d'une telle évaluation réside essentiellement dans le fait de trouver une contrainte qui modifie le clustering sans trop fortement le perturber. Ainsi, une contrainte doit modifier le résultat de clustering pour que celui-ci puisse prendre en compte l'information donnée par l'utilisateur à travers la contrainte, mais cette modification ne doit pas perturber de manière trop importante le clustering. En effet, une trop grande modification peut impliquer une dégradation de la qualité du résultat global.

En conclusion, l'expert étant au centre du processus, il est indispensable que l'ajout d'une contrainte modifie le résultat (l'expert veut que son travail paie) mais que le résultat reste proche de l'ancien afin que l'effort cognitif pour l'appréhender soit raisonnable. Il reste à notre avis, encore de nombreuses verrous scientifiques à lever avant d'arriver à résoudre paradoxe.

# Bibliographie


Abbasimehr, H. et M. Shabani. 2019, «A new methodology for customer behavior analysis using time series clustering», *Kybernetes*. 1

Aghabozorgi, S., A. S. Shirkhorshidi et T. Y. Wah. 2015, «Time-series clustering–a decade review», *Information Systems*, vol. 53, p. 16–38. 7, 8

Al-Razgan, M. et C. Domeniconi. 2009, «Clustering ensembles with active constraints», dans *Applications of Supervised and Unsupervised Ensemble Methods*, Springer, p. 175–189. 14, 15

Anand, S. S., D. A. Bell et J. G. Hughes. 1995, «The role of domain knowledge in data mining», dans *Proceedings of the fourth international conference on Information and knowledge management*, p. 37–43. 12

Aref, W. G., M. G. Elfeky et A. K. Elmagarmid. 2004, «Incremental, online, and merge mining of partial periodic patterns in time-series databases», *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, n° 3, p. 332–342. 7

Audebert, N., B. Le Saux et S. Lefèvre. 2019, «Deep learning for classification of hyperspectral data : A comparative review», *IEEE geoscience and remote sensing magazine*, vol. 7, n° 2, p. 159–173. 48

Babaki, B. 2017, «MIPKmeans», URL https://github.com/Behrouz-Babaki/MIPKmeans. 44

Babaki, B., T. Guns et S. Nijssen. 2014, «Constrained clustering using column generation», dans *CPAIOR*, p. 438–454. 14

Bagnall, A., H. A. Dau, J. Lines, M. Flynn, J. Large, A. Bostrom, P. Southam et E. Keogh. 2018, «The uea multivariate time series classification archive, 2018», *arXiv preprint arXiv :1811.00075*. 98

Bagnall, A., J. Lines, A. Bostrom, J. Large et E. Keogh. 2017, «The great time series classification bake off : a review and experimental evaluation of recent algorithmic advances», *Data Mining and Knowledge Discovery*, vol. 31, n° 3, p. 606–660. 20

Bahdanau, D., K. Cho et Y. Bengio. 2014, «Neural machine translation by jointly learning to align and translate», *arXiv preprint arXiv :1409.0473*. 82

Bair, E. 2013, «Semi-supervised clustering methods», *Wiley Interdisciplinary Reviews : Computational Statistics*, vol. 5, n° 5, p. 349–361. 9

Ballard, D. H. 1987, «Modular learning in neural networks.», dans *AAAI*, p. 279–284. 83

Bar-Hillel, A., T. Hertz, N. Shental, D. Weinshall et G. Ridgeway. 2005, «Learning a mahalanobis metric from equivalence constraints.», *Journal of machine learning research*, vol. 6, n° 6. 13

Basu, S., A. Banerjee et R. Mooney. 2002, «Semi-supervised clustering by seeding», dans *In Proceedings of 19th International Conference on Machine Learning (ICML-2002*, Citeseer. 13

Basu, S., A. Banerjee et R. J. Mooney. 2004, «Active semi-supervision for pairwise constrained clustering», dans *Proceedings of the 2004 SIAM international conference on data mining*, SIAM, p. 333–344. 13, 32

Basu, S., I. Davidson et K. Wagstaff. 2008, *Constrained clustering : Advances in algorithms, theory, and applications*, CRC Press. 12

Becker, S. 1991, «Unsupervised learning procedures for neural networks», *International Journal of Neural Systems*, vol. 2, n° 01n02, p. 17–33. 83

Belgiu, M. et O. Csillik. 2018, «Sentinel-2 cropland mapping using pixel-based and object-based time-weighted dynamic time warping analysis», *Remote sensing of environment*, vol. 204, p. 509–523. 41

Bengio, Y., P. Simard et P. Frasconi. 1994, «Learning long-term dependencies with gradient descent is difficult», *IEEE transactions on neural networks*, vol. 5, n° 2, p. 157–166. 80

Bo, D., X. Wang, C. Shi, M. Zhu, E. Lu et P. Cui. 2020, «Structural deep clustering network», dans *Proceedings of The Web Conference 2020*, p. 1400–1410. 67, 88, 89, 97, 98

Briandet, R., E. K. Kemsley et R. H. Wilson. 1996, «Discrimination of arabica and robusta in instant coffee by fourier transform infrared spectroscopy and chemometrics», *Journal of agricultural and food chemistry*, vol. 44, n° 1, p. 170–174. 116

Bucha, T. et H.-J. Stibig. 2008, «Analysis of modis imagery for detection of clear cuts in the boreal forest in north-west russia», *Remote Sensing of Environment*, vol. 112, n° 5, p. 2416–2429. 43

Caron, M., P. Bojanowski, A. Joulin et M. Douze. 2018, «Deep clustering for unsupervised learning of visual features», dans *Proceedings of the European Conference on Computer Vision (ECCV)*, p. 132–149. 66, 90, 97

Chang, S., Y. Zhang, W. Han, M. Yu, X. Guo, W. Tan, X. Cui, M. Witbrock, M. A. Hasegawa-Johnson et T. S. Huang. 2017, «Dilated recurrent neural networks», dans *Advances in Neural Information Processing Systems*, p. 77–87. 82

Chen, Y., E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen et G. Batista. 2015, «The ucr time series classification archive», www.cs.ucr.edu/~eamonn/time_series_data/. 22

Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk et Y. Bengio. 2014, «Learning phrase representations using rnn encoder-decoder for statistical machine translation», *arXiv preprint arXiv :1406.1078*. 81

Claeys, E., P. Gancarski, M. Maumy-Bertrand et H. Wassner. 2020, «Dynamic allocation optimization in a/b tests using classification-based preprocessing», *hal-01874969v3*. 1

Coates, A., A. Ng et H. Lee. 2011, «An analysis of single-layer networks in unsupervised feature learning», dans *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, p. 215–223. 115

Cohen, W., M. Fiorella, J. Gray, E. Helmer et K. Anderson. 1998, «An efficient and accurate method for mapping forest clearcuts in the pacific northwest using landsat imagery», *Photogramm Eng Remote Sensing*, vol. 64, p. 293–299. 42

Cohn, D., R. Caruana et A. McCallum. 2003, «Semi-supervised clustering with user feedback», *Constrained Clustering : Advances in Algorithms, Theory, and Applications*, vol. 4, n° 1, p. 17–32. 13, 31

Craddock, R. C., G. A. James, P. E. Holtzheimer III, X. P. Hu et H. S. Mayberg. 2012, «A whole brain fmri atlas generated via spatially constrained spectral clustering», *Human brain mapping*, vol. 33, n° 8, p. 1914–1928. 1

Dao, T.-B.-H., K.-C. Duong, C. Vrain et collab.. 2017, «Constrained clustering by constraint programming», *Artificial Intelligence*, vol. 244, p. 70–94. 14, 21

Dau, H. A., A. Bagnall, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana et E. Keogh. 2019, «The ucr time series archive», *IEEE/CAA Journal of Automatica Sinica*, vol. 6, n° 6, p. 1293–1305. 6, 97, 98, 100

Daubechies, I. 1992, *Ten lectures on wavelets*, SIAM. 7

Davidson, I. et S. Ravi. 2005, «Clustering with constraints : Feasibility issues and the k-means algorithm», dans *Proceedings of the 2005 SIAM international conference on data mining*, SIAM, p. 138–149. 12, 13

Davidson, I., S. Ravi et M. Ester. 2007, «Efficient incremental constrained clustering», dans *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, p. 240–249. 31

Davidson, I., S. Ravi et L. Shamis. 2010, «A SAT-based framework for efficient constrained clustering», dans *SDM*, p. 94–105. 14

Davidson, I., K. L. Wagstaff et S. Basu. 2006, «Measuring constraint-set utility for partitional clustering algorithms», dans *European conference on principles of data mining and knowledge discovery*, Springer, p. 115–126. 13, 27, 29

Dempster, A., F. Petitjean et G. I. Webb. 2020, «Rocket : Exceptionally fast and accurate time series classification using random convolutional kernels», *Data Mining and Knowledge Discovery*, p. 1–42. 66

Demšar, J. 2006, «Statistical comparisons of classifiers over multiple data sets», *Journal of Machine learning research*, vol. 7, n° Jan, p. 1–30. 100

Doersch, C., A. Gupta et A. A. Efros. 2015, «Unsupervised visual representation learning by context prediction», dans *Proceedings of the IEEE international conference on computer vision*, p. 1422–1430. 83

Domany, E. 1999, «Superparamagnetic clustering of data—the definitive solution of an ill-posed problem», *Physica A : Statistical Mechanics and its Applications*, vol. 263, n° 1-4, p. 158–169. 8

Faloutsos, C., M. Ranganathan et Y. Manolopoulos. 1994, «Fast subsequence matching in time-series databases», *Acm Sigmod Record*, vol. 23, n° 2, p. 419–429. 7

Fawaz, H. I., G. Forestier, J. Weber, L. Idoumghar et P.-A. Muller. 2019, «Deep learning for time series classification : a review», *Data Mining and Knowledge Discovery*, vol. 33, n° 4, p. 917–963. 66, 70, 79, 99

Forestier, G., P. Gançarski et C. Wemmert. 2010a, «Collaborative clustering with background knowledge», *Data & Knowledge Engineering*, vol. 69, n° 2, p. 211–228. 15, 16, 18, 22

Forestier, G. et C. Wemmert. 2016, «Semi-supervised learning using multiple clusterings with limited labeled data», *Information Sciences*, vol. 361, p. 48–65. 1

Forestier, G., C. Wemmert et P. Gancarski. 2010b, «Towards conflict resolution in collaborative clustering», dans *2010 5th IEEE International Conference Intelligent Systems*, IEEE, p. 361–366. 17

Franceschi, J.-Y., A. Dieuleveut et M. Jaggi. 2019, «Unsupervised scalable representation learning for multivariate time series», dans *Advances in Neural Information Processing Systems*, p. 4652–4663. 87, 95, 97, 98, 99

Fred, A. et A. Jain. 2002, «Data clustering using evidence accumulation», *ICPR*, p. 276––280. 15

Gançarski, P. et C. Wemmert. 2005, «Collaborative multi-strategy classification : application to per-pixel analysis of images», dans *Proceedings of the 6th international workshop on Multimedia data mining : mining integrated media and complex data*, p. 15–22. 16, 18

Gançarski, P. et C. Wemmert. 2007, «Collaborative multi-step mono-level multi-strategy classification», *MTAP*, vol. 35, n° 1, p. 1–27. 15

Gers, F. A., J. Schmidhuber et F. Cummins. 2000, «Learning to forget : Continual prediction with lstm», *Neural Computation*, vol. 12, n° 10, p. 2451–2471. 81

Ghasedi, K., X. Wang, C. Deng et H. Huang. 2019, «Balanced self-paced learning for generative adversarial clustering network», dans *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, p. 4391–4400. 86, 98

Ghasedi Dizaji, K., A. Herandi, C. Deng, W. Cai et H. Huang. 2017, «Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization», dans *Proceedings of the IEEE international conference on computer vision*, p. 5736–5745. 66, 90, 96, 97, 99, 102

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville et Y. Bengio. 2014, «Generative adversarial nets», dans *Advances in neural information processing systems*, p. 2672–2680. 85

Grabska, E., P. Hostert, D. Pflugmacher et K. Ostapowicz. 2019, «Forest stand species mapping using the sentinel-2 time series», *Remote Sensing*, vol. 11, n° 10, p. 1197. 40

Guan, Q., Y. Huang, Z. Zhong, Z. Zheng, L. Zheng et Y. Yang. 2018, «Diagnose like a radiologist : Attention guided convolutional neural network for thorax disease classification», *arXiv preprint arXiv :1801.09927*. 82

Guo, X., L. Gao, X. Liu et J. Yin. 2017a, «Improved deep embedded clustering with local structure preservation.», dans *IJCAI*, p. 1753–1759. 66, 67, 68, 79, 83, 88, 89, 93, 94, 97, 102

Guo, X., X. Liu, E. Zhu et J. Yin. 2017b, «Deep clustering with convolutional autoencoders», dans *International conference on neural information processing*, Springer, p. 373–382. 89, 97, 99, 102

Guo, Y. et R. Greiner. 2007, «Optimistic active-learning using mutual information.», dans *IJCAI*, vol. 7, p. 823–829. 32

Hadjitodorov, S. T. et L. I. Kuncheva. 2007, «Selecting diversifying heuristics for cluster ensembles», dans *International Workshop on Multiple Classifier Systems*, Springer, p. 200–209. 15

Han, W., R. Feng, L. Wang et Y. Cheng. 2018, «A semi-supervised generative framework with deep learning features for high-resolution remote sensing image scene classification», *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 145, p. 23–43. 1

He, K., X. Zhang, S. Ren et J. Sun. 2016, «Deep residual learning for image recognition», dans *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 770–778. 95, 111

Hochreiter, S. et J. Schmidhuber. 1997, «Long short-term memory», *Neural computation*, vol. 9, n° 8, p. 1735–1780. 80, 81

Hoi, S. C., W. Liu et S.-F. Chang. 2010, «Semi-supervised distance metric learning for collaborative image retrieval and clustering», *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 6, n° 3, p. 1–26. 14

Holm, S. 1979, «A simple sequentially rejective multiple test procedure», *Scandinavian journal of statistics*, p. 65–70. 100

Hopfield, J. J. 1982, «Neural networks and physical systems with emergent collective computational abilities», *Proceedings of the national academy of sciences*, vol. 79, n° 8, p. 2554–2558. 80

Hubert, L. et P. Arabie. 1985, «Comparing partitions», *Journal of classification*, vol. 2, n° 1, p. 193–218. 23

Ienco, D., R. Interdonato, R. Gaetano et D. H. T. Minh. 2019, «Combining sentinel-1 and sentinel-2 satellite image time series for land cover mapping via a multi-source deep learning architecture», *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 158, p. 11–22. 40

Ienco, D. et R. G. Pensa. 2019, «Deep triplet-driven semi-supervised embedding clustering», dans *International Conference on Discovery Science*, Springer, p. 220–234. 68, 82

Jaderberg, M., K. Simonyan, A. Zisserman et K. Kavukcuoglu. 2015, «Spatial transformer networks», *arXiv preprint arXiv :1506.02025*. 82

Jain, A. K., M. N. Murty et P. J. Flynn. 1999, «Data clustering : a review», *ACM computing surveys (CSUR)*, vol. 31, n° 3, p. 264–323. 8

Jiang, Z., Y. Zheng, H. Tan, B. Tang et H. Zhou. 2016, «Variational deep embedding : A generative approach to clustering», *CoRR*. 67, 85, 97

Jiao, Y., K. Yang, S. Dou, P. Luo, S. Liu et D. Song. 2020, «Timeautoml : Autonomous representation learning for multivariate irregularly sampled time series», *arXiv preprint arXiv :2010.01596*. 83

Kamvar, K., S. Sepandar, K. Klein, D. Dan, M. Manning et C. Christopher. 2003, «Spectral learning», dans *International Joint Conference of Artificial Intelligence*, Stanford InfoLab. 14, 20, 21

Kane, M. J., N. Price, M. Scotch et P. Rabinowitz. 2014, «Comparison of arima and random forest time series models for prediction of avian influenza h5n1 outbreaks», *BMC bioinformatics*, vol. 15, n° 1, p. 276. 48

Keogh, E. et S. Kasetty. 2003, «On the need for time series data mining benchmarks : a survey and empirical demonstration», *Data Mining and knowledge discovery*, vol. 7, n° 4, p. 349–371. 6, 7

Keogh, E., S. Lonardi, C. A. Ratanamahatana, L. Wei, S.-H. Lee et J. Handley. 2007, «Compression-based data mining of sequential data», *Data Mining and Knowledge Discovery*, vol. 14, n° 1, p. 99–129. 7

Kingma, D. P. et M. Welling. 2013, «Auto-encoding variational bayes», *arXiv preprint arXiv :1312.6114*. 85

Kipf, T. N. et M. Welling. 2016, «Semi-supervised classification with graph convolutional networks», *arXiv preprint arXiv :1609.02907*. 89

Kittler, J., M. Hatef, R. P. Duin et J. Matas. 1998, «On combining classifiers», *IEEE transactions on pattern analysis and machine intelligence*, vol. 20, n° 3, p. 226–239. 15

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», cahier de recherche, Stanford. 13

Kramer, M. A. 1991, «Nonlinear principal component analysis using autoassociative neural networks», *AIChE journal*, vol. 37, n° 2, p. 233–243. 83

Krizhevsky, A., I. Sutskever et G. E. Hinton. 2012, «Imagenet classification with deep convolutional neural networks», dans *Advances in neural information processing systems*, p. 1097–1105. 66, 79

Lacroix, P., G. Bièvre, E. Pathier, U. Kniess et D. Jongmans. 2018, «Use of sentinel-2 images for the detection of precursory motions before landslide failures», *Remote Sensing of Environment*, vol. 215, p. 507–516. 40

Lampert, T., B. Lafabregue, N. Serrette, G. Forestier, B. Crémilleux, C. Vrain, P. Gancarski et collab.. 2018, «Constrained distance based clustering for time-series : a comparative and experimental study», *Data Mining and Knowledge Discovery*, vol. 32, n° 6, p. 1663–1707. 19, 20

Lampert, T., B. Lafabregue, N. Serrette, C. Vrain, P. Gançarski et collab.. 2019, «Constrained distance-based clustering for satellite image time-series», *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 12, n° 11, p. 4606–4621. 41, 48, 49

Larsson, G., M. Maire et G. Shakhnarovich. 2017, «Colorization as a proxy task for visual understanding», dans *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, p. 6874–6883. 83

Laxman, S. et P. Sastry. 2006, «A survey of temporal data mining», *Sadhana*, vol. 31, n° 2, p. 173–198. 7

LeCun, Y., L. Bottou, Y. Bengio et P. Haffner. 1998, «Gradient-based learning applied to document recognition», *Proceedings of the IEEE*, vol. 86, n° 11, p. 2278–2324. 66, 67, 79, 115

LeCun, Y., K. Kavukcuoglu et C. Farabet. 2010, «Convolutional networks and applications in vision», dans *Proceedings of 2010 IEEE international symposium on circuits and systems*, IEEE, p. 253–256. 111

Lewis, D. D. et W. A. Gale. 1994, «A sequential algorithm for training text classifiers», dans *SIGIR'94*, Springer, p. 3–12. 31

Li, X., Z. Chen, L. K. Poon et N. L. Zhang. 2018, «Learning latent superstructures in variational autoencoders for deep multidimensional clustering», dans *International Conference on Learning Representations*. 85

Li, Z., J. Liu et X. Tang. 2009, «Constrained clustering via spectral regularization», dans *2009 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, p. 421–428. 14, 21

Lin, J., E. Keogh, L. Wei et S. Lonardi. 2007, «Experiencing sax : a novel symbolic representation of time series», *Data Mining and knowledge discovery*, vol. 15, n° 2, p. 107–144. 7

Lines, J. et A. Bagnall. 2015, «Time series classification with ensembles of elastic distance measures», *Data Min Knowl Discov*, vol. 29, n° 3, p. 565–592. 20

Lipton, Z. C. 2018, «The mythos of model interpretability : In machine learning, the concept of interpretability is both important and slippery.», *Queue*, vol. 16, n° 3, p. 31–57. 111

Lipton, Z. C. et S. Tripathi. 2017, «Precise recovery of latent vectors from generative adversarial networks», *arXiv preprint arXiv :1702.04782*. 86

Lkhagva, B., Y. Suzuki et K. Kawagoe. 2006, «New time series data representation esax for financial applications», dans *22nd International Conference on Data Engineering Workshops (ICDEW'06)*, IEEE, p. x115–x115. 8

Lobo, F. d. L., P. W. M. Souza-Filho, E. M. L. d. M. Novo, F. M. Carlos et C. C. F. Barbosa. 2018, «Mapping mining areas in the brazilian amazon using msi/sentinel-2 imagery (2017)», *Remote Sensing*, vol. 10, n° 8, p. 1178. 40

Ma, Q., S. Li, W. Zhuang, J. Wang et D. Zeng. 2020, «Self-supervised time series clustering with model-based dynamics», *IEEE Transactions on Neural Networks and Learning Systems*. 106

Ma, Q., J. Zheng, S. Li et G. W. Cottrell. 2019, «Learning representations for time series clustering», dans *Advances in Neural Information Processing Systems*, p. 3776–3786. 67, 88, 92, 93, 96, 98, 99

Maaten, L. v. d. et G. Hinton. 2008, «Visualizing data using t-sne», *Journal of machine learning research*, vol. 9, n° Nov, p. 2579–2605. 104, 108

Madiraju, N. S., S. M. Sadat, D. Fisher et H. Karimabadi. 2018, «Deep temporal clustering : Fully unsupervised learning of time-domain features», *arXiv preprint arXiv :1802.01059*. 92, 95, 98, 99

Makhzani, A. et B. Frey. 2013, «K-sparse autoencoders», *arXiv preprint arXiv :1312.5663*. 84

Mallapragada, P. K., R. Jin et A. K. Jain. 2008, «Active query selection for semi-supervised clustering», dans *2008 19th International Conference on Pattern Recognition*, IEEE, p. 1–4. 45

McConville, R., R. Santos-Rodriguez, R. J. Piechocki et I. Craddock. 2019, «N2d :(not too) deep clustering via clustering the local manifold of an autoencoded embedding», *arXiv preprint arXiv :1908.05968*. 104

McInnes, L., J. Healy et J. Melville. 2018, «Umap : Uniform manifold approximation and projection for dimension reduction», *arXiv preprint arXiv :1802.03426*. 104, 108

Mohammad, Y. et T. Nishida. 2009, «Constrained motif discovery in time series», *New Generation Computing*, vol. 27, n° 4, p. 319. 66

Moon, Y.-S., K.-Y. Whang et W.-S. Han. 2002, «General match : a subsequence matching method in time-series databases based on generalized windows», dans *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, p. 382–393. 7

Mukherjee, S., H. Asnani, E. Lin et S. Kannan. 2019, «Clustergan : Latent space clustering in generative adversarial networks», dans *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, p. 4610–4617. 86, 98

Ng, A., M. Jordan et Y. Weiss. 2001, «On spectral clustering : analysis and an algorithm», dans *Proceedings of the International Conference on Neural Information Processing Systems*, p. 849–856. 24

Oliveira, J. F., M. S. Rodrigues, L. M. Skalinski, A. E. Santos, L. C. Costa, L. L. Cardim, E. S. Paixão, M. d. C. N. Costa, W. K. Oliveira, M. L. Barreto et collab.. 2020, «Interdependence between confirmed and discarded cases of dengue, chikungunya and zika viruses in brazil : A multivariate time-series analysis», *PloS one*, vol. 15, n° 2, p. e0228 347. 1

Ottosen, T.-B., S. T. Lommen et C. A. Skjøth. 2019, «Remote sensing of cropping practice in northern italy using time-series from sentinel-2», *Computers and Electronics in Agriculture*, vol. 157, p. 232–238. 40

Panuccio, A., M. Bicego et V. Murino. 2002, «A hidden markov model-based approach to sequential data clustering», dans *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, Springer, p. 734–743. 7

Paparrizos, J. et L. Gravano. 2015, «k-shape : Efficient and accurate clustering of time series», dans *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, p. 1855–1870. 98, 106

Pascanu, R., C. Gulcehre, K. Cho et Y. Bengio. 2013, «How to construct deep recurrent neural networks», *arXiv preprint arXiv :1312.6026*. 82

Pedrycz, W. 2002, «Collaborative fuzzy clustering», *Pattern Recognition Letters*, vol. 23, n° 14, p. 1675–1686. 15

Pelleg, D. et D. Baras. 2007, «K-means with large and noisy constraint sets», dans *European Conference on Machine Learning*, Springer, p. 674–682. 13

Pereira, F., N. Tishby et L. Lee. 1993, «Distributional clustering of english words», dans *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, Association for Computational Linguistics, p. 183–190. 31

Petitjean, F., A. Ketterlin et P. Gançarski. 2011, «A global averaging method for dynamic time warping, with applications to clustering», *Pattern Recognition*, vol. 44, n° 3, p. 678–693. 20, 105, 122

Rand, W. M. 1971, «Objective criteria for the evaluation of clustering methods», *Journal of the American Statistical association*, vol. 66, n° 336, p. 846–850. 57

Ren, Y., K. Hu, X. Dai, L. Pan, S. C. Hoi et Z. Xu. 2019, «Semi-supervised deep embedded clustering», *Neurocomputing*, vol. 325, p. 121–130. 68

Rhif, M., A. Ben Abbes, I. R. Farah, B. Martínez et Y. Sang. 2019, «Wavelet transform application for/in non-stationary time-series analysis : a review», *Applied Sciences*, vol. 9, n° 7, p. 1345. 7

Rifai, S., P. Vincent, X. Muller, X. Glorot et Y. Bengio. 2011, «Contractive auto-encoders : Explicit invariance during feature extraction», dans *Icml*. 84, 85

Rosenblatt, F. 1958, «The perceptron : a probabilistic model for information storage and organization in the brain.», *Psychological review*, vol. 65, n° 6, p. 386. 78

Rosenkrantz, D. J., R. E. Stearns et P. M. Lewis, II. 1977, «An analysis of several heuristics for the traveling salesman problem», *SIAM journal on computing*, vol. 6, n° 3, p. 563–581. 45

Rousseeuw, P. 1987, «Silhouettes : a graphical aid to the interpretation and validation of cluster analysis», *Computational and Applied Mathematics*, vol. 20, p. 53–65. 27

Roweis, S. T. et L. K. Saul. 2000, «Nonlinear dimensionality reduction by locally linear embedding», *science*, vol. 290, n° 5500, p. 2323–2326. 104

Roy, N. et A. McCallum. 2001, «Toward optimal active learning through sampling estimation of error reduction. int. conf. on machine learning», . 32

Rumelhart, D. E., G. E. Hinton et R. J. Williams. 1986, «Learning representations by back-propagating errors», *nature*, vol. 323, n° 6088, p. 533–536. 71, 83

Russwurm, M. et M. Körner. 2018, «Multi-temporal land cover classification with sequential recurrent encoders», *ISPRS International Journal of Geo-Information*, vol. 7, n° 4, p. 129. 40

Ryan, S., N. Carlson, H. Butler, T. Fingerlin, L. Maier et F. Xing. 2020, «Cluster activation mapping with applications to medical imaging», *arXiv preprint arXiv :2010.04794*. 113

Saito, N. 1994, *Local feature extraction and its applications using a library of bases*, thèse de doctorat, Yale University. 116

Sak, H., A. Senior et F. Beaufays. 2014, «Long short-term memory recurrent neural network architectures for large scale acoustic modeling», dans *Fifteenth Annual Conference of the International Speech Communication Association*. 80

Sakamoto, T., M. Yokozawa, H. Toritani, M. Shibayama, N. Ishitsuka et H. Ohno. 2005, «A crop phenology detection method using time-series modis data», *Remote sensing of environment*, vol. 96, n° 3-4, p. 366–374. 41

Sakoe, H. et S. Chiba. 1978, «Dynamic programming algorithm optimization for spoken word recognition», *IEEE transactions on acoustics, speech, and signal processing*, vol. 26, n° 1, p. 43–49. 7, 20, 105

Saksa, T., J. Uuttera, T. Kolström, M. Lehikoinen, A. Pekkarinen et V. Sarvi. 2003, «Clear-cut detection in boreal forest aided by remote sensing», *Scandinavian Journal of Forest Research*, vol. 18, p. 537–546. 42, 43

Salaou, A.-D. 2020, *Metric learning for multivariate time series analysis using DTW : application to remote sensing and software engineering*, thèse de doctorat. 14

Schroff, F., D. Kalenichenko et J. Philbin. 2015, «Facenet : A unified embedding for face recognition and clustering», dans *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 815–823. 87

Selvaraju, R. R., M. Cogswell, A. Das, R. Vedantam, D. Parikh et D. Batra. 2017, «Grad-cam : Visual explanations from deep networks via gradient-based localization», dans *Proceedings of the IEEE international conference on computer vision*, p. 618–626. 111

Settles, B. 2009, «Active learning literature survey», cahier de recherche, University of Wisconsin-Madison Department of Computer Sciences. 31

Settles, B. et M. Craven. 2008, «An analysis of active learning strategies for sequence labeling tasks», dans *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, p. 1070–1079. 31

Settles, B., M. Craven et S. Ray. 2008, «Multiple-instance active learning», dans *Advances in neural information processing systems*, p. 1289–1296. 32

Sezer, O. B., M. U. Gudelek et A. M. Ozbayoglu. 2020, «Financial time series forecasting with deep learning : A systematic literature review : 2005–2019», *Applied Soft Computing*, vol. 90, p. 106 181. 1

Shahriari, B., K. Swersky, Z. Wang, R. P. Adams et N. De Freitas. 2015, «Taking the human out of the loop : A review of bayesian optimization», *Proceedings of the IEEE*, vol. 104, n° 1, p. 148–175. 83

Shi, J. et J. Malik. 2000, «Normalized cuts and image segmentation», *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, n° 8, p. 888–905. 14

Souza, T. V. et C. Zanchettin. 2019, «Improving deep image clustering with spatial transformer layers», dans *International Conference on Artificial Neural Networks*, Springer, p. 641–654. 82

Strehl, A. et J. Ghosh. 2002, «Cluster ensembles—a knowledge reuse framework for combining multiple partitions», *JMLR*, vol. 3, p. 583–617. 15

Sun, D., J. Wulff, E. B. Sudderth, H. Pfister et M. J. Black. 2013, «A fully-connected layered model of foreground and background flow», dans *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, p. 2451–2458. 79

Sutskever, I., O. Vinyals et Q. V. Le. 2014, «Sequence to sequence learning with neural networks», dans *Advances in neural information processing systems*, p. 3104–3112. 80

Tatsumi, K., Y. Yamashiki, M. A. C. Torres et C. L. R. Taipe. 2015, «Crop classification of upland fields using random forest of time-series landsat 7 etm+ data», *Computers and Electronics in Agriculture*, vol. 115, p. 171–179. 48

Tenenbaum, J. B., V. De Silva et J. C. Langford. 2000, «A global geometric framework for nonlinear dimensionality reduction», *science*, vol. 290, n° 5500, p. 2319–2323. 104

Togami, M. 2019, «Spatial constraint on multi-channel deep clustering», dans *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, p. 531–535. 68

Van Craenendonck, T., W. Meert, S. Dumančić et H. Blockeel. 2018a, «Cobras ts : A new approach to semi-supervised clustering of time series», dans *International Conference on Discovery Science*, Springer, p. 179–193. 32, 44

Van Craenendonck, T., W. Meert, S. Dumančić et H. Blockeel. 2018b, «Cobras ts : A new approach to semi-supervised clustering of time series», dans *International Conference on Discovery Science*, Springer, p. 179–193. 45

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser et I. Polosukhin. 2017, «Attention is all you need», *arXiv preprint arXiv :1706.03762*. 82

Vincent, P., H. Larochelle, Y. Bengio et P.-A. Manzagol. 2008, «Extracting and composing robust features with denoising autoencoders», dans *Proceedings of the 25th international conference on Machine learning*, p. 1096–1103. 84

Vlachos, M., G. Kollios et D. Gunopulos. 2002, «Discovering similar multidimensional trajectories», dans *Proceedings 18th international conference on data engineering*, IEEE, p. 673–684. 7

Von Luxburg, U. 2007, «A tutorial on spectral clustering», *Statistics and computing*, vol. 17, n° 4, p. 395–416. 14, 44

Wagstaff, K. et C. Cardie. 2000, «Clustering with instance-level constraints», *AAAI/IAAI*, vol. 1097, p. 577–584. 12

Wagstaff, K., C. Cardie, S. Rogers, S. Schrödl et collab.. 2001, «Constrained k-means clustering with background knowledge», dans *Icml*, vol. 1, p. 577–584. 13, 20

Wagstaff, K. L., S. Basu et I. Davidson. 2006, «When is constrained clustering beneficial, and why ?», *Ionosphere*, vol. 58, n° 60.1, p. 62–63. 23, 28

Wang, C., S. Pan, R. Hu, G. Long, J. Jiang et C. Zhang. 2019a, «Attributed graph clustering : A deep attentional embedding approach», *arXiv preprint arXiv :1906.06532*. 82

Wang, H., M. Du, F. Yang et Z. Zhang. 2019b, «Score-cam : Improved visual explanations via score-weighted class activation mapping», *arXiv preprint arXiv :1910.01279*. 113

Wang, X., A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann et E. Keogh. 2013, «Experimental comparison of representation methods and distance measures for time series data», *Data Mining and Knowledge Discovery*, vol. 26, n° 2, p. 275–309. 20

Wang, X., B. Qian et I. Davidson. 2014, «On constrained spectral clustering and its applications», *Data Mining and Knowledge Discovery*, vol. 28, n° 1, p. 1–30. 14

Wang, Z., W. Yan et T. Oates. 2017, «Time series classification from scratch with deep neural networks : A strong baseline», dans *2017 International joint conference on neural networks (IJCNN)*, IEEE, p. 1578–1585. 70, 71, 79, 95

Weiss, G., Y. Goldberg et E. Yahav. 2018, «On the practical computational power of finite precision rnns for language recognition», *arXiv preprint arXiv :1805.04908*. 81

Wemmert, C., P. Gançarski et J. J. Korczak. 2000, «A collaborative approach to combine multiple learning methods», *International Journal on Artificial Intelligence Tools*, vol. 9, n° 01, p. 59–78. 15

Wilcoxon, F. 1992, «Individual comparisons by ranking methods», dans *Breakthroughs in statistics*, Springer, p. 196–202. 100

Woo, S., J. Park, J.-Y. Lee et I. S. Kweon. 2018, «Cbam : Convolutional block attention module», dans *Proceedings of the European conference on computer vision (ECCV)*, p. 3–19. 82

Xiao, W., Y. Yang, H. Wang, T. Li et H. Xing. 2016, «Semi-supervised hierarchical clustering ensemble and its application», *Neurocomputing*, vol. 173, p. 1362–1376. 15

Xiao, Y. et K. Cho. 2016, «Efficient character-level document classification by combining convolution and recurrent layers», *arXiv preprint arXiv :1602.00367*. 79

Xiao, Z., X. Xu, H. Xing et J. Chen. 2020, «Rtfn : Robust temporal feature network», *arXiv preprint arXiv :2008.07707*. 99

Xie, J., R. Girshick et A. Farhadi. 2016, «Unsupervised deep embedding for clustering analysis», dans *International conference on machine learning*, p. 478–487. 66, 67, 68, 79, 83, 84, 88, 94, 97, 99, 102, 113

Xiong, S., J. Azimi et X. Z. Fern. 2013, «Active learning of constraints for semi-supervised clustering», *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, n° 1, p. 43–54. 46

Xu, J., L. Xiao et A. M. López. 2019, «Self-supervised domain adaptation for computer vision tasks», *IEEE Access*, vol. 7, p. 156 694–156 706. 83

Yang, K. et C. Shahabi. 2004, «A pca-based similarity measure for multivariate time series», dans *Proceedings of the 2nd ACM international workshop on Multimedia databases*, p. 65–74. 7

Yang, X., C. Deng, F. Zheng, J. Yan et W. Liu. 2019, «Deep spectral clustering using dual autoencoder network», dans *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, p. 4066–4075. 66, 67, 88, 91, 97, 99

Yang, Y., W. Tan, T. Li et D. Ruan. 2012, «Consensus clustering based on constrained self-organizing map and improved cop-kmeans ensemble in intelligent decision support systems», *Knowl-Based Syst*, vol. 32, p. 101—115. 15

Yu, F. et V. Koltun. 2015, «Multi-scale context aggregation by dilated convolutions», *arXiv preprint arXiv :1511.07122*. 80

Yu, H. H., C. H. Chen et S. Tseng. 2011a, «Mining emerging patterns from time series data with time gap constraint», *International Journal of Innovative Computing, Information and Control*, vol. 7, n° 9, p. 5515–5528. 66

Yu, Z., H. Wongb, J. You, Q. Yang et H. Liao. 2011b, «Knowledge based cluster ensemble for cancer discovery from biomolecular data», *IEEE Trans Nanobioscience*, vol. 10, n° 2, p. 76—85. 15

Zeng, N., H. Zhang, B. Song, W. Liu, Y. Li et A. M. Dobaie. 2018, «Facial expression recognition via learning deep sparse autoencoders», *Neurocomputing*, vol. 273, p. 643–649. 84

Zha, H., X. He, C. Ding, M. Gu et H. D. Simon. 2002, «Spectral relaxation for k-means clustering», dans *Advances in neural information processing systems*, p. 1057–1064. 93

Zhang, H., S. Basu et I. Davidson. 2019, «A framework for deep constrained clustering-algorithms and advances», dans *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, p. 57–72. 68, 72

Zhang, Q., J. Wu, P. Zhang, G. Long et C. Zhang. 2018, «Salient subsequence learning for time series clustering», *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, n° 9, p. 2193–2207. 7, 99, 106

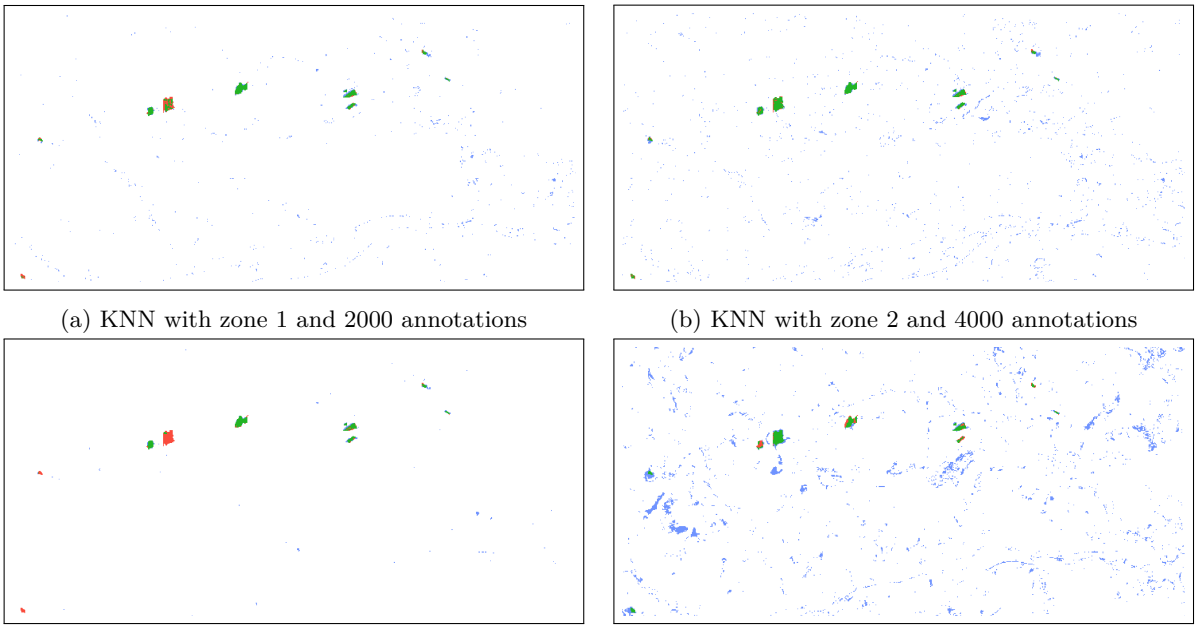
Zhou, B., A. Khosla, A. Lapedriza, A. Oliva et A. Torralba. 2016, «Learning deep features for discriminative localization», dans *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 2921–2929. 111

Zhu, Y., N. Zabaras, P.-S. Koutsourelakis et P. Perdikaris. 2019, «Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data», *Journal of Computational Physics*, vol. 394, p. 56–81. 68

# Annexe A

# Annexes

## A.1 Annexe des figures



(a) KNN with zone 1 and 2000 annotations

(b) KNN with zone 2 and 4000 annotations

(c) RF with zone 1 and 1000 annotations

(d) RF with zone 2 and 250 annotations

FIGURE A.1 – Résultats des classifieurs KNN et Random Forest qui obtiennet des résulats équivalents à ceux d'I-SAMARAH, en vert les Vrais Positifs, en rouge les Faux Négatifs et en bleu les Faux Positifs.

## A.2 Annexe des tableaux

| Méthode | 5% | | 10% | | 15% | | 50% | |
|---|---|---|---|---|---|---|---|---|
| | ARI | Sat. | ARI | Sat. | ARI | Sat. | ARI | Sat. |
| COP-KMeans | 0.410 | **1.000** | 0.412 | **1.000** | 0.411 | **1.000** | 0.377$^{\dagger}$ | **1.000** |
| Spec | **0.818** | 0.926 | **0.819** | 0.926 | **0.804** | 0.910 | **0.826** | 0.926 |
| CCSR | 0.497 | 0.752 | 0.491 | 0.754 | 0.487 | 0.754 | 0.483 | 0.750 |
| CPClustering | 0.245 | **1.000** | 0.345 | **1.000** | 0.297 | **1.000** | 0.247 | **1.000** |
| SAMARAH | 0.760 | 0.890 | 0.754 | 0.884 | 0.748 | 0.872 | 0.759 | 0.878 |

TABLEAU A.1 – Performances sur ECG5000. Le meilleur score pour chaque fraction de contraintes est indiqué en gras. $^{\dagger}$9 contraintes.

| Méthode | 5% | | 10% | | 15% | | 50% | |
|---|---|---|---|---|---|---|---|---|
| | ARI | Sat. | ARI | Sat. | ARI | Sat. | ARI | Sat. |
| COP-KMeans | 0.327 | **1.000** | **0.347** | **1.000** | **0.343** | **1.000** | **0.351** | **1.000** |
| Spec | 0.231 | 0.769 | 0.231 | 0.754 | 0.231 | 0.764 | 0.231 | 0.764 |
| CCSR | 0.328 | 0.779 | 0.328 | 0.794 | 0.328 | 0.792 | 0.328 | 0.796 |
| CPClustering | 0.199 | **1.000** | 0.194 | **1.000** | 0.185 | **1.000** | 0.099 | **1.000** |
| SAMARAH | **0.329** | 0.816 | 0.335 | 0.813 | 0.312 | 0.799 | 0.329 | 0.805 |

TABLEAU A.2 – Performances sur ElectricDevices. Le meilleur score pour chaque fraction de contraintes est indiqué en gras.

| Méthode | 5% | | 10% | | 15% | | 50% | |
|---|---|---|---|---|---|---|---|---|
| | ARI | Sat. | ARI | Sat. | ARI | Sat. | ARI | Sat. |
| COP-KMeans | 0.494 | **1.000** | 0.478 | **1.000** | 0.466 | **1.000** | 0.495 | **1.000** |
| Spec | 0.463 | 0.948 | 0.427 | 0.935 | 0.405 | 0.914 | 0.299 | 0.874 |
| CCSR | 0.328 | 0.943 | **0.619** | 0.945 | **0.614** | 0.942 | **0.610** | 0.940 |
| CPClustering | 0.146 | **1.000** | 0.158 | **1.000** | 0.156 | **1.000** | 0.168 | **1.000** |
| SAMARAH | **0.541** | 0.952 | 0.540 | 0.944 | 0.529 | 0.937 | 0.584 | 0.943 |

TABLEAU A.3 – Performances sur FacesUCR. Le meilleur score pour chaque fraction de contraintes est indiqué en gras.

| Méthode | 5% | | 10% | | 15% | | 50% | |
|---|---|---|---|---|---|---|---|---|
| | ARI | Sat. | ARI | Sat. | ARI | Sat. | ARI | Sat. |
| COP-KMeans | 0.060 | **1.000** | 0.059 | **1.000** | 0.060 | **1.000** | 0.058 | **1.000** |
| Spec | **0.148** | 0.863 | **0.147** | 0.862 | **0.150** | 0.873 | **0.153** | 0.862 |
| CCSR | 0.135 | 0.845 | 0.134 | 0.849 | 0.133 | 0.864 | 0.135 | 0.853 |
| CPClustering | 0.073 | **1.000** | 0.080 | **1.000** | 0.077 | **1.000** | 0.075 | **1.000** |
| SAMARAH | 0.048 | 0.875 | 0.056 | 0.872 | 0.052 | 0.876 | 0.146 | 0.845 |

TABLEAU A.4 – Performances sur InsectWingbeatSound. Le meilleur score pour chaque fraction de contraintes est indiqué en gras.

| Méthode | 5% | | 10% | | 15% | | 50% | |
|---|---|---|---|---|---|---|---|---|
| | ARI | Sat. | ARI | Sat. | ARI | Sat. | ARI | Sat. |
| COP-KMeans | 0.823 | **1.000** | 0.831 | **1.000** | 0.791 | **1.000** | 0.858 | **1.000** |
| Spec | 0.931 | 0.982 | 0.931 | 0.987 | 0.931 | 0.986 | **0.931** | 0.984 |
| CCSR | **0.934** | 0.980 | **0.934** | 0.987 | **0.933** | 0.985 | 0.931 | 0.983 |
| CPClustering | 0.624 | **1.000** | 0.627 | **1.000** | 0.638 | **1.000** | 0.614 | **1.000** |
| SAMARAH | 0.864 | 0.971 | 0.898 | 0.979 | 0.848 | 0.966 | 0.887 | 0.975 |

TABLEAU A.5 – Performances sur MALLAT. Le meilleur score pour chaque fraction de contraintes est indiqué en gras.

| Méthode | 5% | | 10% | | 15% | | 50% | |
|---|---|---|---|---|---|---|---|---|
| | ARI | Sat. | ARI | Sat. | ARI | Sat. | ARI | Sat. |
| COP-KMeans | 0.534 | **1.000** | 0.536 | **1.000** | 0.526$^{\dagger}$ | **1.000**$^{\dagger}$ | - | - |
| Spec | **0.678** | 0.828 | **0.678** | 0.834 | **0.678** | 0.834 | 0.678 | 0.834 |
| CCSR | 0.537 | 0.780 | 0.537 | 0.765 | 0.538 | 0.775 | 0.538 | 0.772 |
| CPClustering | 0.619 | **1.000** | 0.581 | **1.000** | 0.635 | **1.000** | **0.685** | **1.000** |
| SAMARAH | 0.566 | 0.795 | 0.608 | 0.807 | 0.577 | 0.792 | 0.586 | 0.789 |

TABLEAU A.6 – Performances sur StarLightCurves. Le meilleur score pour chaque fraction de contraintes est indiqué en gras. $^{\dagger}$7 contraintes.

| Méthode | 5% | | 10% | | 15% | | 50% | |
|---|---|---|---|---|---|---|---|---|
| | ARI | Sat. | ARI | Sat. | ARI | Sat. | ARI | Sat. |
| COP-KMeans | 0.908 | **1.000** | 0.934 | **1.000** | 0.914$^{\dagger}$ | **1.000**$^{\dagger}$ | 0.913$^{\ddagger}$ | **1.000**$^{\ddagger}$ |
| Spec | **1.000** | **1.000** | **1.000** | **1.000** | **1.000** | **1.000** | **1.000** | **1.000** |
| CCSR | **1.000** | **1.000** | **1.000** | **1.000** | **1.000** | **1.000** | **1.000** | **1.000** |
| CPClustering | 0.269 | **1.000** | 0.233 | **1.000** | 0.228 | **1.000** | 0.235 | **1.000** |
| SAMARAH | 0.867 | 0.948 | 0.881 | 0.958 | 0.881 | 0.958 | 0.870 | 0.951 |

TABLEAU A.7 – Performances sur TwoPatterns. Le meilleur score pour chaque fraction de contraintes est indiqué en gras. $^{\dagger}$9 contraintes ; $^{\ddagger}$7 contraintes.

| Méthode | 5% | | 10% | | 15% | | 50% | |
|---|---|---|---|---|---|---|---|---|
| | ARI | Sat. | ARI | Sat. | ARI | Sat. | ARI | Sat. |
| COP-KMeans | **0.439** | **1.000** | **0.450** | **1.000** | **0.438** | **1.000** | **0.441** | **1.000** |
| Spec | 0.103 | 0.659 | 0.142 | 0.738 | 0.078 | 0.663 | 0.066 | 0.677 |
| CCSR | 0.405 | 0.870 | 0.407 | 0.862 | 0.407 | 0.875 | 0.406 | 0.862 |
| CPClustering | 0.220 | **1.000** | 0.250 | **1.000** | 0.250 | **1.000** | 0.212 | **1.000** |
| Samarah | 0.409 | 0.898 | 0.416 | 0.882 | 0.420 | 0.876 | 0.420 | 0.870 |

Tableau A.8 – Performances sur uWaveGestureLibraryX. Le meilleur score pour chaque fraction de contraintes est indiqué en gras.

| Méthode | 5% | | 10% | | 15% | | 50% | |
|---|---|---|---|---|---|---|---|---|
| | ARI | Sat. | ARI | Sat. | ARI | Sat. | ARI | Sat. |
| COP-KMeans | 0.431 | **1.000** | 0.440 | **1.000** | 0.435 | **1.000** | 0.422 | **1.000** |
| Spec | **0.463** | 0.892 | **0.482** | 0.894 | **0.466** | 0.878 | **0.502** | 0.892 |
| CCSR | 0.439 | 0.878 | 0.450 | 0.886 | 0.458 | 0.886 | 0.457 | 0.881 |
| CPClustering | 0.187 | **1.000** | 0.184 | **1.000** | 0.182 | **1.000** | 0.180 | **1.000** |
| Samarah | 0.386 | 0.885 | 0.370 | 0.873 | 0.376 | 0.867 | 0.384 | 0.855 |

Tableau A.9 – Performances sur UWaveGestureLibraryAll. Le meilleur score pour chaque fraction de contraintes est indiqué en gras.

| Méthode | 5% | | 10% | | 15% | | 50% | |
|---|---|---|---|---|---|---|---|---|
| | ARI | Sat. | ARI | Sat. | ARI | Sat. | ARI | Sat. |
| ECG5000 | 0.587 | 0.69 | 0.625 | 0.802 | 0.612 | 0.815 | 0.787 | 0.910 |
| ElectricDevices | 0.121 | 0.702 | 0.089 | 0.691 | 0.149 | 0.712 | 0.231 | 0.864 |
| FacesUCR | 0.157 | 0.810 | 0.112 | 0.792 | 0.21 | 0.821 | 0.187 | 0.832 |
| InsectWingbeatSound | 0.112 | 0.844 | 0.129 | 0.841 | 0.145 | 0.841 | 0.159 | 0.902 |
| MALLAT | 0.421 | 0.902 | 0.458 | 0.924 | 0.491 | 0.922 | 0.524 | 0.931 |
| StarLightCurves | 0.281 | 0.624 | 0.185 | 0.601 | 0.281 | 0.624 | 0.221 | 0.729 |
| TwoPatterns | 0.058 | 0.818 | 0.268 | 0.892 | 0.312 | 0.889 | 0.719 | 0.913 |
| uWaveGestureLibraryX | 0.015 | 0.659 | 0.035 | 0.521 | 0.048 | 0.489 | 0.015 | 0.789 |
| uWaveGestureLibraryAll | 0.196 | 0.789 | 0.251 | 0.812 | 0.146 | 0.805 | 0.206 | 0.928 |

Tableau A.10 – Performances par jeux de données de la méthode DCC.

| Méthode | 5% | | 10% | | 15% | | 50% | |
|---|---|---|---|---|---|---|---|---|
| | ARI | Sat. | ARI | Sat. | ARI | Sat. | ARI | Sat. |
| ECG5000 | 0.489 | 0.741 | 0.498 | 0.757 | 0.481 | 0.751 | 0.52 | 0.778 |
| ElectricDevices | 0.045 | 0.564 | 0.051 | 0.594 | 0.049 | 0.575 | 0.035 | 0.547 |
| FacesUCR | 0.129 | 0.784 | 0.118 | 0.794 | 0.115 | 0.769 | 0.121 | 0.781 |
| InsectWingbeatSound | 0.058 | 0.812 | 0.048 | 0.821 | 0.101 | 0.835 | 0.035 | 0.795 |
| MALLAT | 0.288 | 0.847 | 0.385 | 0.912 | 0.360 | 0.896 | 0.404 | 0.921 |
| StarLightCurves | 0.315 | 0.725 | 0.384 | 0.787 | 0.356 | 0.765 | 0.401 | 0.801 |
| TwoPatterns | 0.094 | 0.821 | 0.015 | 0.812 | 0.065 | 0.823 | 0.052 | 0.789 |
| uWaveGestureLibraryX | 0.135 | 0.702 | 0.128 | 0.687 | 0.145 | 0.708 | 0.140 | 0.700 |
| uWaveGestureLibraryAll | 0.426 | 0.884 | 0.284 | 0.826 | 0.189 | 0.789 | 0.286 | 0.842 |

Tableau A.11 – Performances par jeux de données de la méthode DCC.