THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *Date de défense (17/04/2018)* **par :**

VINCENT COURJAULT-RADE

Ballstering: un algorithme de clustering dédié à de grands échantillons

---

JURY

SAME ALLOU	Maître de conférences	Rapporteur
CARRIZOSA EMILIO	Professeur d'Université	Rapporteur
MOTHE JOSIANE	Professeur d'Université	Examinatrice
MAHEY PHILIPPE	Professeur de classe exceptionnelle	Examinateur
D'ESTAMPES LUDOVIC	Maître de conférences	Directeur de Thèse

École doctorale et spécialité :
MITT : Domaine Mathématiques : Mathématiques appliquées
Unité de Recherche :
ENAC TOULOUSE
Directeur(s) de Thèse :
PUECHMOREL Stéphane et *d'ESTAMPES Ludovic*
Rapporteurs :
Same Allou et *Carrizosa Emilio*

Résumé

Ballstering appartient à la famille des méthodes de machine learning qui ont pour but de regrouper en classes les éléments formant la base de données étudiée et ce sans connaissance au préalable des classes qu'elle contient. Ce type de méthodes, dont le représentant le plus connu est k-means, se rassemblent sous le terme de "partitionnement de données" ou "clustering".

Récemment un algorithme de partitionnement "Fast Density Peak Clustering" (FDPC) paru dans le journal Science a suscité un intérêt certain au sein de la communauté scientifique pour son aspect innovant et son efficacité sur des données distribuées en groupes non-concentriques. Seulement cet algorithme présente une complexité telle qu'il ne peut être aisément appliqué à des données volumineuses. De plus nous avons pu identifier plusieurs faiblesses pouvant nuire très fortement à la qualité de ses résultats, dont en particulier la présence d'un paramètre général d_c difficile à choisir et ayant malheureusement un impact non-négligeable. Compte tenu de ces limites, nous avons repris l'idée principale de FDPC sous un nouvel angle puis apporté successivement des modifications en vue d'améliorer ses points faibles. Modifications sur modifications ont finalement donné naissance à un algorithme bien distinct que nous avons nommé Ballstering.

Le fruit de ces 3 années de thèse se résume principalement en la conception de ce dernier, un algorithme de partitionnement dérivé de FDPC spécialement conçu pour être efficient sur de grands volumes de données. Tout comme son précurseur, Ballstering fonctionne en deux phases : une phase d'estimation de densité suivie d'une phase de partitionnement. Son élaboration est principalement fondée sur la construction d'une sous-procédure permettant d'effectuer la première phase de FDPC avec une complexité nettement amoindrie tout évitant le choix de d_c qui devient dynamique, déterminé suivant la densité locale. Nous appelons ICMDW cette sous-procédure qui représente une partie conséquente de nos contributions.

Nous avons également remanié certaines des définitions au cœur de FDPC et revu entièrement la phase 2 en s'appuyant sur la structure arborescente des résultats fournis par ICDMW pour finalement produire un algorithme outrepassant toutes les limitations que nous avons identifié chez FDPC.

Abstract

Ballstering belongs to the machine learning methods that aim to group in classes a set of objects that form the studied dataset, without any knowledge of true classes within it. This type of methods, of which k-means is one of the most famous representative, are named clustering methods. Recently, a new clustering algorithm "Fast Density Peak Clustering" (FDPC) has aroused great interest from the scientific community for its innovating aspect and its efficiency on non-concentric distributions.

However this algorithm showed a such complexity that it can't be applied with ease on large datasets. Moreover, we have identified several weaknesses that impact the quality results and the presence of a general parameter d_c, difficult to choose while having a significant impact on the results.

In view of those limitations, we reworked the principal idea of FDPC in a new light and modified it successively to finally create a distinct algorithm that we called Ballstering.

The work carried out during those three years can be summarised by the conception of this clustering algorithm especially designed to be effective on large datasets. As its Precursor, Ballstering works in two phases : An estimation density phase followed by a clustering step. Its conception is mainly based on a procedure that handle the first step with a lower complexity while avoiding at the same time the difficult choice of d_c, which becomes automatically defined according to local density. We name ICMDW this procedure which represent a consistent part of our contributions.

We also overhauled cores definitions of FDPC and entirely reworked the second phase (relying on the graph structure of ICMDW's intermediate results), to finally produce an algorithm that overcome all the limitations that we have identified.

Remerciements

Je tiens tout d'abord à remercier mes encadrants Stéphane Puechmorel et Ludovic d'Estampes qui ont sû m'orienter tout le long de mon doctorat. Je remercie Egalement les rapporteurs Same Allou et Carrizosa Emilio ainsi que les examinateurs Mothe Josiane et Mahey Philippe qui m'ont examiné avec interêt et en faisant preuve d'une grande bienveillance.

Merci particulièrement à notre petit groupe de doctorants et les agrégés avec qui j'ai passé de très bons moments tous les jours au laboratoire.

Je remercie aussi tous les membres de l'ENAC et en particulier ceux qui ont généreusement donné de leur temps pour m'aider.

Merci à mes amis et camarades qui m'ont soutenu durant nos études.

Un très grand merci à Gilles qui a passé beaucoup trop de temps à m'aider et à créer la magnifique vidéo et images qui illustrent mon algorithme.

Un merci bonus à ceux qui m'ont fait l'honneur de leur présence à la soutenance.

Et enfin, un énorme merci à toutes les personnes bienveillantes qui ont fait mon quotidien durant ces trois années.

Table des matières

Table des figures

Liste des tableaux

Introduction

La science des données est une discipline récente qui a émergé simultanément à l'accumulation d'informations enregistrées dans des bases de données. Cette discipline tâche de recouvrir toutes les problématiques qui y sont liées dont le stockage, la protection et la visualisation des données ainsi que l'extraction de connaissances (prédictions, corrélations). Cette science, qui s'appuie donc fortement sur la statistique et l'informatique, est en plein essor au vu des quantités toujours plus importantes des données récoltées.

La branche qui nous intéresse particulièrement est l'exploration de données, plus largement connue sous le nom de « data mining », qui a pour but d'extraire de l'information et connaissances d'une certaine base de données, souvent volumineuse, afin d'aider à la prise de décisions stratégiques dans des secteurs variés comme la gestion de relation client, la détection de fraude, la prévention d'incidents ou encore en Bio-Informatique.

Un des outils les plus adaptés à ce problème est l'apprentissage automatique ou « machine learning »en anglais. Les méthodes d'apprentissage automatique ont la particularité de pouvoir apprendre, d'ajuster leur modèle et leur comportement en fonction des observations et ainsi évoluer d'elles-mêmes au cours du traitement des données, permettant ainsi de faire face à la complexité du problème posé. Un de ses principaux objectifs est de construire des méthodes simples d'utilisation, idéalement automatisées, capables d'analyser des données volumineuses et potentiellement hétérogènes afin d'en extraire de l'information.

Parmi les méthodes de machine learning, on peut dénoter une catégorie d'algorithmes ayant pour but de regrouper en classes les éléments formant la base de données étudiée et ce sans connaissance au préalable des classes qu'elle contient. Ce type de méthodes, dont le représentant le plus connu est k-means, se regroupent sous le terme de « partitionnement de données » ou « clustering ».

Récemment un algorithme de partitionnement « Fast Density Peak Clustering » (FDPC) paru dans le journal Science en 2014 a suscité un intérêt certain au sein de la communauté scientifique pour son aspect innovant et son efficacité sur des données distribuées en groupes non-concentriques. Seulement cet algorithme présente une telle complexité qu'il ne peut être aisément appliqué à une quantité conséquente de données. De plus, nous avons pu identifier plusieurs faiblesses pouvant nuire très fortement à la qualité de ses résultats dont en particulier la présence d'un paramètre général d_c difficile à choisir et ayant malheureusement un impact non négligeable sur les résultats.

Compte tenu de ces limites, nous avons repris l'idée principale de FDPC sous un angle nouveau puis avons apporté successivement des modifications en vue d'améliorer ses points faibles. Modification après modification, le nouvel algorithme, nommé Ballstering, n'est plus réellement comparable à FDPC.

Le fruit de ces 3 années de thèse se résume principalement en la conception de cet algorithme, un algorithme de partitionnement dérivé de FDPC spécialement conçu pour être efficient sur de grands volumes de données. Tout comme son précurseur, il fonctionne en deux phases : une phase d'estimation de densité suivie d'une phase de partitionnement. Son élaboration est principalement fondée sur une sous-procédure permettant d'effectuer la première phase de FDPC avec une complexité nettement amoindrie tout en évitant le choix de d_c qui devient dynamique, déterminé suivant la densité locale. Nous appelons ICMDW cette sous-procédure qui représente une partie conséquente de ma contribution.

De la construction de ICMDW, nous avons ainsi pu établir trois travaux supplémentaires.

1. Nous avons construit un dérivé de la phase 2 de FDPC en conservant son principe de fonctionnement tout en utilisant les particularités des résultats intermédiaires fournis par ICMDW pour accélérer cette deuxième phase. En résulte un algorithme aux comportements semblables à FDPC mais plus rapide et robuste vis-à-vis du paramètre d'entrée $N_{\min}$.
2. Nous avons remanié certaines des définitions au cœur de FDPC et revu entièrement la phase 2 en nous appuyant sur la structure arborescente des résultats

fournis par ICDMW pour finalement produire un algorithme *outrepassant toutes les limitations* que nous avons identifiées chez FDPC. C'est cet algorithme que nous nommons Ballstering.

3. Enfin, ICMDW permet l'obtention d'une méthode non-paramétrique d'estimation de densité, faite de manière adaptative et quasiment automatique dans le sens où aucun paramètre n'est à choisir dans la plupart des cas.

À tout ce qui précède s'agrège un travail de réflexion théorique et d'expérimentation empiriques sur la complexité, la qualité des partitions produites ainsi que sur les limitations potentielles de ce que nous avons établi.

Dans le chapitre 1, nous situons la place de Ballstering dans la littérature et décrivons FDPC, la méthode qui a fait germer l'idée de Ballstering et dont ce dernier est dérivé, ainsi que les tentatives d'amélioration existantes et les méthodes auxquelles Ballstering sera comparé dans la dernière partie.

Dans le chapitre 2, nous présentons divers outils et développements mathématiques auxiliaires qui sont manipulés au cours de ce mémoire.

Le chapitre 3 consiste en la description de la procédure ICMDW et le chapitre 4 porte sur l'exploitation des outils fournis par cette dernière, à savoir l'estimation de densité non-paramétrique et les deux manières d'effectuer la phase 2 d'extraction des partitions.

Dans le chapitre 5, nous débattons de la sélection des paramètres, de la complexité et des avantages et inconvénients de Ballstering.

Nous commentons dans le chapitre 6 les résultats empiriques obtenus en comparant Ballstering à d'autres méthodes, attestant de sa qualité, rapidité et robustesse.

Enfin dans le dernier chapitre, après avoir rappelé les différents travaux faits pendant la thèse, nous présentons les différentes perspectives : parallélisation, problème de surestimation, changement d'espace euclidien.

Chapitre 1

Clustering : FDPC, k-means et HDBSCAN

Notre algorithme Ballstering a pour objectif principal de séparer des données d'un échantillon, les agréger de manière à constituer un ensemble de groupes homogènes, c'est à dire contenant chacun des éléments partageant certaines caractéristiques communes. Cet objectif est plus largement celui du clustering, ou partitionnement de données en français, qui est une problématique phare du machine learning.

En premier lieu dans ce chapitre nous éclaircirons ce que désignent les termes clustering et machine learning et ce qu'il se fait dans ce domaine. Dans un second temps nous parlerons en détail de l'algorithme FDPC, dont l'idée générale a été reprise pour constituer Ballstering, ainsi que de k-means et HDBSCAN deux méthodes connues et efficaces auxquelles notre algorithme sera expérimentalement comparé.

1.1 Machine learning

Compte tenu du nombre d'observations et de paramètres les dépendances et tendances émergeant des jeux de données sont souvent trop complexes pour être décrits et mis en évidence aisément, d'où la nécessité d'utiliser des algorithmes pour automatiser le traitement.

Le machine learning est issu de la branche de l'intelligence artificielle et est donc lié par essence à l'informatique. Il a pour but de résoudre ces problèmes difficiles dans l'optique de prendre des décisions stratégiques à partir d'informations stockées sous la forme de base de données. Ces algorithmes également rassemblés sous le terme

« d'apprentissage automatique » ont la particularité de pouvoir apprendre, d'ajuster leur modèle et leur comportements en fonction des observations et ainsi d'évoluer d'eux-même pendant le traitement des données. Cette souplesse permet d'obtenir des algorithmes opérationnels malgré la difficulté du problème, de relever le défi généré par des bases de données toujours plus volumineuses et complexes auxquelles on peut désormais se confronter à notre époque.

Derrière le terme d'apprentissage automatique se cache de nombreuses branches qui se différencient selon le type de données à traiter, le choix de modélisation, l'objectif de l'étude et les moyens techniques à disposition.

Les données les plus classiques sont sous forme matricielle où chaque ligne représente une observation et chaque colonne représente une variable. Lorsque les variables sont de nature discrète (catégorielles) on parle de variables qualitatives. On utilise le terme « quantitatives » quand elles sont continues et « mixte » lorsque ces deux dernières sont présentes.

Mais on peut part exemple être amené à étudier des données de type enregistrement en temps réel comme celles obtenues par des capteurs sonores et visuels ou encore issues de l'enregistrement de l'activité d'individus sur internet. On peut vouloir également étudier des données fonctionnelles, c'est à dire décrivant une courbe d'évolution temporelle d'une mesure comme des enregistrements de trajectoires d'avions ou de séries temporelles issues de la finance par exemple. Enfin on peut dénoter les données textuelles où chaque observation est un fichier texte qui doit être analysé au préalable ou encore les données sous forme de graphe.

Dans chacun de ces cas, les méthodes et modélisations employées pourront énormément varier. Parmi l'ensemble des familles d'algorithmes traitant de ce domaine, les plus classiques sont :

— les réseaux de neurones (Deep learning si à couches multiples),
— les machines à vecteur de support (SVM),
— les algorithmes et programmation génétique (itérations et mutations des solutions),
— les méthodes statistiques (Algorithmes de type EM par exemple),
— les arbres de décision,

— k plus proches voisins,

— Boosting (fusion de classifieurs binaires).

Les applications réelles sont variées. On retrouve notamment le machine learning en première place dans les problématiques de reconnaissance visuelle (visages, caractères...) ou vocale, dans la recherche de correspondance entre individus/produits (Matching) comme proposer un individu pertinent sur les sites de rencontres ou cibler un consommateur pour un produit particulier, dans l'aide au diagnostic médical, la détection de comportements suspects pour la sécurité informatique, l'analyse des tendances boursières et l'Intelligence Artificielle dans les jeux vidéos, des drones et robots en général.

De nos jours certains algorithmes, issus du deep learning (réseaux de neurones profonds) sont capables d'apprendre et de réagir en temps réel, comme c'est le cas du projet de la voiture autonome qui, à partir de données issues de multiples capteurs et un apprentissage en simulation, arrive à produire une stratégie de conduite concurrençant l'humain. D'autres prouesses ont pu être réalisées comme récemment la résolution du jeu de go qui est pourtant un jeu d'une complexité effrayante si l'on s'essaie à considérer l'ensemble des coups possibles.

1.1.1 Classification automatique

La classification automatique est une des sous-branches du machine learning dont la vocation est d'extraire/retrouver automatiquement les différents groupes et tendances communes au sein d'un échantillon de données.

Afin de pouvoir comprendre et se faire une représentation du réel, les limites de la cognition humaine nous obligent à penser en terme de classes, à catégoriser les choses qui nous entourent afin de pouvoir naviguer dans cette complexité. Cette catégorisation notre cerveau l'exécute de manière remarquable dans toutes les situations de la vie quotidienne. Cependant nous sommes bien incapables d'effectuer la même tâche lorsqu'on considère des gigas d'informations stockées dans des ordinateurs, sous forme matricielle par exemple. Notre subjectivité n'étant vue que comme un obstacle à la découverte d'informations fondues au sein d'un échantillon, le but de la classification automatique est d'attribuer une classe à chaque individu à partir de

procédés ne faisant intervenir que les données et très peu l'utilisateur. La subjectivité de l'expérimentateur ne doit être présente que dans le choix de la modélisation et des représentations que l'algorithme utilise.

On peut différentier deux grandes familles d'algorithmes de classification automatique par rapport à l'expertise, à la connaissance préalable nécessaire à la modélisation et résolution : les algorithmes supervisés et les algorithmes non-supervisés.

Dans le premier cas, supervisé, les classes et leurs caractéristiques sont connues au préalable. À partir d'une base de données d'apprentissage qui contient des observations dont certaines possèdent leur classe de renseignée, on cherche à obtenir la meilleure règle de classification possible, pour enfin appliquer cette règle et prédire la classe des observations dont la classe est manquante. On parle alors de classification.

À l'inverse les algorithmes non-supervisés utilisent des jeux de données dans lesquels on ne connaît initialement rien sur les classes en son sein. Il faut donc que ces algorithmes soient capable de déterminer leur nombre (bien qu'il puisse être laissé à l'utilisateur dans certains algorithmes...), leur forme et quelles observations leur appartiennent. On parle dans ce cas de partitionnement, d'agrégation de données ou encore clustering. C'est dans cette dernière catégorie que notre recherche s'inscrit.

1.1.2 Partitionnement de données (Clustering)

Apparu au milieu du $20^{\text{ème}}$ siècle, le partitionnement de données appartient au domaine du machine learning et plus précisément à la classification automatique non supervisée. On se place donc dans le cadre où le nombre de groupes et les caractéristiques communes de ces groupes au sein de l'échantillon ne sont pas connus par l'expérimentateur.

Le pari est fait qu'au sein du jeu de données d'intérêt, des concentrations de données similaires vont se démarquer des autres amas de données par leur dissimilarité. Ainsi on va chercher à catégoriser une collection d'objets de telle manière que des observations appartenant à un même groupe soient plus similaires entre elles qu'avec tout autre élément appartenant à d'autres groupes, et dissimilaires quand ils appartiennent à des groupes différents. En d'autres termes le clustering vise à diviser les observations en différents groupes, homogènes au sein du groupe et hétérogènes

entre groupes. On peut souvent lire que pour obtenir un bon partitionnement, il est nécéssaire d'à la fois minimiser l'inertie intra-classe (variance au sein d'un groupe) pour obtenir des clusters les plus homogènes possibles mais aussi maximiser l'inertie inter-classe (variance entre les groupes) afin d'obtenir des groupes distincts. Un partitionnement idéal est constitué de groupes bien séparés et compacts.

Cette homogénéité est très souvent évaluée selon une distance ou plus largement une mesure de dissimilarité dont le choix revient à l'expert et peut plus ou moins fortement influencer sur la qualité des partitions, en induisant un biais potentiel. Cette mesure de similarité peut se traduire tout simplement par une distance euclidienne si les données s'y prêtent (repère orthonormé) ou se baser sur une mesure propre au problème et définie au préalable par l'utilisateur.

La communauté travaillant dans ce domaine éprouve des difficultés à unifier ce relativement informel concept qu'est le clustering. Un grand nombre d'approches et de concepts différents sont à la racine des algorithmes de regroupement. Les possibilités sont larges, les compromis inévitables compte tenu de la difficulté du problème et les objectifs ainsi que la nature des données sont tellement variés que les algorithmes existants ne sont pas tous aisément comparables. Cependant on peut classer les méthodes existantes en catégories selon sur quel principe elles se reposent.

Certains types d'algorithmes de clustering peuvent reposer sur des hypothèses sur la distribution des individus à classifier. Les groupes présents au sein de l'échantillon sont alors supposés suivre une loi connue, où seuls les paramètres de cette loi demeurent inconnus et sont à déterminer. On appelle ces dernières les méthodes paramétriques car définir un groupe revient à en définir ses paramètres (moyenne et variance pour une loi normale par exemple). Les chances pour un individu d'être affecté à un groupe donné pourront alors être calculées de manière probabiliste, donnant des probabilités d'appartenance à chaque groupe plutôt qu'une attribution exclusive, induisant ainsi un clustering qualifié de « doux ». Les algorithmes de type « Éspérance-Maximisation » en sont de bons représentants.

Le second type de méthodes, non-paramétriques, ne reposent sur aucune hypothèse vis-à-vis des distributions et donnent lieu à des méthodes variées se reposant sur des concepts diversifiés. Les résultats qu'ils produisent peuvent aussi être de plusieurs

formes. Sous forme de partitions, où chaque élément possède une classe bien définie (« flat clustering »en anglais, opposé au clustering « doux » cité plus haut), ou sous une forme hiérarchique à laquelle on doit appliquer une étape supplémentaire pour en obtenir les partitions finales.

En fonction de l'approche choisie pour la modélisation (paramétrique / non-paramétrique) et le type de données étudiées, on se place dans une famille particulière d'algorithmes de clustering dont voici une liste non-exhaustive.

On peut tout d'abord compter les méthodes basées sur des centroïdes comme k-means [36], k-medoïds [23], « vector quantization » [19] ou mean-shift [9] pour ne citer que les pus connus. Comme leur nom l'indique, elles ont la particularité d'être toutes basées sur une notion de centre de masse, placés aléatoirement à l'initialisation, qui vont idéalement se déplacer graduellement vers les centres de masse réellement présents dans le jeu de données. Ces méthodes, qui sont non-paramétriques, supposent que les groupes sont concentriques, c'est à dire semblables à un amas gaussien.

Les méthodes dites à regroupement hiérarchiques [24] produisent des hiérarchies qui mettent en évidence la force des liens entre les différentes observations ce qui permet ensuite d'en extraire des partitions adéquates. Elles peuvent être ascendantes ou descendantes. Les ascendantes partent de l'état initial où chaque observation définit un cluster puis le nombre de clusters est ensuite réduit de manière itérative en fusionnant les groupes les plus proches jusqu'à arriver à un seul groupe. Ensuite revient le choix de l'expert (ou de l'algorithme lui-même) de choisir quelles sont les meilleures partitions parmi l'ensemble des possibles fournies par la hiérarchie. La représentation de la hiérarchie se fait très souvent sous la forme d'un dendrogramme (un exemple est donné figure 1.1). Les descendantes suivent le même principe mais partent d'un seul cluster auquel toutes les observations appartiennent initialement pour le subdiviser progressivement. Les méthodes appartenant à cette catégorie diffèrent à la fois sur le choix de la métrique (ou mesure de dissimilarité) et sur la stratégie d'agglomération ou d'élagage. L'un des meilleurs représentants actuels est HDBSCAN [7], dont une section complète est dédiée dans le chapitre suivant.

On peut distinguer les algorithmes de mixtures de lois [61] qui sont de nature probabiliste et paramétriques. Ces algorithmes supposent que les clusters suivent chacun une loi particulière dont l'objectif est de déterminer les paramètres. Dans

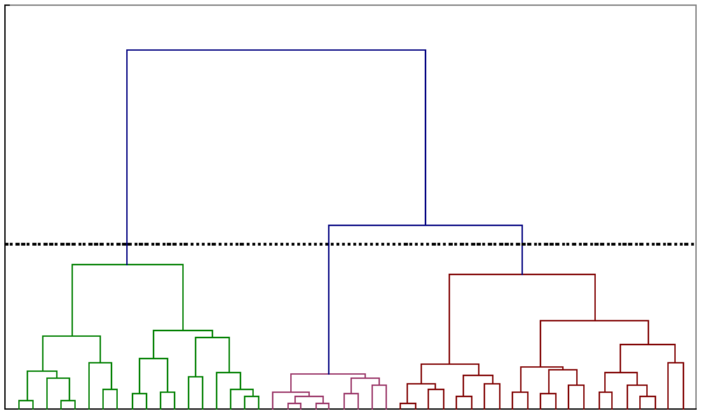


FIGURE 1.1 – Exemple de hiérarchie sous forme standard de dendrogramme. La section horizontale indique à quel niveau la hiérarchie les subdivisions ne comptent plus, ce qui produit dans cet exemple trois partitions.

le cas de mixtures gaussiennes par exemple, il s'agira de déterminer les paramètres moyennes μ_i et variances σ_i de chaque groupe i. Dans cette catégorie on retrouve entre autres l'algorithme EM [11, 37], la « bottleneck information method » [56] et les algorithmes bayésiens basés sur les processus de Dirichlet [16, 46].

Il existe la famille des algorithmes basés sur une estimation non-paramétrique de la densité qui considèrent les clusters comme étant des zones denses séparées par des régions de relativement faible densité. On retrouve dans cette catégorie une importante variété d'algorithmes tant les modélisations sont diverses. Les densités estimées, ou des mesures reflétant la densité locale, peuvent être obtenues de plusieurs façons : avec des estimateurs à noyau [50], en découpant le domaine en une grille et en supposant chaque cellule uniforme, ou encore grâce aux k plus proches voisins. De plus les densités obtenues peuvent être exploitées de manières très différentes. Ces approches à densité ont comme particularité de souvent pouvoir détecter et reconstruire de manière effective les clusters de forme arbitraire. DBSCAN et OPTICS [2] en sont les représentants les plus classiques. L'algorithme FDPC [48] et notre algorithme Ballstering font également partie de cette catégorie.

Certaines méthodes ont pour objectif d'analyser des données sous forme de graphe. Elles se partagent en deux types qu'il ne faut pas confondre, bien qu'elles utilisent toutes deux la même théorie des graphes. L'une cherche à trouver les groupes au sein

d'un graphe, tandis que l'autre cherche à classer un ensemble de graphes, où chaque graphe représente une observation. Dans le premier cas on cherche à classer les nœuds du graphe qui se ressemblent, donc agréger les nœuds qui sont fortement connectés au sein du graphe global. Cela revient à faire en sorte d'avoir un minimum d'arêtes connectant un groupe à l'autre [65]. Dans le second cas on fera appel à la théorie des graphes afin d'obtenir une notion de similarité inter-graphe par exemple [6, 47] pour ensuite appliquer des algorithmes plus classiques en exploitant ces mesures de similarité.

Des algorithmes sont basés sur les méthodes d'optimisation déterministes ou stochastiques. Les représentants sont les séparateurs à vaste marge [63], machines à vecteur support [20] et les méthodes discriminatoires [25].

Et enfin toutes les méthodes se basant sur des réseaux de neurones qui ont débuté entre autres avec les cartes auto-adaptatives [27] et sont en pleine expansion de nos jours.

Cette discipline possède beaucoup d'applications concrètes et utiles dans de nombreux domaines comme la segmentation d'image par exemple. Identifier les pixels homogènes (proches en couleur et peu espacés) et les catégoriser permet de compresser des données spatiales. On peut considérer en guise d'exemple les photos du clown dans la figure 1.2 où l'image de droite a été compressée. Les zones homogènes sont segmentées et classées dans une même catégorie qui prend alors comme valeur la moyenne des attributs. Des techniques similaires de segmentation peuvent être utilisées en médecine [43] ou dans le traitement de photos aériennes [54] afin de mettre en évidence certaines zones d'intérêt, comme les tumeurs en ce qui concerne la médecine notamment.

On la retrouve aussi dans les problèmes de reconnaissance de formes [67], la reconnaissance vocale [49] et d'écriture manuscrite [21].

Le clustering est aussi présent pour ce qui relève de la segmentation de base de données qui permet de discrétiser une base de données afin de l'indexer. L'indexation peut permettre un accès plus rapide aux données mais peut aussi servir à classer une population en sous-groupes afin de faciliter la gestion de la relation client par exemple.

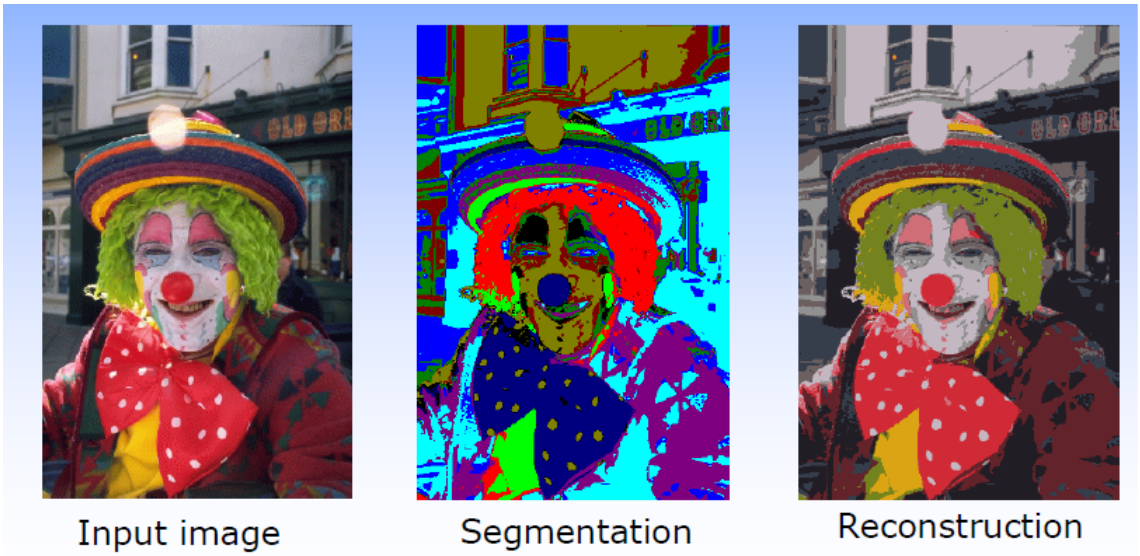


FIGURE 1.2 – Segmentation d'image effectuée grâce au clustering. De gauche à droite : la photo initiale, la mise en évidence des partitions obtenues via segmentation, et enfin l'image compressée.

Enfin on peut vouloir produire des partitions pour l'extraction de connaissances qui permet de découvrir sans aucun objectif à priori et de mettre en évidence des tendances et corrélations invisibles au premier abord. Cela permet également de fournir des moyens de visualisation de bases de données volumineuses et complexes comme les enregistrements des caméras, des comportements sur internet, sur la génétique et bien d'autres domaines sont sources de données de plus en plus volumineuses. Afin de pouvoir archiver et utiliser ces données nous avons besoin d'outils de visualisation et de catégorisation.

1.2 Fast Density Peak Clustering (FDPC)

L'algorithme de clustering FDPC [48] est un algorithme basé sur la densité capable de mettre en évidence des clusters non-convexes en donnant une règle de décision pratique pour identifier les groupes de manière pertinente.

Tout comme pour l'algorithme Mean-Shift [9], les centres de clusters, que nous préférons appeler cœurs de clusters (car ils ne sont pas obligatoirement au centre de leur cluster) sont définis comme des maximums locaux de densité. Autrement dit, un cœur de cluster est défini comme un point de relativement haute densité dont tous les voisins ont une densité plus faible et qui est éloigné de tout autre maximum local.

Cette innovante méthode qui a suscité l'intérêt de nombreux chercheurs (en moyenne une citation tous les trois jours depuis sa parution) présente à la fois des résultats très intéressants mais aussi des limitations très sévères.

Son premier inconvénient est que sa complexité est telle qu'il est impensable de l'appliquer sur des jeux de données très volumineux. Aussi sa manière d'estimer la densité se fait à travers un paramètre d_c très difficile à choisir et qui a pourtant un impact très important sur les résultats. Enfin, la phase de sélection des cœurs nécessite l'expertise humaine pour être pertinente et peut malgré tout provoquer des erreurs. De plus l'affiliation des points non-cœurs aux clusters n'est pas plus robuste.

Beaucoup de chercheurs se sont penchés sur cet algorithme en vue de l'améliorer ou le modifier légèrement afin de pouvoir l'appliquer à certains problèmes spécifiques. Il a été utilisé avec succès pour une étude de comportements de consommation d'électricité [60], pour du clustering de textes [34], pour partitionner des sons via décomposition en sous-mots [64], pour la modélisation et surveillance de processus de production industriels [44], sur des données médicales [30] et a été légèrement modifié pour travailler à la segmentation d'images [18, 52, 8], pour la détection d'outliers [13] et enfin pour la sélection de bandes hyper-spectrales [22]. Dans [40] on mène une étude comparative de k-means et FDPC et conclut que ce dernier est moins rapide mais fournit un clustering plus qualitatif.

1.2.1 Fonctionnement et limites

Phase 1 : estimation des densités

Considérons l'échantillon de points $(x_i)_{i=1\dots N} \in R^p$. Lors de la première phase de FDPC, les distances euclidiennes d_{ij} entre tous les couples $\{(x_i, x_j), i \neq j\}$ doivent être calculées. Ensuite un paramètre d_c qui reflète une distance est fixé afin de compter, pour chacun des x_i, le nombre de points qui sont à une distance inférieure ou égale à d_c. Ce compte, pouvant être représenté comme le nombre de points appartenant à la boule $B(x_i, d_c)$ de centre x_i et de rayon d_c, sert de mesure de la densité locale. On l'appelle donc compte de densité et il est explicitement déterminé par :

$$\rho_i := \rho(x_i) = \sum_{j=1}^{N} (\mathbf{1}_{x<0}(d_{ij} - d_c)) \tag{1.1}$$

Remarquons que nous pouvons réécrire cette formule sous la forme standard d'un estimateur de densité non-paramétrique

$$\sum_{j=1}^{N} K\left(x_i - x_j\right),$$

où K est une fonction noyau qui ici se réduit à la fonction caractéristique d'une boule $1_{B(0,d_c)}$.

Phase 2 : détection des cœurs et clustering

L'étape suivante consiste en la recherche, pour chaque point x_i, du point le plus proche parmi ceux de plus haute densité. On l'appellera NPHD pour « Nearest Point of Higher Density ». La distance séparant x_i de son NPHD, qui est notée δ_i et s'obtient grâce à la formule

$$\delta_i = \min_{j:\rho_j > \rho_i} (d_{ij}),$$

est utilisée conjointement aux comptes de densité pour trouver les cœurs de clusters.

En effet, une fois ces deux variables obtenues pour chaque observation, on trace le nuage de points appelé graphe de décision, avec en abscisse les densités ρ_i et en ordonnée δ_i. Les cœurs de clusters sont alors définis comme les outliers dans ce graphe. Un exemple d'un tel graphe est donné dans l'image du milieu de la figure 1.3. En d'autres termes, les cœurs sont les points possédant à la fois une très haute densité, mais aussi une grande distance δ_i. Le point de plus haute densité x_k ne possède pas de NPHD donc par convention est fixé à $\delta_k = \max_j(d_{ij})$.

Une fois les cœurs déterminés, les autres points non-cœurs sont attribués récursivement au même cluster que leur NPHD. L'algorithmique entière de FDPC est détaillée dans l'Algorithme 1 et un exemple des résultats qu'il peut obtenir est illustré dans la figure 1.3.

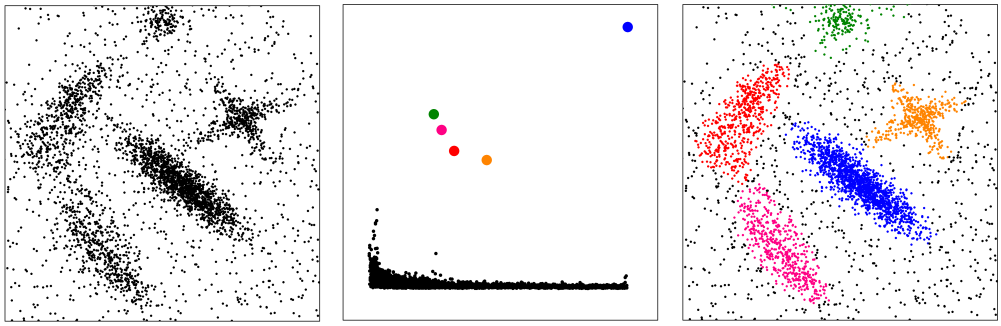


FIGURE 1.3 – Exemple de clustering de FDPC (Fast Density Peak Clustering). De gauche à droite : le jeu de données initial, le graphe de décision qui sert à détecter les cœurs de clusters et la partition finale.

Algorithm 1 FDPC

for all $(i, j) \in \{ 1, ..., N\}^2$ **do**

$d_{ij} \leftarrow \text{distance}(x_i, x_j)$

end for

for all $i \in \{1, ..., N\}$ **do**

$\rho_i \leftarrow \sum_{j=1}^{N} (\mathbf{1}_{x<0}(d_{ij} - d_c))$

end for

for all $j \in \{1, ..., N\}$ **do**

$\text{nphd}_j \leftarrow \text{argmin}_{k:\rho(x_k)>\rho(x_j)}(d_{jk})$

$\delta_j \leftarrow d_{j,\text{nphd}_j}$

end for

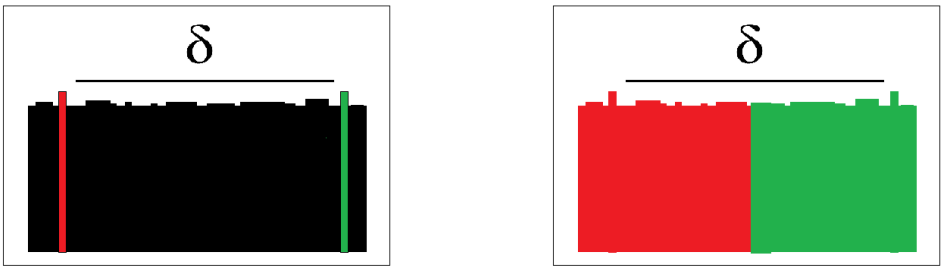
Cluster cores $\leftarrow$ outliers in decision graph

$\forall j$, $\text{Label}_j \leftarrow \text{Label}_{\text{nphd}_j}$

Comme nous l'avons déjà dit, cet algorithme montre des résultats intéressants mais souffre de nombreux problèmes très contraignants. Voici les problèmes que nous avons pu identifier concernant FDPC :

— Dès la première étape FDPC doit calculer et enregistrer autant de distances qu'il y a de couples de points dans l'échantillon ce qui mène à une complexité de l'ordre de $O(N^2)$. Ce premier point limite déjà cet algorithme à de relativement petit jeux de données.

— Le choix de d_c est difficile alors qu'il a un impact très important sur toutes les étapes de l'algorithme. De plus puisque d_c est statique, cette méthode est



(A) Mauvaise détection des cœurs (B) Mauvais clustering induit

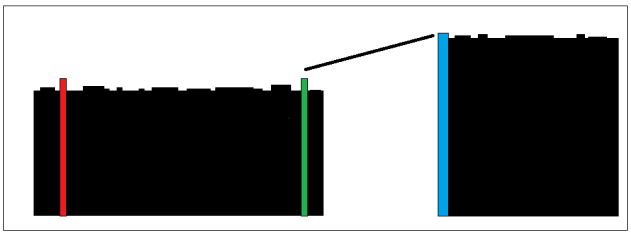
FIGURE 1.4 – Mauvaise détection des coeurs par FDPC. Dans cet exemple où δ est grand, deux coeurs sont détectés au lieu d'un seul.

incapable de gérer correctement à la fois les zones denses et les zones peu denses. En effet, s'il est bien choisi pour représenter une zone dense donnée, il sera alors trop petit pour bien représenter les zones moins denses (et vice versa).

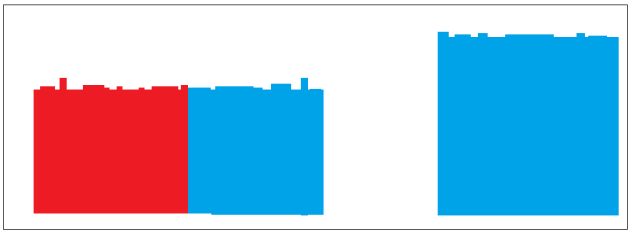
— Aucune règle de décision automatique satisfaisante n'est donnée pour détecter les cœurs dans le graphe de décision. Dans l'illustration de la figure 1.3 les cœurs ressortent très bien et sont bien séparés des non-cœurs mais dans de nombreux cas réels il y a toute une continuité entre cœurs et non-cœurs. Inévitablement le choix des cœurs est donc relativement approximatif dès que le jeu de données monte en complexité.

— De plus, certains points peuvent avoir une grande densité et un grand δ à la fois, donc détectés en tant que cœurs de cluster, alors qu'ils ne doivent clairement pas être considérés comme tels, mettant directement en cause leur définition de cœur de cluster. C'est le cas dans l'exemple illustré dans la figure 1.4 où FDPC détecte deux cœurs de clusters au lieu d'un seul (ce dernier suivant une distribution uniforme $1d$). En effet dans cet exemple où le point en rouge à gauche (fig A) ne se démarque des voisins qu'à cause de l'aléa, il se retrouve identifié à tort comme un cœur de cluster, à la condition que δ soit considéré comme grand.

— Enfin, la définition de NPHD utilisé dans FDPC qui lie deux points pour en déduire la distance qui les sépare δ ne prend pas en compte la densité qui sépare ces deux points. Ce problème peut mener à de sérieuses erreur d'affectation comme on peut le voir illustré dans la figure 1.5. On suppose pour cet exemple que FDPC ne s'est pas trompé et a détecté correctement les deux cœurs de



(A) Le point correspondant au bleu détecte à tort le point bleu en tant que NPHD.



(B) Mauvais clustering induit

FIGURE 1.5 – Illustration des mauvaises détections des NPHD et des implications. Il associe la moitié du cluster de gauche au cluster de droite.

cluster, le rouge et le bleu. Avec leur définition de NPHD, le point vert trouve comme NPHD le point bleu car il est pus proche que le rouge, et donc hérite du label du cluster bleu et le propage récursivement de NPHD en NPHD jusqu'à la moitié du cluster de gauche.

1.2.2 Extensions récentes de FDPC

Afin de dépasser certaines des limitations soulignées plus haut, un nombre de travaux non négligeable ont été effectués dans le but d'améliorer l'algorithme originel. La plupart ne se concentrent que sur un seul des problèmes à la fois, suivant l'application qu'ils veulent en faire.

Gérer de gros volumes de données

La seule méthode ayant pour but d'améliorer FDPC sur ce point est une extension proposée par [33] dans le but de traiter de nombreuses trajectoires de taxis (données GPS bi-dimensionnelles). L'approche ici consiste en la projection du jeu de données pour former une image de la densité qui est utilisée pour dessiner le contour des clusters. Cette méthode a montré de bon résultats mais est limité à des jeux de données bi-dimensionnels uniquement.

Le choix de d_c

Le choix de base pour d_c faisant apparemment consensus est de le fixer à la valeur telle que le compte de densité moyen induit se trouve égal à 2% de N ($\bar{\rho} = 2\%N$ selon [13]). Un autre choix moins trivial et basé sur l'entropie a été proposé par [58], mais ce dernier coûte un nombre considérable d'opérations le rendant inapplicable dès que N devient grand. Il existe deux méthodes qui remplacent l'estimation de la densité (1.1) par une non-paramétrique, évitant ainsi le choix de d_c :

— [59] estime la densité locale grâce à un estimateur à noyau multivarié. Pour chaque observation x_i la densité locale est calculée à travers la formule :

$$\rho_i = \frac{1}{n\prod_{l}^{d} h_l} . \sum_{j=1}^{n} K(\frac{x_{i1} - x_{j1}}{h_1}, ..., \frac{x_{id} - x_{jd}}{h_d}),$$

où d est le nombre de dimensions, K est un noyau Gaussien et $\{h_1, ..., h_d\}$ sont les fenêtres dans chaque dimension choisies automatiquement et de manière adaptative.

— Dans [38], on estime la densité avec un autre estimateur de densité non paramétrique qui est basé sur les équations de la chaleur qui permet une meilleure précision mais est bien plus cher en terme de calculs.

Sélection des cœurs

Dans le papier de [35], les points de l'échantillon sont triés selon leur valeur γ_i, définie par le produit $\gamma_i = \delta_i.\rho_i$, puis les m points possédant la plus grande valeur γ_i sont sélectionnés pour les soumettre à un test supplémentaire.

Parmi ces points pré-sélectionnés, ceux qui ont un γ_i supérieur à un certain « truning-point » sont choisis pour devenir des cœurs de cluster. Ce « truning-point » est obtenu grâce à la formule :

$$\text{argmax}_{i\in\{1,...,m\}}(\frac{k_i^1}{k_1^{i-1}}), \text{ où } k_i^n = \frac{\gamma_{i+n} - \gamma_i}{n}.$$

[39] et [13] définissent la grandeur de δ_i relativement à la variance de l'ensemble des δ_j. Ainsi δ_i est considéré comme grand par les premiers auteurs si $\delta_i > 2.\text{sd}(\delta)$

(et $\delta_i > 3.\text{sd}(\delta)$ selon [13]).

Les deux s'accordent à considérer la densité comme haute si elle dépasse la moyenne, c'est à dire si $\rho_i > \text{mean}(\rho)$. De plus ils ajoutent une étape d'agrégation des clusters : Deux clusters seront agrégés s'il existe un point x qui appartient au premier cluster et un point y qui appartient au second tels que la distance $d(x, y)$ soit inférieure à d_c.

Un autre algorithme basé sur d_c est présenté par [32]. Afin de prendre en compte la densité avoisinante lors de la sélection des cœurs, les concepts de DBSCAN [15] sont couplés à la « divide and conquer strategy ». Les cœurs potentiels sont soumis à un test supplémentaire : si le point considéré est densité-atteignable depuis un cœur de cluster alors ce point est ignoré et ne deviendra pas cœur de cluster.

[57] proposent un compte de densité légèrement différent basé sur les k plus proches voisins. Le compte de densité change pour être défini par

$$\rho_i = \frac{[K]}{\sum_{j=1} K d_{ij}}$$

.

Ensuite ils prouvent que l'ensemble des produits $\gamma = \rho.\delta$ suit une distribution à queues épaisses pour enfin appliquer un test statistique à ce modèle afin de trouver les outliers de l'échantillon, qui définissent donc les cœurs de clusters.

Autres améliorations

[62] utilise les k plus proches voisins pour améliorer la règle d'affiliation aux clusters des points restants, qui ne définissent pas un cœur de cluster. Donc après la phase de détection des cœurs, les outliers restants qui possèdent un grand $\delta_i^K = \max_{j \in \text{KNN}_i}(d_{ij})$ sont supprimés et considérés comme du bruit.

Ensuite les points restants sont assignés à travers une certaine stratégie : le label d'un cœur x_c est propagé sur ses voisins non affiliés qui sont à une distance inférieure à $\sum_{j \in KNN_c} d_{cj}/K$ et ces points récemment affiliés propagent récursivement leur label selon la même logique.

Les points qui ont survécu à cette première passe sont soumis à une deuxième stratégie dont l'aspect principal réside en l'apprentissage de la probabilité p_i^c qu'un

point x_i appartienne à au cluster c. Après avoir établi ces probabilités, chaque x_i est naturellement assigné au cluster le plus probable.

Un dérivé de FDPC qui peut gérer des densités inégales a été proposé par [66]. Les densités ρ_i et distances δ_i sont remplacées par des variables au comportement similaires.

Une notion d'attraction mutuelle entre deux points est proposée et donnée selon la formule :

$$f_{ij} = \min(f_{i\to j}, f_{j\to i}), \text{ où } f_{i\to j} = e^{-\frac{d(i,j)^2}{2\sigma_i^2}}.$$

Le paramètre σ_i, qui contrôle la largeur de la fenêtre de cette fonction-noyau, est calculée localement à travers la formule :

$$\sigma_i = \frac{\sum_{j\in J} d(i,j)}{\text{card}(J)}, \text{ où} J = \{j | d(i,j) \leq d_c\}.$$

Les densités ρ_i sont alors remplacées par les scores d'influence :

$$\rho_i = \frac{\sum_{j\in A} f_{i,j}}{\text{card}(A)}$$

et les δ_i sont calculés avec :

$$\delta_i = \min_{j:\rho_j>\rho_i} \frac{d(i,j)}{\min(\sigma_i, \sigma_j)}.$$

Grâce à ce procédé, les point situés aux zones très peu denses ont une sphère d'influence plus large que ceux situés aux zones denses, rendant l'algorithme capable de gérer des clusters de densités non-homogène.

1.3 *k*-means

Avec ses 60 ans d'ancienneté, *k*-moyennes, mais plus souvent appelé *k*-means [36], est l'un des plus connus et des plus anciens algorithmes de classification automatique

non-supervisée. Cette heuristique est vouée à résoudre le problème de partitionnement modélisé de la manière suivante.

Soit $(x_1, ..., x_N)$ l'échantillon constitué de N points que nous voulons partitionner. Pour un paramètre k fixé par l'utilisateur, l'algorithme k-means cherche à partitionner les observations x_i en k ensembles $\mathbf{C} = \{C_1, ..., C_k\}$ de manière optimale, au sens qu'il minimise la formule

$$\underset{\mathbf{C}}{\operatorname{argmin}} \sum_{j=1}^{k} \sum_{x_i \in C_i} ||x_i - \mu_j||^2, \tag{1.2}$$

où μ_j correspond à la moyenne du $j^{\text{ème}}$ cluster C_j.

Pour résoudre ce problème grâce à l'heuristique k-means on commence par choisir le nombre de groupes k et initialiser les moyennes correspondantes $\{\mu_1^0, ..., \mu_k^0\}$ de manière aléatoire.

Puis la procédure suivante est répétée jusqu'à convergence, c'est-à-dire lorsque les itérations n'améliorent plus la solution. Cela consiste en deux étapes successives à savoir la ré-affectation de chaque observation au cluster qui lui est le plus proche

$$C_j^{t+1} = \{x_i : ||x_i - \mu_j^t|| \leq ||x_i - \mu_l^t||, \forall l \in \{1, ..., k\}\},$$

suivi d'une mise à jour des moyennes en fonction des nouvelles affectations

$$\nu_j^{t+1} = \frac{1}{|C_j^t|} \sum_{x_i \in C_j^t} x_i.$$

L'initialisation des moyennes joue un rôle important sur les résultats en sortie. Si elles sont mal initialisées, on risque de faire converger l'algorithme sur un minimum local potentiellement éloigné du minimum global. La manière basique (méthode de Forgy) est d'initialiser les k moyennes initiales à k données d'entrée choisies aléatoirement. Des variantes de l'algorithme de base ont donc été proposés comme k-means++ [3] dont la stratégie d'initialisation est également de choisir aléatoirement mais en abaissant les chances de choisir un point proche des centres déjà initialisés par exemple.

Les moyennes peuvent également être remplacées par des médianes [26] ou encore par la somme de fonctions variées comme l'algorithme Fuzzy C-means [14] où l'appartenance n'est plus exclusive mais traduite par un degré d'appartenance.

Les distances aux moyennes aussi peuvent être remplacées par une probabilité d'appartenance au groupe qui est supposé gaussien, ce qui donne lieu aux algorithmes de type Espérance-Maximisation EM [11].

Cet algorithme et ses variantes sont très efficaces lorsque les données sont réparties en amas gaussiens mais leur efficacité s'effondre dès que les clusters sont de forme arbitraires, particulièrement pour des clusters non-concentriques.

Aussi, le comportement de cet algorithme peut énormément varier selon le paramètre k qui est bien souvent difficile à déterminer compte tenu du cadre non-supervisé.

Cependant la complexité de cet algorithme est exemplaire lorsqu'elle utilise les arbres k-d (voir section 2.1.1) puisqu'elle se trouve ainsi égale à $O(kTN)$ où T est le nombre d'itérations et N le nombre d'observations. Ainsi, bien implémentée, cette heuristique se trouve être une des plus rapides ce qui participe en partie à sa renommée et celle de ses variantes.

1.4 DBSCAN et HDBSCAN

1.4.1 DBSCAN

DBSCAN [15] est un algorithme de regroupement basé sur la densité qui a été pensé pour découvrir des partitions de forme arbitraire. Un point important dans la compréhension de cet algorithme est que pour appartenir à un cluster un objet doit être un objet cœur, c'est à dire que chaque élément doit contenir un certain nombre minimum de points dans son voisinage. Les points sont liés entre eux au sein d'un même cluster s'ils possèdent ces cœurs en tant que voisins.

Cette idée repose donc sur une notion de voisinage et sur un seuil qui détermine la densité minimum acceptable qui seront définis plus bas. Il ne possède officiellement qu'un paramètre à choisir (bien qu'un second apparaisse finalement mais est fixé de manière hasardeuse par les auteurs) et une aide est fournie afin d'effectuer ce choix.

Cet algorithme introduit entre autres les quatre définitions suivantes qu'il faut expliciter avant d'aller plus loin dans les explications.

— **Objet cœur** : $\mathbf{x}_p$ est appelé objet cœur si son nombre de voisins à une distance inférieure à Eps, est supérieur à m_{pts}, le paramètre à choisir. De manière formelle, $\mathbf{x}_p$ est un objet cœur si

$$|\mathbf{N}_{Eps}(\mathbf{x}_p)| \geq m_{pts},$$

où

$$\mathbf{N}_{Eps}(x_p) := \{x \in \mathbf{X} | d(x, x_p) \leq Eps\}$$

— **Directement Densité-Atteignable** : Une observation quelconque x_p est directement densité-atteignable depuis un cœur x_q si $x_q \in N_{Eps}(x_p)$. Seul un cœur peut directement atteindre un autre point. Notez donc que cette notion est asymétrique, c'est à dire que si x_p est directement densité-atteignable depuis x_q, x_q n'est pas forcément directement densité-atteignable depuis x_p si ce dernier n'est pas cœur.

— **Densité-Atteignable** : Deux objets sont densité atteignable s'il existe une chaine de points deux à deux directement densité-atteignables qui lie ces deux objets.

— **Densité-Connectés** : Deux objets sont densité-connectés s'ils sont tous deux densité-atteignables depuis un tierce objet.

— **Cluster** : Un cluster **C** est un sous ensemble maximal de **X** tel que tous les points de **C** sont densité-connectés deux à deux. Un ensemble est dit maximal si tout point densité-atteignable depuis un élément de cet ensemble appartient aussi à ce même cluster.

— **Bruit** : Une observation est considérée comme du bruit si elle n'appartient à aucun cluster

Pour les paramètres Eps et m_{minpts} donnés, l'algorithme DBSCAN fonctionne comme suit. Dans un premier temps on détermine pour chaque objet s'il est un objet-cœur. Ensuite, un premier point p de l'échantillon est choisi arbitrairement puis est

construit l'ensemble des points densité-atteignables depuis ce point p. S'il s'avère être un point-cœur alors lui et les autres points densité-atteignables définissent un cluster et y sont assignés. Si on ne peut atteindre aucun autre point depuis p alors il est juste ignoré et le point suivant est étudié.

Certains clusters, ceux très proches l'un de l'autre, peuvent passer inaperçus lors du premier passage. En effet lorsque m_{minpts} est élevé certaines variations de densité assez fines ne sont pas pas détectés ce qui a tendance à regrouper plusieurs clusters en un seul.

Afin de les détecter tous, ils proposent de relancer localement l'algorithme de manière récursive avec uniquement les données appartenant à certain clusters avec un paramètre m_{minpts} plus élevé. Cette solution peut paraître chère en terme de calculs, mais ce n'est apparemment pas si grave car ils la disent rapide et nécessaire uniquement pour certains clusters, ceux susceptibles d'en abriter plus d'un, et qu'ils assurent aisément identifiables. Cela permet d'utiliser un m_{minpts} adapté à la zone étudiée pour faire apparaitre les sous clusters.

Son algorithmique est donnée dans la figure C.1 où $SetOfPoints$ peut être l'ensemble du jeu de données comme il peut être un sous ensemble définissant un cluster à potentiellement séparer. la partie principale de DBSCAN est contenue dans la fonction $ExpandCluster$ qui est explicitée dans la figure C.1 (annexe C).

Passons au choix des paramètres. m_{minpts} est fixé à 4 car à partir de cette valeur, et pour toutes les valeurs supérieures, les résultats ne varient pas de manière significative, tandis que le temps de calcul lui augmente très rapidement.

Pour choisir Eps ils définissent d'abord la fonction $k\text{-}dist(p)$ qui renvoie la distance telle que exactement $k+1$ points sont à une distance inférieure à $k\text{-}dist(p)$ de p. Ensuite à chaque point est associé sa distance $k\text{-}dist$ et ils sont rangés par ordre décroissant. Ceci produit un graphe (voir exemple 1.6) dans lequel on peut voir une « vallée » qui permet de discriminer le bruit des clusters. Il est difficile de trouver automatiquement quand commence cette « vallée » donc on demande à l'utilisateur de déterminer à quel point $p*$ elle commence.

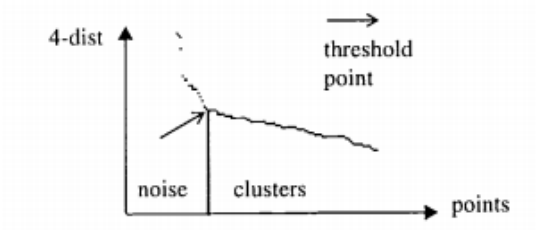


FIGURE 1.6 – DBSCAN : Choix du paramètre *Eps*.

Ainsi, la valeur de *Eps* retenue est $k\text{-}dist(p*)$. Cette heuristique dépend directement de k qui devrait être choisi avec soin. D'après leur expérience (sur des clusters contenant peu de points et en deux dimensions uniquement) on peut le fixer à $k = m_{minpts} = 4$ pour des résultats corrects. Aucune indication n'est fournie pour des jeux de données plus complexes.

1.4.2 HDBSCAN

HDBSCAN [7] est un algorithme qui conserve les concepts et définitions de DBSCAN mais les modifie pour en faire une version hiérarchique qui permet de se passer du choix du paramètre crucial qu'est *Eps*. Mieux, il fournit sous la forme d'un dendrogramme toutes les solutions de DBSCAN pour tous les *Eps* possibles, et le meilleur *Eps* est choisi localement sur chaque sous-branche grâce à une optimisation sur ces arbres hiérarchisés.

Dans un premier temps il est expliqué comment, à partir des concepts de DBSCAN et en les adaptant, on construit un graphe de proximité puis en extrait un arbre couvrant de poids minimum G qui possède des propriétés intéressantes. En effet, en supprimant itérativement les arêtes de cet arbre par ordre de poids décroissant et en retenant l'état des partitions à chaque retrait d'arête, on forme un dendrogrammme dont chaque niveau correspond à une partition qui serait donnée par DBSCAN avec le *Eps* correspondant.

Ce dendrogramme est ensuite simplifié pour mettre en avant les apparitions et disparitions de clusters au fur et a mesure que le niveau λ diminue, associant à chaque cluster potentiel une mesure de stabilité pour enfin maximiser la stabilité générale à travers un problème d'optimisation en $\{0, 1\}$ classique [28].

Décrivons maintenant en détail le fonctionement de cet algorithme. Soit $\mathbf{X} = \{\mathbf{X_1}, ..., \mathbf{X_N}\}$ le jeu de données constitué de N observations. Certaines définitions de DBSCAN sont reprises et leur nom modifiés pour certains dans le but de gagner en cohérence. La plupart d'entre elles ne sont que très légèrement remaniées par HDBSCAN. Ils ont principalement simplifié les définitions en écartant les objets non-cœurs des autres définitions, laquelle posait des problèmes de symétrie. Les objets non-cœurs ne sont désormais comptés que comme du bruit et *Eps* est noté ϵ.

1 : Objet cœur : $\mathbf{x}_p$ est appelé objet cœur si son nombre de voisins à une distance inférieure à ϵ est supérieur à m_{pts}. De manière formelle, $\mathbf{x}_p$ est un objet cœur si $|\mathbf{N}_\epsilon(\mathbf{x}_p)| \geq m_{pts}$ où $\mathbf{N}_\epsilon(x_p) := \{x \in \mathbf{X} | d(x, x_p) \leq \epsilon\}$

2 : ϵ-Atteignable : deux observations x_p et x_q sont ϵ-atteignables si $x_p \in \mathbf{N}_\epsilon(x_q)$ et $x_q \in \mathbf{N}_\epsilon(x_p)$. Cette définition découle de la définition "Directement Densité-Atteignable" de DBSCAN. Comme on peut le constater, on ne fait plus attention si les objets sont des cœurs ou pas. Ceci rend la notion de ϵ-Atteignable symétrique ce qui permet d'obtenir une notion de "Densité-Connectés" plus simple.

3 : Densité-Connectés : Deux objets cœurs sont densité-connectés si ils sont directement ou transitivement (de cœurs en cœurs) $\epsilon-$atteignables. Rq : cette définition ne s'applique qu'à des objets cœurs, contrairement à celle de DBSCAN qui s'applique aussi aux non cœurs.

4 : Cluster : Un cluster $\mathbf{C}$ est un sous ensemble maximal de $\mathbf{X}$ tel que tous points de $\mathbf{C}$ sont densité-connectés deux à deux. Cette définition ne change pas si on omet le changement de définition de densité-connecté sur laquelle on s'appuie pour définir les clusters. Si on considère le graphe formé par les sommets $\mathbf{X}$ dans lequel deux éléments sont adjacents si ils sont ϵ-atteignables, on peut définir les clusters comme étant chacune des ses composantes connexes.

5 : Bruit : Une observation est considérée comme du bruit si elle n'est pas cœur.

6 : Distance-Cœur : la distance-coeur d'une observation est la plus grande des distances à ses m_{pts} plus proches voisins. Elle est notée $d_{core}(x_p)$.

7 : ϵ-Objet Cœur : x_p est un $\epsilon-$objet coeur si $d_{core}(x_p) < \epsilon$

8 : Mutual Reachability Distance : La "Mutual reachability distance" entre deux points x_p et x_q est définie par la formule

$$d_{mreach}(x_p, x_q) = max(d_{core}(x_p), d_{core}(x_q), d(x_p, x_q)). \tag{1.3}$$

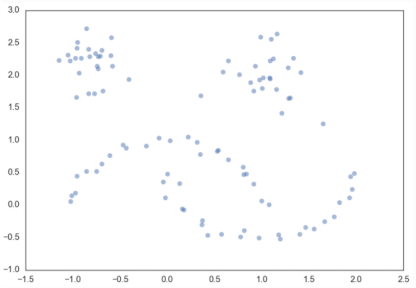
9 : Le graphe de mutual reachability : C'est le graphe $G_{m_{pts}}$ dans lequel les sommets sont les éléments de $\mathbf{X}$ dont toute paire de sommets est connectée par une arrête (graphe complet) ayant un poids correspondant à leur "mutual reachability distance".

L'algorithme commence par calculer les distances "mutual reachability" pour toutes les paires de points formées par les éléments de $\mathbf{X}$. Pour cela on calcule les distances-cœur d_{core} de chaque points qui dépendent directement du paramètre m_{pts} qu'il faut donc choisir au préalable. Puis, pour chaque paire de points (x_i, x_j), on établit $d_{mreach}(x_i, x_j)$ avec la formule 1.3.

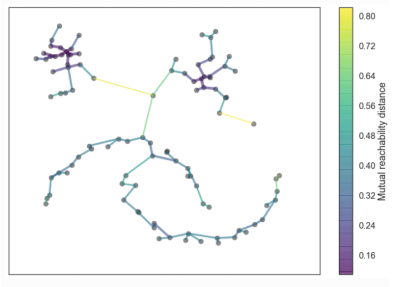
Cela permet de construire le graphe $G(\mathbf{X}, R)$ ayant en sommet les points de l'échantillon $\mathbf{X}$ et pour poids d'arêtes les distances « mutual reachability » correspondantes. Ce graphe est très complexe car le nombre de nœuds est élevé et chaque nœud est adjacent à tous les autres nœuds, c'est à dire que pour toute paire de nœuds il existe une arête qui les relie directement. On dit que ce graphe est complet et il possède $N(N-1)$ arêtes.

À partir de ce graphe, on en extrait un arbre couvrant de poids minimum qui est le graphe possédant les mêmes nœuds mais pour lequel on a gardé qu'un sous ensemble d'arêtes [5]. Ce nouveau graphe est tel que tout nœud peut être atteint via un chemin depuis n'importe quel autre (connexité), et est celui ayant la plus petite somme de poids parmi tous les graphes qui vérifient cette connexité. Un jeu de données basique qui servira dans toute cette section ainsi que l'arbre couvrant de poids minimum associé sont illustrés dans la figure 1.7.

Ensuite, comme nous l'avons énoncé plus haut, les arrêtes de l'arbre sont supprimées une à une en commençant par les poids les plus faibles. Si on considère les sous graphes $G_{m_{pts},\epsilon}$ obtenus en supprimant graduellement les arêtes du graphe G de poids supérieur ou égal à ϵ, alors chacun de ces sous-graphes donne une solution



(A) Jeu de données exemple



(B) Arbre couvrant de poids minimum

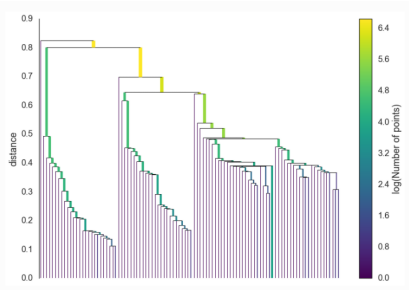
FIGURE 1.7 – Jeu de données exemple et arbre couvrant de poids minimum associé.

de DBSCAN qui serait obtenue avec le même ϵ. Ainsi, en éliminant itérativement les arrêtes de poids le plus élevé au plus faible on obtient toutes les solutions possibles de DBSCAN pour $\epsilon \in]0, +\infty[$.

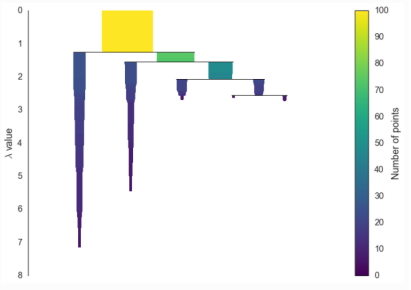
L'évolution des partitions induite par la suppression progressive des arêtes est naturellement résumée en un dendrogramme dont on peut voir un exemple dans la figure 1.8 où les couleurs renseignent sur le nombre de points dans chaque partition.

De plus cet arbre est étendu artificiellement en ajoutant une boucle sur chacun des sommets (invisibles sur l'illustration). Une boucle sur un sommet est une arête partant et finissant sur ce même sommet. Cette extension astucieuse permet de savoir quand un sommet n'est plus un objet cœur et doit être considéré comme du bruit. En effet, lorsque sa dernière arête a disparu cela veut dire que le niveau hiérarchique est égal à sa distance-cœur et donc à partir de cette distance ce nœud n'est plus objet cœur.

Plus précisément, la construction du dendrogramme passe par les étapes suivantes. Comme on l'avait avancé précédemment, cet arbre couvrant de poids minimum étendu est petit à petit dénudé de ses arêtes, en éliminant les arêtes de poids le plus élevé en premier. Chaque itération correspond un poids d'arête et toutes les arêtes partageant



(A) Dendrogramme avec tous les niveaux



(B) Dendrogramme simplifié.

FIGURE 1.8 – Hiérarchie obtenue par suppression successive des arrêtes du graphe. La première image représente le dendrogramme brut qui contient tous les niveaux alors que le second n'affiche que les niveaux pertinents.

ce même poids sont supprimées du graphe de manière simultanée. Le poids d'arête considéré va définir un niveau du dendrogramme, le niveau hiérarchique auquel on se situe. La suppression d'arêtes pourra entrainer l'apparition d'une ou plusieurs composantes connexes qui définissent donc chacun une nouvelle partition.

Certains sommets verront leur boucle disparaitre, auquel cas ils sont, à partir du niveau hiérarchique courant et pour tous les suivants, considérés comme du bruit. Il est important de préciser que les nouveaux clusters ne sont valables que pour le niveau hiérarchique courant. Rien n'indique pour l'instant si la duplication des clusters est souhaitable ou pas, il ne s'agit que d'exploration de différentes partitions, chacune associée à un niveau hiérarchique différent.

Le dendrogramme brut obtenu de cette manière peut être relativement peu lisible, avec énormément de niveaux hiérarchiques apparents. La simplification de la hiérarchie proposée est telle que seuls sont visibles les niveaux ayant induit une apparition d'une nouvelle composante connexe significative ou une disparition totale d'une composante connexe. Ces deux graphes, brut et simplifiés sont illustrés dans la figure 1.8.

Dans le dendrogramme simplifié, les niveaux hiérarchiques intermédiaires ayant juste identifié du bruit ne sont pas représentés. On considère comme du bruit toutes les nouvelles composantes connexes qui sont trop petites, qui ne contiennent pas assez de sommets et on les appelle tout simplement "faux clusters". Pour cela on introduit un nouveau paramètre m_{clsize} et on qualifie une nouvelle composante connexe de faux cluster si elle possède moins de m_{clsize} sommets.

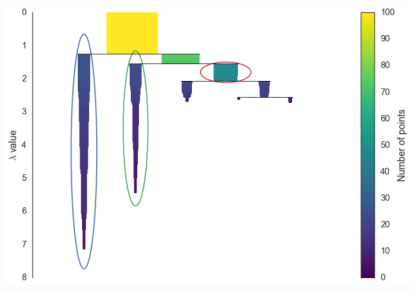
Lorsqu'un cluster C se retrouve séparé en plusieurs composantes connexes on peut se retrouver dans trois cas. Premier cas, toutes les sous-composantes connexes engendrées sont de faux clusters auquel cas le cluster initial a disparu au niveau hiérarchique courant. Second cas, au moins deux des sous composantes connexes sont significatives et donc plusieurs nouveaux clusters émergent du cluster C. Le troisième cas intervient quand un cluster se retrouve décomposé en un certain nombre de composantes connexes mais qu'une seule d'entre elles est une vraie partition, les autres étant trop petites.

Dans les deux premiers cas, ces évènements sont importants pour décrire la hiérarchie et sont donc renseignés avec leur niveau de hiérarchie associé dans le dendrogramme. En revanche dans le troisième cas, lorsqu'une seule des composantes est significative, rien de particulier n'est à déplorer : le cluster C n'a qu'été réduit mais n'a pas disparu ni donné naissance à d'autres sous-clusters. Il n'est donc pas représenté dans le dendrogramme.

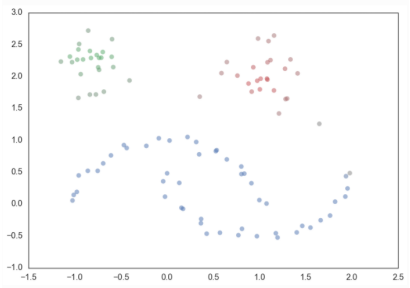
Une fois la hiérarchie rendue plus accessible, on est intéressé par l'extraction de clusters de cette hiérarchie. Cela revient à déterminer quels sont les séparations pertinentes et lesquelles ne sont pas souhaitables. Le problème est difficile et il n'existe que des heuristiques pour y parvenir.

Ici on approche le problème à travers la maximisation de la stabilité générale du partitionnement. On introduit la définition de stabilité d'un cluster qui est très semblable et basée sur la notion d'excès de masse suivante :

$$E(\mathbf{C_i}) = \int_{x \in \mathbf{C}_i} (f(x) - \lambda_{min}(\mathbf{C}_i))dx$$



(A) Dendrogramme simplifié dont les clusters extraits via optimisation de la stabilité générale sont entourés



(B) Illustration du clustering associé

FIGURE 1.9 – Résultats de l'extraction des clusters de la hiérarchie. Les trois partitions retenues sont entourées dans le dendrogramme et les couleurs correspondent aux partitions dans la deuxième image.

où λ_{min} est le niveau auquel le cluster $\mathbf{C}_i$ apparait et f est la densité.

Le problème avec la mesure d'excès de masse est que son évolution est monotone au sein de n'importe quelle branche de l'arbre hiérarchique. Pour remédier à cette impasse, une mesure d'excès de masse relatif est introduite.

$$E_R(\mathbf{C_i}) = \int_{x \in \mathbf{C}_i} (\lambda_{max}(x, \mathbf{C}_i) - \lambda_{min}(\mathbf{C}_i))dx$$

où $\lambda_{max}(x, \mathbf{C}_i) = \min(f(x), \lambda_{max}(\mathbf{C}_i))$, et $\lambda_{max}(\mathbf{C}_i)$ est le niveau de densité auquel $\mathbf{C}_i$ est séparé en sous clusters ou disparait.

La stabilité d'un cluster est enfin définie en discrétisant la formule d'excès de masse relatif précédente par :

$$S(\mathbf{C_i}) = \Sigma_{x_j \in \mathbf{C}_i}(\lambda_{max}(x_j, \mathbf{C}_i) - \lambda_{min}(\mathbf{C}_i)) = \Sigma_{x_j \in \mathbf{C}_i}(\frac{1}{\epsilon_{min}(x_j, \mathbf{C}_i)} - \frac{1}{\epsilon_{max}(\mathbf{C}_i))}$$

Comme nous l'avons émis plus haut, on cherche à extraire la partition la plus

pertinente parmi toutes celles possible au sein du dendrogramme. Le choix du partitionnement ne se résous pas naïvement au choix brutal d'un niveau dans la hierarchie, mais d'une selection plus fine, faite au niveau de chaque sous branche. Dans le cas de HDBSCAN on est quête de maximisation de la stabilité globale, de la somme des stabilités de chaque cluster activés en respectant certaines contraintes. Cette maximisation est entreprise à travers le problème d'optimisation binaire suivant :

$$\max_{\{\delta_2,...,\delta_K\}} J = \sum_{i=2}^{K} \delta_i S(\mathbf{C_i})$$
$$\text{subject to} \begin{cases} \delta_i \in \{0,1\}, i = 2,...,K. \\ \sum_{j \in \mathbf{I}_h} \delta_j = 1, \forall h \in \mathbf{L}. \end{cases}$$

où $\delta_i \in \{0,1\}, (i = 2,...,k)$ est la variable booléenne qui indique si le cluster $\mathbf{C}_i$ est présent dans la partition ($\delta_i \leftarrow 1$) ou non ($\delta_i \leftarrow 0$). $\mathbf{L}$ est l'ensemble d'indices désignant les cluster "feuilles" de l'arbre, c'est à dire ceux qui sont actifs (conservé dans la partition courante) et donc empêchent l'activation des cluster qui découleraient de ces cluster "feuilles". Les clusters qui découlent du cluster $\mathbf{C}_h$ dans la hiérarchie sont justement désignés à travers les indices $\mathbf{I}_h$ qui inclus également h. Ainsi, les contraintes de ce problème traduisent le fait que tous les sous clusters issus d'un cluster $\mathbf{C}_i$ (mais aussi tous les clusters ascendants, "plus haut" dans l'arbre) ne doivent pas être présents dans la partition dès que $\mathbf{C}_i$ est actif.

Pour ce qui est de la résolution, le mécanisme est relativement simple. La solution initiale non-réalisable est celle où tous les clusters sont présents dans la partition. On se place ensuite au plus bas possible dans l'arbre et on évalue chacun des nœuds $\mathbf{C}_i$ de ce niveau de la manière suivante. On compare $S(\mathbf{C}_i)$ la stabilité de $\mathbf{C}_i$ avec la somme des stabilités des sous clusters directement engendrés. Si $\mathbf{C}_i$ est moins stable que la stabilité cumulée de ses filles activées alors il se retrouve désactivé ($\delta_i \leftarrow 0$) et sa stabilité est remplacée par la somme de la stabilité de ses filles. Il servira par la suite de représentant pour son/ses sous clusters qui ont été activés à une itération antérieure. Si au contraire il est plus stable que ses filles, alors toutes ses filles jusqu'au plus bas de l'arbre sont désactivées et sa stabilité reste inchangée. Une fois que tous les nœuds du niveau courant on été traités, on réitère au niveau supérieur en se basant sur les nouvelles stabilités qui viennent d'être mises à jour. On monte ainsi

Algorithm 1. HDBSCAN main steps

1. Compute the core distance w.r.t. m_{pts} for all data objects in $\mathbf{X}$.
2. Compute an MST of $G_{m_{pts}}$, the Mutual Reachability Graph.
3. Extend the MST to obtain MST_{ext}, by adding for each vertex a "self edge" with the core distance of the corresponding object as weight.
4. Extract the HDBSCAN hierarchy as a dendrogram from MST_{ext}:
   - 4.1 For the root of the tree assign all objects the same label (single "cluster").
   - 4.2 Iteratively remove all edges from MST_{ext} in decreasing order of weights (in case of ties, edges must be removed simultaneously):
     - 4.2.1 Before each removal, set the dendrogram scale value of the current hierarchical level as the weight of the edge(s) to be removed.
     - 4.2.2 After each removal, assign labels to the connected component(s) that contain(s) the end vertex(-ices) of the removed edge(s), to obtain the next hierarchical level: assign a new cluster label to a component if it still has at least one edge, else assign it a null label ("noise").

Algorithm 2. HDBSCAN step 4.2.2 with (optional) parameter $m_{clSize} \geq 1$

4.2.2 After each removal (to obtain the next hierarchical level), process one at a time each cluster that contained the edge(s) just removed, by relabeling its resulting connected subcomponent(s):

Label *spurious* subcomponents as noise by assigning them the null label. If all subcomponents of a cluster are *spurious*, then the **cluster has disappeared**.
Else, if a single subcomponent of a cluster is *not spurious*, keep its original cluster label (**cluster has just shrunk**).
Else, if two or more subcomponents of a cluster are *not spurious*, assign new cluster labels to each of them (**"true" cluster split**).

FIGURE 1.10 – Algorithmique de HDBSCAN.

récursivement jusqu'à la racine qui est exclue, donnant lieu à l'algorithmique 1.10.

En plus de bénéficier de l'habilité de DBSCAN à gérer les clusters intriqués et non-concentriques, HDBSCAN possède peu de paramètres qui sont tous très robustes et fournit une hiérarchie et un partitionnement très pertinents. Sa complexité qui est originellement de $O(N^2)$ est en outre réduite à $O(Nlog(N))$ grâce à l'utilisation de la structure de données de type arbre-kd dont on parlera plus loin, dans la sous-section 2.1.1. Pour ces raisons il est l'un des algorithmes de clustering les plus connus et utilisés.

Chapitre 2

Outils auxiliaires

Nous définirons rapidement dans ce chapitre ce que sont les arbres binaires de recherches et leur extension que sont les arbres kd, ces derniers étant employés pour améliorer la complexité de HDBSCAN. Également, discuter des arbres hiérarchisés nous permettra de les mettre en parallèle avec la structure arborescente des résultats intermédiaires de Ballstering.

Une sous-section sera portée sur les hypersphères de dimension quelconque et plus précisément sur le calcul pratique du volume de superposition entre deux hypersphères, nécessaire au fonctionnement de Ballstering.

Enfin nous expliquerons l'usage de bases de splines cubiques utiles pour la reconstruction de signaux de nature fonctionnelle et verrons leur application à des données réelles issues du trafic aérien français. Cela nous permet de transformer les données brutes en données exploitables par Ballstering, exploitation qui sera discutée ultérieurement dans le chapitre 6.

2.1 Arbres de recherche et hypersphères

2.1.1 Arbres de recherche

Les arbres, binaires ou généraux, sont très souvent utilisés en informatique car les informations peuvent s'organiser de manière hiérarchisée et peuvent donc être représentées naturellement de manière arborescente, ce qui permet d'effectuer certaines opérations bien plus efficacement qu'avec d'autres modèles. Ils sont employés, entre autres, pour le stockage de données volumineuses ou pour le tri de données. En effet,

la structure d'indexation arborescente permet un accès rapide à une donnée particulière ou au voisinage d'un objet. Enfin, on retrouve cette théorie dans la formulation de nombreux problèmes d'optimisation combinatoire avec la collection d'algorithmes qui y sont associés [55].

Dans le cadre dans lequel on s'inscrit, car il existe plusieurs angles d'approches pour ce même objet, un arbre est une hiérarchie dont chaque élément est appelé nœud. Chacun de ces nœuds est situé à un certain niveau dans la hiérarchie (ou profondeur), et ses relations avec les autres nœuds sont règlementées. Au sein d'un arbre chaque nœud possède nombre de nœuds fils quelconque et un et un seul père sauf le premier nœud que l'on appelle la racine, élément premier de la hiérarchie et débutant l'arbre. Les nœuds ne possédant aucun fils sont appelés feuilles de l'arbre. Le nombre de niveaux qui sépare la racine de la feuille la plus basse dans l'arbre est appelé hauteur de l'arbre. Une illustration est donnée par la figure 2.1.

En théorie des graphes un arbre est un graphe simple (non-orienté) connexe et sans cycles, ce qui veut dire qu'entre deux nœuds de cet arbre il existe un et un seul chemin les reliant. De cet autre point de vue, l'aspect hiérarchique peut être inexistant : Il n'y a pas de pères ni de fils mais seulement des nœuds connectés deux à deux. On dit d'ailleurs dans ce cas que c'est un arbre non planté puisqu'il n'y a pas de sommet particulier qui joue le rôle de racine. Ce cadre bien plus large ne sera pas traité par la suite et nous parlerons uniquement d'arbres orientés, hiérarchisés.

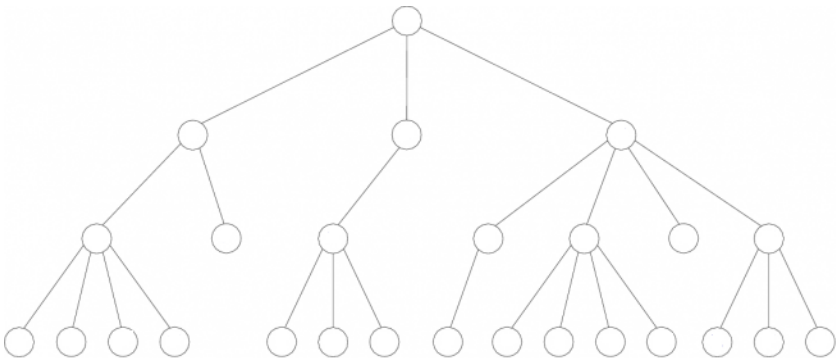


FIGURE 2.1 – Exemple d'arbre général : chaque cercle représente un nœud. S'il est hiérarchisé, les nœuds directement liés à un autre nœud de niveau directement supérieur (au dessus) sont appelés « fils ». Chaque nœud possède un nombre de fils quelconque.

Les arbres binaires

Tandis que les arbres généraux possèdent des nœuds pères ayant potentiellement une infinité de fils, les arbres binaires eux n'ont que des nœuds fils singularisés. Autrement dit chaque nœud possède uniquement deux fils : un fils gauche et un fils droit. De telles règles de construction donnent une structure particulière aux arbres binaires dont une illustration est donnée dans la figure 2.2.

Nous allons énumérer quelques notions de vocabulaire associé aux arbres binaires afin de fluidifier la compréhension de ce qui suit. Tout comme pour les notions de « père » et « fils » que nous avons définis plus haut, la terminologie employée pour décrire les éléments d'un arbre s'appuie essentiellement sur les liens familiaux, ce qui est relativement intuitif compte tenu de la succession de générations de nœuds et de leur type de relations :

— Les descendants d'un nœud sont les nœuds qui apparaissent dans ses sous-arbres.

— Un ancêtre d'un nœud est soit son père, soit un ancêtre de son père.

— Un frère d'un nœud est l'autre fils de son père.

Arbres binaires de recherche

À travers une structure arborescente munie d'une organisation particulière, les arbres binaires de recherche ont pour but d'indexer une base de données afin de

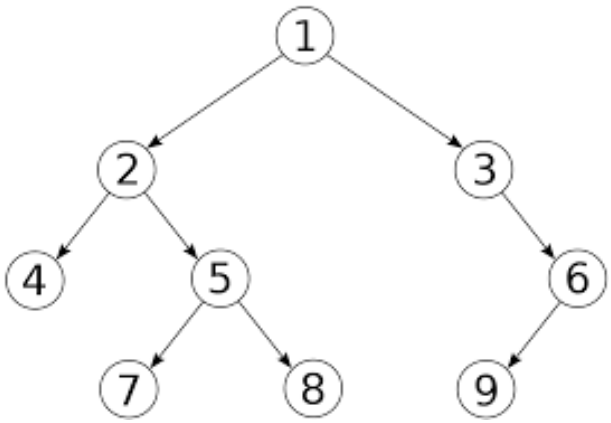


FIGURE 2.2 – Exemple d'arbre binaire : chaque boule représente un nœud. le nœud le plus haut, élément premier de la hiérarchie et débutant l'arbre, est appelé racine. Chaque nœud possède au maximum deux nœuds fils : le fils gauche et le fils droit.

pouvoir trouver rapidement, et de manière efficace, une observation particulière correspondant aux critères de recherche.

Ces arbres binaires particuliers héritent des propriétés des arbres binaires classiques auxquelles s'ajoutent de nouvelles caractéristiques. Tout d'abord, chaque nœud p contient une certaine valeur associée à une variable réelle unidimensionnelle (souvent des clés informatiques). Cette valeur que contient ce nœud p va permettre de représenter la séparation des observations qui est faite à partir de ce dernier : si un nœud p contient la valeur v alors il sépare les éléments étant plus grand que v des éléments plus petits que v en deux sous-ensembles distincts.

Ces deux sous ensembles engendrés par séparation, sont naturellement et implicitement représentés par les nœuds fils de p : Le fils de gauche de p contient les éléments de p qui ont une valeur inférieure ou égale à v et le fils de droite contient les observations qui ont une valeur supérieure ou égale à v.

Autrement dit tous les nœuds du sous-arbre de gauche d'un nœud de l'arbre ont une valeur inférieure ou égale à la sienne, et de manière analogue tous les nœuds du sous-arbre de droite d'un nœud de l'arbre ont une valeur supérieure ou égale à la sienne.

Puisque chaque nœud hérite des séparations effectuées récursivement par ses ancêtres, les feuilles de l'arbre, qui sont donc le résultat de multiples séparations, sont de petits sous-ensembles constitué d'observations similaires ou une seule observation.

Une fois l'arbre construit, on peut profiter de ses caractéristiques dans le but de retrouver efficacement une ou des observations correspondant à certains critères sur ses valeurs choisis au préalable. La méthode de recherche au sein d'un arbre binaire se décrit commodément de manière récursive.

Pour une valeur recherchée c, voici comment peut se découler une telle procédure de recherche. Le nœud courant, qui est modifié au cours des récursions, est en premier lieu initialisé au nœud racine. Si c est strictement supérieure à la valeur v du nœud courant p alors on doit chercher du côté droit dans l'arbre. Le nœud courant p devient le fils droit et on opère une récursion supplémentaires. On procède de manière analogue si c est strictement inférieur à v mais avec le fils de gauche cette fois-ci. Si v est égal à c alors la recherche est terminée puisqu'on arrive sur la donnée précise qui a

engendré le nœud p. Enfin, si p est une feuille de l'arbre la recherche est infructueuse, ce qui n'advient que lorsqu'on cherche une observation absente de la base de données.

Si l'arbre est équilibré (autant de nœuds à gauche qu'à droite de la racine) chaque récursion au cours de la recherche divise par deux le nombre de nœuds potentiels. Une telle recherche possède une complexité de l'ordre de $O(\log_2(n))$ où n est le nombre de nœuds. Ceci rend donc possible un accès très rapide à une donnée précise.

Arbres k-d

Les arbres multidimensionnels de recherche binaire, ou arbres k-d [4] sont une forme particulière des partitions binaires de l'espace. Cette famille de partitions a pour ambition d'étendre les arbres binaires de recherche classiques monodimensionnels vu dans la section précédente au cadre multidimensionnel. On cherche toujours à structurer les données en les partitionnant dans le but de pouvoir trouver rapidement le voisinage de chaque observation de la base de donnée considérée mais ce problème est bien plus complexe car les données à indexer auxquelles on s'intéresse sont désormais de dimension k. Ce sont des outils qui sont depuis longtemps prisés dans le

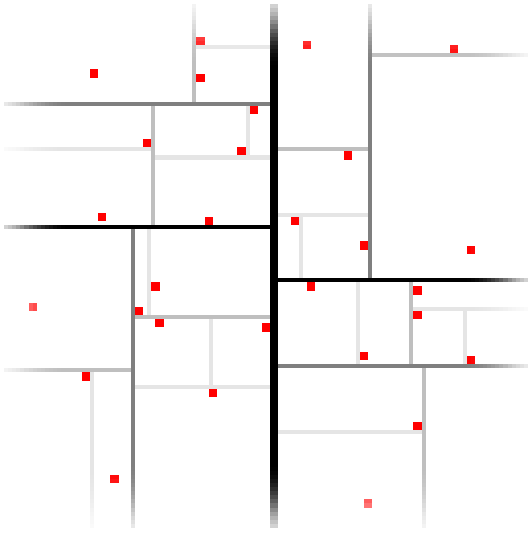


FIGURE 2.3 – Exemple de séparations récursives effectués par une méthode de type k-d sur un jeu de donnée bi-dimensionnel. les séparations sont représentées par les axes qui possèdent un certain niveau de gris qui correspond au moment de leur occurrence. un axe gris foncé a servi à discriminer antérieurement à un axe plus clair. Les données sont représentées en rouge.

champs du rendu graphique sur ordinateur [51].

Le principe fondamental des partitions binaires qui réside dans la séparation récursive de l'espace en deux parties est toujours présent, mais s'opère suivant un hyperplan choisi via des stratégies qui peuvent différer selon l'algorithme.

Tout comme dans la sous-section précédente ce sont des arbres binaires mais dans lesquels chaque nœud contient désormais à la fois un point x en dimension k et une information sur l'axe qui a servi à la coupe. Chaque nœud p (qui n'est pas une feuille) représente la division de l'espace en deux demi-espaces selon un hyperplan auquel x appartient. Comparativement aux partitions binaires de l'espace, les arbres k-d ont la particularité de subdiviser l'espace selon les axes canoniques. Plus précisément les plans séparateurs sont toujours perpendiculaires à un des axes du repère de l'espace.

Prenons en guise d'exemple un repère en trois dimensions dont les axes de la base canonique (u_1,u_2,u_3) sont respectivement associés aux variables x, y et z. Supposons qu'à été choisi le premier axe pour la séparation de l'ensemble E pour le nœud courant p. C'est alors un plan normal au premier axe u_1 qui sera choisi pour la subdivision, mais il en existe un infinité et on doit choisir un point particulier v de l'axe u_1 pour définir ce plan. Il existe plusieurs façons de choisir v mais le choix standard et le plus intuitif si on veut obtenir un arbre équilibré est de choisir le point médian m_1 selon cet axe u_1.

Ainsi, tous les points appartenant à E qui possèdent un attribut x de valeur inférieure au point médian m_1 sur le premier axe seront alors stockés dans le sous-arbre de gauche. De manière analogue, les points de coordonnée x supérieure à m_1 seront stockés dans le sous-arbre engendré par le fil de droite.

Les arbres k-d diffèrent entre eux en fonction des stratégies de choix de l'axe selon lequel couper selon le choix de l'endroit de coupe sur cet axe. Ils sont très utilisés en connivence d'algorithmes basés sur des requêtes de k plus proches voisins, ce qui est le cas de HDBSCAN par exemple. En effet, avec une structuration adéquate au problème, on peut obtenir les k plus proches voisins d'un point donné en entrée avec une complexité de de l'ordre de $O(nlog(n))$ au lieu de $O(n^2)$. Il en est de même pour

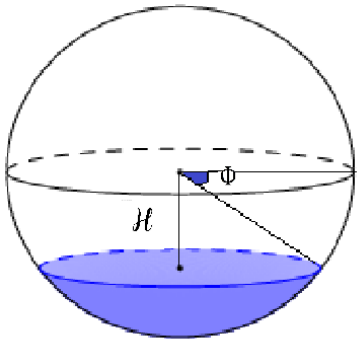


FIGURE 2.4 – Illustration de ce qu'est une calotte pour une 3-sphère qui est ici est représentée par la zone en bleu.

un grand nombre de méthodes basées sur le voisinage de points.

2.1.2 Volume de superposition entre deux n-sphères

Nous allons ici expliciter, en se basant sur les travaux de [31], comment calculer le volume que forme l'interaction entre deux hypersphères de dimension n, qu'on appelle de manière équivalente n-sphère. On dérivera ensuite la formule permettant le calcul du volume d'interaction entre deux n-sphères de même rayon r, laquelle nous intéresse tout particulièrement dans l'algorithme Ballstering.

Volume d'une calotte d'hypersphère

Dans [31] deux formules servant respectivement au calcul de l'aire et du volume d'une calotte hypersphérique sont données et défendues. On appelle calotte la portion de la n-sphère obtenue lorsqu'on la sectionne avec un hyperplan (en bleu dans la figure 2.4). Ces élégantes formules ont l'avantage d'être concises, aisément compréhensibles et pratiques car elles se reposent sur la fonction bêta régularisée qui est déjà disponible dans de nombreux packages. Elles sont obtenues en intégrant le rapport $\frac{\text{aire}}{\text{volume}}$ d'une $(n-1)$-hypersphère sur un arc de grand cercle dont l'intégration est faite selon les coordonnées hypersphériques. Notons que dans la partie qui suit nous ne nous intéressons qu'au volume de la calotte qui correspond à la plus petite partie de la sphère. Afin obtenir le volume de la grande partie, que j'appelle de manière abusive mais parlante l'« anti-calotte », il suffira de soustraire le volume de la calotte au volume total de l'hypersphère.

Définissons de prime abord le cadre dans lequel nous nous plaçons. Soit S^n une n-sphère de rayon r étant au sein d'un espace euclidien de dimension n :

$$S^n := \{x \in R^n t.q. ||x|| = r\}$$

Le volume de cette n-sphère V_n peut être calculé avec l'équation suivante :

$$V_n(r) = \frac{\pi^{n/2}}{\Gamma(\frac{n}{2}+1)} r^n,$$

où Γ est la fonction Gamma. Une calotte sur cette n-sphère peut être entièrement définie par un angle ϕ ou la longueur h comme indiqué sur la figure 2.4. Dans ce qui suit on se limite au cas où $\phi \leq \frac{\pi}{2}$ qui en fait recouvre tous les cas : si l'on effectue une simple rotation on peut revenir au cadre initial.

Le volume $V_n^c(r)$ de la calotte de la sphère de rayon r définie par l'angle ϕ peut être obtenu en intégrant le volume d'une $(n-1)$-sphère de rayon $rsin(\theta)$ selon l'élément de hauteur $drcos(\theta)$ ce qui donne :

$$V_n^{cal} = \int_{\theta=\phi}^{\theta=0} V_{n-1}(rsin(\theta))drcos(\theta)$$

Il est extrait de cette expression initiale, après un développement de quelques lignes, la formule concise faisant appel à la fonction bêta régularisée I et au volume d'une n-sphère :

$$V_n^{cal} = \frac{1}{2} V_n(r) I_{sin^2(\phi)}(\frac{n+1}{2}, \frac{1}{2}).$$

Notons que cette formule écrite en fonction de ϕ peut également s'écrire en fonction de h, la distance entre le centre de la n-sphère et l'hyperplan effectuant la coupe. Écrit selon cette hauteur h, la formule précédente deviendra :

$$
\begin{aligned}
V_n^{cal} &= \frac{1}{2} V_n(r) I_{sin^2(\phi)}(\frac{n+1}{2}, \frac{1}{2}). \\
&= \frac{1}{2} V_n(r) I_{1-cos^2(\phi)}(\frac{n+1}{2}, \frac{1}{2}). \\
V_n^{cal}(h, r) &= \frac{1}{2} V_n(r) I_{1-\frac{h^2}{r^2}}(\frac{n+1}{2}, \frac{1}{2}).
\end{aligned} \tag{2.1}
$$

Volume de superposition entre deux n-sphères quelconques

Ce qui nous importe est le volume de superposition entre deux hypersphères. L'intérêt de ce que nous venons d'expliciter, à savoir le volume d'une calotte hypersphérique, réside dans l'observation suivante.

Considérons deux n-sphères de rayon différents R et r espacées d'une distance d. On peut considérer, pour un déroulement plus simple et sans perte de généralité, que la première n-sphère est centrée à l'origine du repère et que la seconde est éloignée d'une distance d selon une seule direction du repère. On suppose en outre que $R + r < d$, sinon cela voudrait dire qu'elles ne s'intersectent pas donc leur volume de superposition est nul. De même on suppose que $d > |r - R|$ car une des n-sphères serait contenue dans l'autre donc le volume d'intersection serait le volume de la plus petite des deux hypersphères.

Comme on peut le voir dans la figure 2.5, une illustration en 2 dimensions de ce que nous venons d'énoncer, le volume d'interaction entre les deux sphères n'est autre

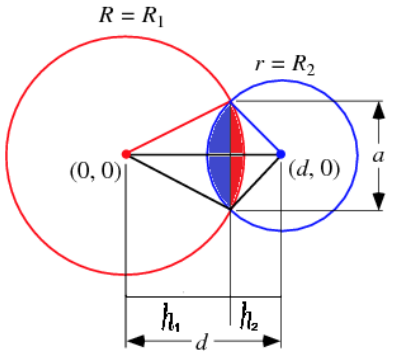


FIGURE 2.5 – Illustration du volume de superposition entre deux disques (2-sphères). On peut observer que ce volume n'est autre que la somme de deux volumes de calottes agglutinées de rayon différents (rouge et bleu).

que la somme de deux volumes de calottes agglutinées. Le volume de la première calotte visible en rouge sera alors calculé avec la formule 2.1 en considérant la n-sphère rouge de rayon R dont la hauteur de la coupe h (qui définit la calotte) est notée h_1. De manière analogue, la seconde calotte visible en bleu sera calculée à partir de la boule bleue de rayon r et avec la hauteur h égale à $h_2 = d - h_1$. Il ne restera ensuite qu'à sommer les deux volumes calculés pour obtenir le volume total de superposition.

Compte tenu de ce que nous avons exposé plus haut, il apparaît donc nécessaire d'obtenir les hauteurs h_1 et h_2 en fonction des données qui sont connues dans un problème réel à savoir les rayons R et r ainsi que la distance d.

Sur ce plan en $2d$, on peut remarquer que h_1 correspond à l'abscisse des points d'intersection entre les deux cercles. Puisque ce point d'intersection appartient aux deux sphères, et si on associe à la variable y la seconde dimension, la remarque précédente se traduit mathématiquement par :

$$\begin{aligned} h_1^2 + y^2 &= R^2 \\ (h_1 - d)^2 + y^2 &= r^2, \end{aligned}$$

ce qui implique l'équation du second degré suivante :

$$h_1^2 - 2dh_1 + d^2 - h_1^2 = r^2 - R^2.$$

En la résolvant on obtient l'expression de h_1 en fonction des données du problèmes, et on obtient immédiatement h_2 par simple jeu d'écriture :

$$\begin{aligned} h_1 &= \frac{d^2 - r^2 + R^2}{2d} \\ h_2 &= \frac{d^2 + r^2 - R^2}{2d}. \end{aligned}$$

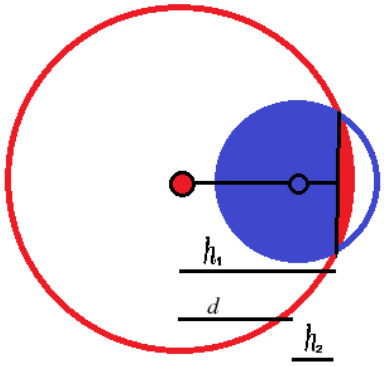


FIGURE 2.6 – Illustration du cas où h_2 est négatif. Dans ce cas il faut calculer le volume de l'anti-calotte par h_2.

Il se peut que h_1 ou h_2 soient négatifs. h_2 sera négatif quand $d^2 + r^2 - R^2 < 0$ donc lorsque les deux boules sont proches et la boule bleue est bien plus petite que la boule rouge, comme c'est le cas dans la figure 2.6. h_1 sera négatif dans le cas symétrique, c'est à dire lorsque la boule rouge est plus petite et les deux sphères toujours suffisamment proche l'une de l'autre.

La négativité des hauteurs h_1 et h_2 est importante puisqu'elle va déterminer si c'est le volume de la calotte ou de l'anti-calotte qu'il faut mesurer. En effet la figure 2.6, où h_2 est négatif, met en évidence que si l'une des hauteurs h est négative alors il faut compter le volume de l'anti-calotte de la boule correspondante.

Nous avons désormais toutes les informations en main pour pouvoir calculer le volume de superposition V_n^s des deux hypersphères qui est donc l'accumulation du volume de deux calottes (ou d'une calotte et d'une anti-calotte) de dimension n :

$$\begin{aligned} V_n^s =& \mathbf{1}_{[h_1 \geq 0]} V_n^{cal}(h_1, R) + \mathbf{1}_{[h_2 \geq 0]} V_n^{cal}(h_2, r) \\ &+ \mathbf{1}_{[h_1 < 0]} (V_n^{\overline{cal}}(-h_1, R)) + \mathbf{1}_{[h_2 < 0]} (V_n^{\overline{cal}}(-h_2, r)), \end{aligned} \tag{2.2}$$

où $V_n^{\overline{cal}}(h, r) = V_n(r) - V_n^{cal}(h, r)$ est le volume de l'anti-calotte.

Volume de superposition entre deux n-sphères de même rayon

Dans le cas particulier où les n-sphères sont toutes deux de rayon égal à r, h_1 et h_2 sont égaux et obligatoirement positifs ou nuls :

$$h_1 = h_2 = d/2$$

Le volume de superposition que nous notons $OL_r(d, n)$ prend alors une forme finale bien plus concise :

$$\begin{aligned} OL_r(d,n) =& 2V_n^{cal}(h,r) \\ =& r^p \frac{\pi^{n/2}}{\mathbf{\Gamma}(\frac{n}{2}+1)} \mathbf{I}_{1-(\frac{d}{2.r})^2}(\frac{n+1}{2}, \frac{1}{2}) \end{aligned} \tag{2.3}$$

où n est le nombre de dimensions de l'espace dans lequel vivent les hypersphères, d la distance les séparant, r est leur rayon, $\mathbf{\Gamma}$ est la fonction Gamma et $\mathbf{I}$ est la fonction bêta incomplète régularisée.

2.2 Pré-traitement de trajectoires d'avions

Dans cette section nous allons expliquer une manière de transformer la représentation de trajectoires d'avions, initialement complexes et difficilement comparables entre elles, afin d'en obtenir une concise et aisément exploitable par divers algorithmes de clustering.

Notre approche se résume par une ré-expression des données dans une base fonctionnelle particulière, les B-splines, projetant ainsi les trajectoires dans un même espace euclidien bien moins complexe, dans lequel les trajectoires sont directement comparables. Cette représentation est ensuite à nouveau travaillée et réduite via analyse en composante principale (ACP) ce qui nous permet de travailler au partitionnement sur des données en basse dimension.

L'exploitation de ces données transformées sera traitée dans la partie résultats.

2.2.1 Données fonctionnelles et description de données de vol

Se regroupent sous le terme *données fonctionnelles* toute série de relevés qui peut s'interpréter comme l'évolution continue d'une variable en fonction d'une ou plusieurs variables. On voit cet ensemble d'observations non pas comme indépendantes mais au contraire issues d'un même processus décrivant une courbe aléatoire.

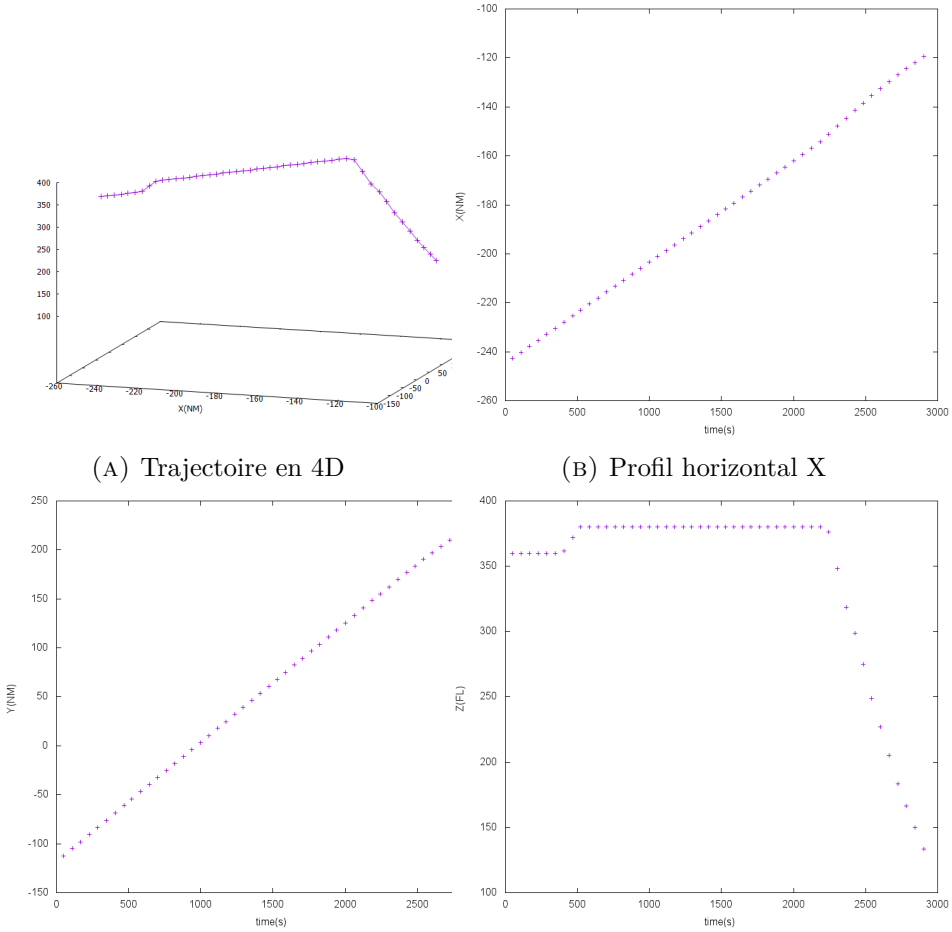
On ne peut pour l'instant (et très certainement jamais) mesurer de manière réellement continue l'évolution d'un processus aléatoire. On ne peut que se contenter de mesures discrètes, avec une certaine fréquence, du signal réel qui lui est théoriquement continu. On n'accède donc qu'à une série de relevés finie sous forme vectorielle $x := (x_1, x_2, ..., x_T)$ où T est le nombre de relevés, issus de l'observation ponctuelle de la courbe aléatoire X.

Très souvent elles dépendent du temps et décrivent l'évolution d'un processus comme la position d'un objet, l'évolution de la température, de la bourse ou encore de la croissance d'une plante par exemple. Elles peuvent aussi ne pas dépendre du temps même si c'est plus rare. En guise d'exemple on peut considérer l'analyse spectrométrique des matériaux (examen du spectre de lumière issu d'un matériau) qui décrit l'évolution de la réponse du matériau selon la longueur d'onde.

Des outils particuliers qui prennent en compte cette nature fonctionnelle existent pour les analyser et se dérivent sous plusieurs formes suivant la nature des fonctions observées (périodicité, discontinuité, nombre de dimensions...). Les outils d'analyse multivariée peuvent très bien servir à traiter ce type de données (en considérant x comme un vecteur aléatoire) mais il est plus sage d'utiliser ces outils d'analyse dès que la situation s'y prête.

Une première raison à cela est le nombre de relevés qui, selon la fréquence d'enregistrement, peut être démesurément grand et donc entraîner autant de variables aléatoires, pouvant dépasser le nombre d'observations. Cette situation pose beaucoup de problèmes pour la communauté statistique tant sur le plan théorique que sur le plan numérique.

Une deuxième raison est qu'en oubliant l'aspect fonctionnel des observations leur nature est perdue et on se prive donc de tous les outils permettant sa description, de



(A) Trajectoire en 4D

(B) Profil horizontal X

(C) Profil horizontal Y

(D) Profil vertical

FIGURE 2.7 – Illustration de l'évolution de la position dans l'espace d'une trajectoire d'avion dans le repère Euclidien tri-dimensionnel ainsi que ses trois profils représentant l'évolution au cours du temps de sa position dans chacune des dimensions séparément.

saisir ses particularités comme sa continuité, sa courbure ou encore sa périodicité.

Enfin, la réécriture des signaux bruts dans un espace fonctionnel défini par une base de fonction permet de hautement réduire la dimension du problème initial et de rendre les comparaisons entre les courbes observées plus pratiques.

Les données fonctionnelles que nous allons traiter sont des trajectoires d'avions dont la représentation brute classique consiste en une succession de coordonnées de type GPS (x, y, z, t) où les trois premières composantes sont spatiales (position dans l'espace) et la dernière, le temps de relevé, est temporelle (illustration figure 2.7).

Les trajectoires contiennent très souvent un nombre de relevés différents selon la

fréquence de relevé des coordonnées et ont une longueur et un temps de trajet variables. De plus le nombre de relevés par trajectoire peut être relativement important et dépasser le millier. Même échantillonné, les trajectoires brutes restent peu pratiques à analyser, que ce soit du point de vue de la taille des données induite que de la comparabilité inter-trajectoire.

Il y a donc deux nécessités : réduire la dimension du problème mais aussi inscrire les observations (trajectoires) dans un espace où elles peuvent être aisément comparées. On utilisera une base de fonctions particulièrement pratiques, les B-splines, qui permettent de répondre aux deux problématiques précédentes.

Tout au long de cette section nous allons illustrer les explications sur un jeu de données réelles issu de relevés de trafic aérien. Il s'agit de l'ensemble des vols ayant survolé la France métropolitaine sur une période de 4 mois (Août-Novembre 2016), engrangeant ainsi 877 705 trajectoires au total. Chaque trajectoire a été échantillonnée en amont et contient 100 coordonnées GPS environ, allant de 30 à 200 points suivant la trajectoire et dont la fréquence de relevé correspond à un enregistrement toutes les 40 secondes environ.

C'est un jeu de données relativement complexe puisqu'on ne se limite pas aux vols intérieurs au pays mais on considère l'ensemble du trafic mondial passant par la France. Un nombre important d'aéroports (la plupart étrangers) sont donc impliqués, induisant ainsi une grande variété de couloirs aériens empruntés et de trajectoires types.

On verra que grâce à une représentation fonctionnelle adaptée on peut réduire la représentation de ce trafic français à un espace euclidien en 4 dimension dans lequel chaque point représente une trajectoire, le tout en préservant la majeure partie de l'information initiale.

2.2.2 Interpolation via B-splines cubiques

La stratégie que nous utilisons pour simplifier cette représentation consiste en la réécriture de ces trajectoires selon une base fonctionnelle nommée B-spline, qui se fait par interpolation.

Les B-splines que nous utilisons ne s'appliquent qu'à l'évolution d'une variable réelle 1-D. Il existe des méthodes similaires pour traiter le cas multivarié [10] qui sembleraient plus adaptées à ce que nous voulons effectuer, mais c'est un problème encore ouvert et difficile. Afin de pouvoir l'appliquer à notre cas tri-dimensionnel nous traitons donc séparément l'évolution dans chaque dimension donnant lieu à trois courbes pour chaque trajectoire. Les deux courbes définies respectivement par l'évolution de X et Y en fonction du temps sont appelées profil horizontaux, et celle décrivant l'évolution de Z qui est appelée profil vertical. Un exemple de la décomposition d'une trajectoire en ses trois profils est donnée dans la figure 2.7.

Base de fonctions et base de splines

Commençons tout d'abord par expliciter la notion de base de fonctions. Une base de fonctions est une famille de fonctions $\{\phi_1, ..., \phi_K\}$ engendrant par combinaisons linéaires un espace de fonctions qui lui est propre. Plus précisément, toute fonction X engendrée par cette base sera exprimée comme combinaison linéaire de fonctions de cette base pondérée par des coefficients c_k associés aux fonctions $\phi_k(t)$:

$$X(t) = \sum_{k=1}^{K} c_k \phi_k(t) = \mathbf{c}'\phi, \tag{2.4}$$

où $\mathbf{c}$ est le vecteur réunissant tous les coefficients c_k et ϕ réunissant les fonctions de la base.

Pour un signal enregistré $x(t)$ dont on connaît les valeurs uniquement sur un ensemble de temps discret $\{t_1, ..., t_n\}$, on cherche à construire une fonction $\tilde{X}(t)$ qui correspond le mieux à x en jouant sur les coefficients c_k. En règle générale on détermine les coefficients idéaux grâce à la méthode des moindres carrés [29] qui minimise la somme des écarts au carré ce qui a entre autres la propriété de pénaliser particulièrement les gros écarts, même ponctuels :

$$\min_{c_1,..c_K} \sum_{i=1}^{n} (x(t_i) - \tilde{X}(t_i))^2 \tag{2.5}$$

Selon la nature des fonctions de la base et celle du signal enregistré, la reconstruction sera plus ou moins fidèle au signal d'origine. Il est donc d'importance de choisir précautionneusement la base en fonction de l'allure des données traitées. Par exemple, les séries de Fourier [41] dont les éléments de la base sont des sinus et cosinus sont très pratiques pour reconstruire des signaux périodiques.

Pour notre travail de reconstruction de trajectoires d'avions, nous avons opté pour l'usage de splines, des fonction polynomiales par morceaux. Elles forment une base plus flexible que la plupart, mais sont aussi plus difficiles à paramétrer.

Les splines sont définies par leur intervalle leur domaine de définition $[a, b]$, le degré des polynômes qui les constituent et d'un ensemble de nœuds. Mathématiquement, une courbe spline S est une fonction polynomiale par morceaux définie sur un intervalle $[t_0, t_k]$ divisé en k sous-intervalles $I_i[t_{i-1}, t_i]$ tels que les nœuds t_i vérifient $t_0 < t_1 < ... < t_k$.

Un polynôme différent P_i est défini sur chacun de ces intervalles $I_i = [t_{i-1}, t_i]$ de telle manière que la spline soit la somme de ces polynômes :

$$S(t) = \sum_{i=0}^{k} \mathbf{1}_{t \in I_i} P_i(t).$$

Il est possible de forcer une cohérence sur les jointures entre polynômes au niveau des nœuds. On peut vouloir par exemple que la fonction soit continue, ne fasse pas de sauts, c'est à dire que la valeur donnée par un polynôme à l'extrémité droite de son domaine coïncide avec la valeur donnée par le polynôme suivant à son extrémité gauche. Il peut être également souhaitable que la spline soit C^1. Pour cela, Puisque les polynômes sont C^∞, il suffit de la même manière que coïncident à chaque nœud les dérivées des deux polynômes concernés. De manière générale, pour que la spline soit C^n il faut que :

$$P_{i-1}^{(j)}(t_{i-1}) = P_i^{(j)}(ti-1), \forall i\{1, ..., k\}, \forall j \in \{0, ..., n\}$$

De nombreuses fonctions splines différentes existent donc en fonction des cohérences que l'on souhaite sur les dérivées aux nœuds ainsi qu'en fonction du degré des

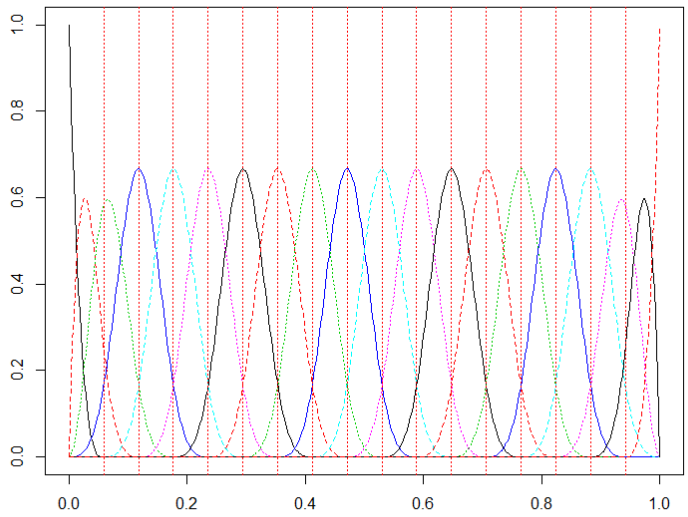


FIGURE 2.8 – Illustration d'une base de fonctions constituée de 20 splines. Chacune des splines est positive que de un à quatre intervalles contigus maximum. Elles commencent et terminent toutes à zéro (sauf la première et la dernière) et atteignent leur pic à un des nœuds t_i.

polynômes qui les constitue. Il existe donc également plusieurs manières de construire des bases de splines portant des noms différents comme les M-splines, les I-splines ou encore les B-splines. Ce sont ces dernières que nous utilisons et allons donc décrire succinctement.

Une base de B-spline est donc un ensemble de fonctions splines dont la particularité est que les splines qui la constituent sont élémentaires au sens qu'elles ne sont définies que par un seul polynôme sur un intervalle donné et sont nulles partout ailleurs. Considérez l'exemple illustré dans la figure 2.8 qui présente une base contenant 20 splines d'ordre 4 de différente couleurs. Comme on peut le voir, chacune des splines n'est positive que sur un seul intervalle plus ou moins grand, constitué de sous intervalles $[t_i, t_{i+1}]$ consécutifs (4 intervalles au maximum pour celles situées au milieu). Mis à part la première et la dernière fonction de la base, elles commencent et finissent toutes à la valeur zéro et atteignent leur pic à un certain nœud t_i.

Leur domaine d'influence individuel de taille limitée permet de n'impliquer à l'estimation de chaque point qu'un nombre restreint de splines de la base, et donc reconstruire des signaux avec précision sans entraîner une complexité démesurée.

Bien qu'il soit possible de jouer sur le placement des points t_i qui n'ont pas forcément à être répartis de manière homogène, nous nous contentons de cette base qui est amplement suffisante pour la reconstruction de nos profils. Pour plus de détails voir [45].

Interpolation

Une fois la base choisie, il convient d'interpoler pour exprimer de la manière la plus fidèle possible les signaux empiriques x dans la base choisie ϕ à travers les coefficients $\mathbf{c}$. On tachera donc de construire la fonction $\tilde{X} = \mathbf{c}'\phi$ en choisissant les coefficients $\mathbf{c}$ tels que $\tilde{X}$ soit le plus fidèle possible au signal théorique X sous-jacent au signal discret observé x. Le terme « fidèle » peut prendre un sens mathématique différent selon la technique qu'on utilise, mais le plus fréquent est de construire $\tilde{X}$ tel quel les écarts au carrés soient minimisés. Cette étape de minimisation est effectuée grâce à la méthode des moindres carrés qui permet d'obtenir les coefficients estimés $\hat{\mathbf{c}}$ grâce à un simple calcul matriciel :

$$\hat{\mathbf{c}} = (\phi'\phi)^{-1}\phi' x, \tag{2.6}$$

où ϕ est la matrice de dimensions $n \times K$ contenant les valeurs numériques $\phi_k(t_i)$.

$\hat{x}$, le vecteur des évaluations aux temps t_i du signal reconstruit $\hat{X}$, s'obtient alors avec la formule :

$$\hat{X} = \phi(\phi'\phi)^{-1}\phi' x, \tag{2.7}$$

Il se peut que le signal soit mal reconstruit pour plusieurs raisons dont le phénomène de Runge qui peut nous impacter dans le cadre de notre problème. Pour décrire très brièvement cet effet de Runge, nous dirons que c'est un phénomène d'oscillations artificielles et non-souhaitables sur la fonction reconstruite $\tilde{X}$ qui peut apparaître lorsque la base de spline est complexe et que le nombre d'évaluations est faible, trop faible pour obliger une certaine régularité dans la reconstruction. Un exemple extrême apparaît lorsqu'il existe un « trou temporel » dans les observations : en l'absence d'observations pour indiquer la valeur à laquelle il faut « coller », la reconstruction pourra se permettre de prendre quasiment n'importe quelle valeur dans cette zone.

Pour limiter les oscillations intempestives il convient d'être parcimonieux vis à vis de la quantité de splines formant la base. Une solution secondaire est de forcer la reconstruction à être lisse. Or une fonction qui oscille beaucoup aura des dérivées (d'ordre 1 ou supérieures) élevées et au contraire une fonction lisse sera caractérisée par des dérivées faibles. Ainsi, afin de limiter les oscillations superflues, il suffit d'ajouter une pénalité sur les dérivées lors de la reconstruction par les moindres carrés. Puisqu'il faut pénaliser les dérivées sur l'ensemble de la courbe, il est naturel d'utiliser comme pénalité les dérivées cumulées. Si l'on veut pénaliser les dérivées $m^{\text{ième}}$ d'une courbe X on se reposera donc sur la mesure PEN_m définie par l'équation suivante :

$$\text{PEN}_m = \int [D^m X(t)]^2 dt, \tag{2.8}$$

où D^m désigne l'opérateur différentiel d'ordre m. Finalement pour obtenir $\hat{c}$ l'estimation par moindre carrés pénalisés sur les dérivées d'ordre m on utilisera la formule :

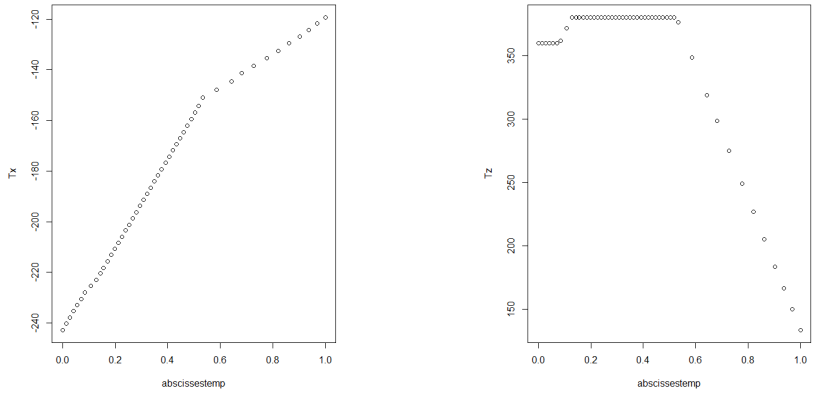
$$\hat{\mathbf{c}} = (\phi'\phi + \lambda \mathbf{R})^{-1} \phi' x, \tag{2.9}$$

où $\mathbf{R} = \int D^m \phi D^m \phi'$ et λ est un paramètre qui permet de moduler l'importance de la pénalité appliquée.

2.2.3 Application et reconstruction des trajectoires

Nous avons pour but de représenter les trajectoires présentées en début de section dans un espace engendré par une base B-spline. Or les trajectoires possèdent un temps de vol, une distance parcourue et donc un domaine de variation très différents, parfois disjoints. A moins de définir une base de grande envergure qui englobe tout le domaine de variation qui sera, soit constituée de trop nombreuses splines et donc engendrera une trop grande complexité et sera sujette au phénomène de Runge, soit trop peu précise pour reconstituer fidèlement les signaux. Il n'est donc en l'état impossible de représenter ces trajectoires dans un même base spline.

Le moyen que nous avons trouvé s'inspire de la théorie de l'abscisse curviligne pour projeter l'ensemble des trajectoires sur le même domaine de variation $[0, 1]$. Pour ce faire nous sacrifions la composante temporelle t de la trajectoire $x(t)$ pour



(A) Version curviligne d'un profil horizontal

(B) Version curviligne d'un profil horizontal

FIGURE 2.9 – Illustration de la nouvelle forme des deux profils exprimés en fonction de l'abscisse curviligne. Dans cette représentation, toutes les trajectoires varient en fonction de la même variable réelle évoluant sur le domaine $[0, 1]$.

penser en terme de distance parcourue, en suivant la courbe, depuis le point initial de celle-ci. Au lieu de se demander quelle est la position de l'objet à un instant t, on se demandera à quelle position l'objet se situe lorsqu'on a parcouru un certain pourcentage de la courbe.

Pour obtenir une telle courbe pour la trajectoire x il faut d'abord calculer la somme des distances cumulées partielles pour chacun des relevé x_k

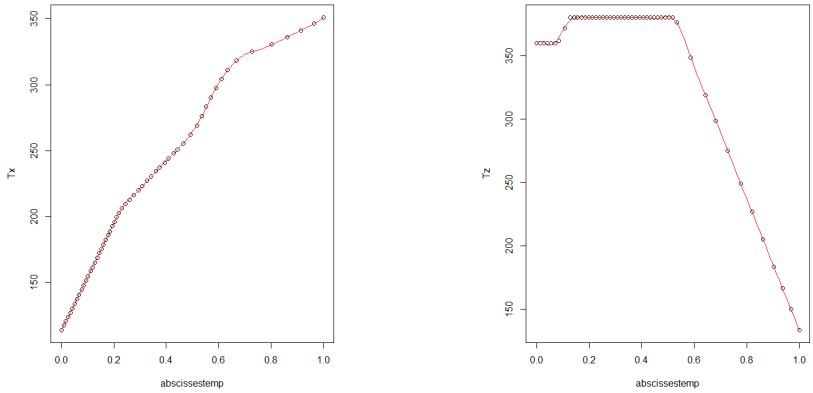
$$l_k = \sum_{i=2}^{k} (dist(x_i - x_{i-1})). \tag{2.10}$$

On obtient alors la trajectoire normalisée en associant à chaque relevé x_k sa somme cumulée partielle l_k divisée par la somme totale :

$$\frac{l_k}{\sum_{i=2}^{m} (dist(x_i - x_{i-1}))}, \tag{2.11}$$

où m est le nombre de relevés de la trajectoire observée x.

Avec cette représentation toutes les trajectoires se retrouvent à varier dans le même domaine $[0, 1]$ et ce sans être dénaturées (voir figure 2.9), et peuvent donc désormais être interpolées de manière pratique via une base B-spline bien choisie. Il se trouve que l'exemple de base splines d'ordre 4 contenant 20 fonctions polynomiales



(A) Reconstruction du profil horizontal d'une trajectoire

(B) Reconstruction d'un profil vertical d'une trajectoire

FIGURE 2.10 – Exemple de reconstruction d'un profil horizontal (A) et d'un profil vertical (B). La trajectoire initiale (abscisses curvilignes) est décrite par les points noirs et sa reconstruction par la courbe rouge.

illustré en figure 2.8 est précisément celle qu'on utilise pour reconstruire nos trajectoires (pour tous les profils). En usant des moindres carrés pénalisés sur la dérivée d'ordre 4, on obtient les fidèles reconstructions illustrées en figure 2.10.

À ce stade nous avons donc réussi à transformer des trajectoires contenant plus d'une centaines de coordonnées tri-dimensionnelles (en nombre variable suivant la trajectoire), en des trajectoires ne contenant que 60 coefficients, puisque 20 par profil. En plus de réduire la dimension du problème, ces trajectoires deviennent comparables entre elles dans cet espace Euclidien à 60 dimensions.

L'information est donc condensée mais nous considérons que ça ne l'est pas assez. Nous appliquons donc à cette matrice $C \in K \times 60$ une analyse en composantes principales (ACP [1]) qui permet de condenser au maximum la variabilité de l'échantillon sur les quelques premiers axes principaux. Appliquée à notre matrice de coefficients C, on obtient l'éboulis des valeurs propres (figure 2.11) qui indique qu'on arrive à condenser 91% de l'information sur les quatre premiers axes principaux. Ce nouvel espace à 4 dimensions défini par les quatre premiers axes principaux sera alors suffisant pour décrire le jeu de données.

Pour résumer, nous avons projeté toutes les trajectoires dans le même intervalle afin de pouvoir les exprimer, à travers des vecteurs de coefficients dans une base

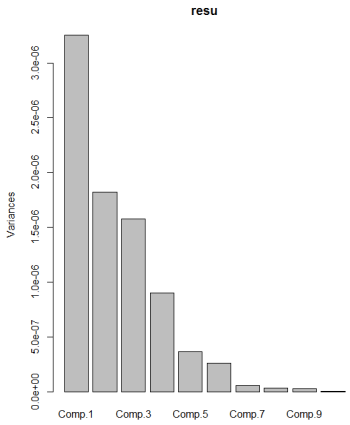


FIGURE 2.11 – Éboulis des valeurs propres (10 premières) de la matrice de variance-covariance des coefficients dans la base spline. Les quatre premières composantes principales concentrent plus de 91% de la variabilité totale.

de splines pour enfin réduire drastiquement le nombre de coefficients nécessaire à la caractérisation de chaque trajectoires grâce à l'analyse en composante principale. Nos trajectoires, qui se retrouvent individuellement représentées par quatre coefficients seulement, appartiennent donc après tout ce travail à un espace euclidien $4D$. Des illustrations $3d$ supplémentaires sont fournies en annexe dans la figure A.1. Nous commenterons plus en détail ce jeu de données obtenu dans la section résultats.

Chapitre 3

Procedure CM itérée à fenêtre décroissante (ICMDW)

Comme on a pu le voir dans le chapitre 2, FDPC présente des difficultés à traiter un grand nombre d'observations et est très sensible au paramètre d_c qui est statique et dont le choix n'est pas trivial. Nous allons au cours de ce chapitre présenter les premières modifications que nous avons apporté pour pallier à ces deux problèmes.

On se concentre donc ici sur ICMDW, une procédure qui permet d'obtenir les comptes de densité ρ_i (au sens de FDPC) en une complexité nettement moindre. D'une part les densités ne sont plus évaluées pour tous les points mais sur un sous-ensemble pertinent seulement, d'autre part chaque compte de densité s'obtient avec un nombre de calculs réduit. De plus ICMDW évalue les ρ_i selon un rayon qui est désormais déterminé automatiquement et dynamique, c'est à dire adapté à la densité locale.

Dans une première partie nous expliciterons de manière informelle le principe général de ICMDW pour ensuite le décrire en détail dans les deux sections suivantes. Tout au long de ce chapitre, le fonctionnement de l'algorithme sera illustré sur le jeu de données Agrégation dont on peut voir la répartition sur la figure 3.1. C'est un jeu de données disponible publiquement sur internet [17] et qui est régulièrement utilisé pour mettre en évidence le bon fonctionnement de divers algorithmes de clustering dont notamment FDPC.

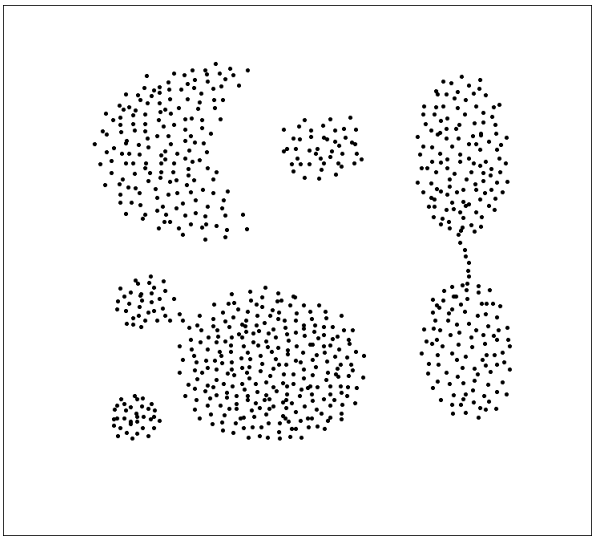


FIGURE 3.1 – Jeu de données Aggregation

3.1 Principe général de ICMDW

ICMDW consiste en l'itération successive d'une sous-procédure que nous appelons Cover Map (CM). Pour un rayon donné d_c, la sous-procédure CM couvre tout l'échantillon de boules ayant un rayon de longueur égale à d_c. Au terme de cette sous-procédure, tout point de l'échantillon est contenu dans au moins une boule. Cependant tout point ne définit pas obligatoirement un centre de boule. Cette procédure (CM) est itérée en réduisant à chaque fois la taille du rayon d_c mais aussi en interdisant la construction de trop petites boules dans les zones peu denses. Comme nous allons le voir, les informations de proximité fournies par l'itération précédente permettent de réduire considérablement le nombre d'opérations nécessaires à son fonctionnement Il en résulte en outre des informations de proximité entre les boules de génération contiguë (formant ainsi un arbre) qui seront utiles pour la phase de partitionnement.

3.2 Procédure pre-Cover Map (pre-CM)

La sous procédure que nous allons décrire ici n'est pas utilisée telle quelle dans l'algorithme complet. Cependant la présenter ici en premier lieu permettra d'appréhender progressivement son fonctionnement et ainsi de faciliter la compréhension de ICMDW qui l'itère.

Pour un rayon donné d_c, qui à cette échelle est statique, la procédure pré-CM couvre entièrement le domaine des observations avec une collection de boules de rayon égal à d_c. Les boules ne seront placées qu'aux endroits où des données sont présentes et, à la fin de cette procédure, tous les points de l'échantillon appartiennent à au moins une de ces boules.

Pour créer cette collection de boules, chaque observation x_i du jeu de données est tirée une fois. À chaque fois que l'on tire un élément x_i, on vérifie s'il appartient à une ou plusieurs boule et, dans le cas contraire, on crée une nouvelle boule qui hérite d'une partie de la densité des boules avec lesquelles elle est en superposition.

Au départ il n'existe aucune boule donc la première observation définit automatiquement la première boule $B_1(x_1, d_c)$ centrée sur x_1, de rayon d_c et son compte de densité $\rho(B_1(x_1, d_c))$ est initialisé à 1. Une nouvelle observation x_2 est tirée et on vérifie si x_2 appartient à B_1. Si c'est le cas alors B_1 voit son compte de densité augmenter de 1. Dans le cas contraire, une nouvelle boule $B_2(x_2, d_c)$ de centre x_2 et de rayon d_c est créée. Son compte de densité est initialisé à 1 plus une part de la densité de B_1 si cette dernière et B_2 se superposent, c'est à dire si $d(x_1, x_2) < 2d_c$. On continue donc de manière analogue jusqu'à ce que toute observation ait été traitée.

La part de densité héritée des boules voisines dépend de la proportion suivante :

$$\frac{\text{volume de superposition}}{\text{volume total}} .$$

Puisque les boules que nous considérons peuvent être multidimensionnelles, des hypersphères, et possèdent toutes le même rayon d_c nous allons nous reposer sur la formule 2.3 que nous avons établi dans le chapitre précédent.

Mathématiquement et selon les notation mises en place dans ce chapitre, le compte de densité initial de la nouvelle boule B_k peut être écrit :

$$\rho(B(x_k, d_c)) = 1 + \sum_j \frac{OL_{d_c}(d(B(x_k, d_c), B_j), p)}{V_{d_c}^p} \rho(B_j), \tag{3.1}$$

où $V_{d_c}^p$ est le volume d'une p-sphère de rayon d_c et $OL_{d_c}(d, p)$ est la fonction

qui permet de calculer le volume de superposition (ou d'intersection) entre deux hypersphères de dimension p, de rayon d_c toutes les deux et séparées d'une distance d :

$$OL_{d_c}(d,p) = {d_c}^p \frac{\pi^{p/2}}{\mathbf{\Gamma}(\frac{p}{2}+1)} \mathbf{I}_{1-(\frac{d}{2.d_c})^2}\left(\frac{p+1}{2}, \frac{1}{2}\right),$$

où $\mathbf{\Gamma}$ est la fonction Gamma et $\mathbf{I}$ est la fonction bêta incomplète régularisée. Cette dernière formule a été élaborée en détails dans la dernière section du chapitre 2 (eq 2.3).

Il peut paraître, et à juste titre, que l'initialisation de la densité à la création de chaque boule induise une quantité de calculs trop importante. Heureusement, nous allons voir que nous n'avons pas besoin de re-calculer à chaque fois les valeurs de OL.

Tout d'abord, il faut remarquer que :

$$OL_r(d.r,p) = r^p.OL_1(d,p). \tag{3.2}$$

De plus, si on considère d'une part le problème à deux hypersphères B_1 et B_2 toutes deux de rayon r et séparées d'une distance d, et qu'on le met en lien avec le problème remis à l'échelle B_1' et B_2' de rayon 1 et séparées de $d' = \frac{d}{r}$, alors le volume de superposition entre B_1 et B_2 peut être exprimé en fonction du second problème (en usant de l'équation 3.2) :

$$\begin{aligned} OL_r(d,p) &= OL_r(\frac{d}{r}.r,p) \\ &= OL_r(d'.r,p) \\ &= r^p OL_1(d',p). \end{aligned}$$

L'équation 3.1 devient alors :

$$rho(B(x_k,d_c)) = 1 + \sum_j \frac{OL_1(\frac{d(B(x_k,d_c),B_j)}{d_c},p)}{V_{d_c}^p} \times \rho(B_j), \tag{3.3}$$

où la distance remise à l'échelle $\frac{d(B(x_k,d_c),B_j)}{d_c} \in [0,2]$ est le seul paramètre de OL et ce dernier ne peut varier que sur l'intervalle $[0,2]$ car en dehors de cet intervalle il n'y a pas de superposition.

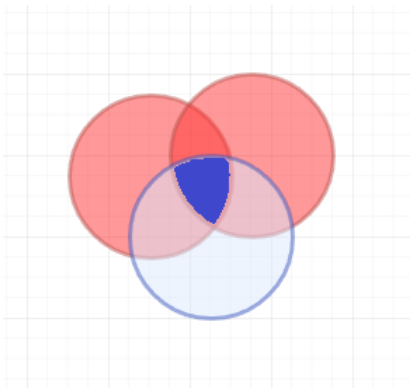


FIGURE 3.2 – Illustration de la zone de superposition multiple. L'aire contenue dans la zone bleue est comptée deux fois lorsqu'on somme les aires de superposition de manière indépendante.

Ce que nous venons d'exposer signifie qu'on peut toujours se ramener à un problème, à un facteur près, de superposition de sphères de rayon unité. Ainsi, au lieu de calculer les proportions d'interaction à chaque création de boule, on peut se contenter de calculer au préalable dans une table les valeurs de $OL_1(d, p)$ pour plusieurs dimensions p et pour plusieurs valeurs de d couvrant finement et de manière homogène le segment $[0, 2]$.

Ensuite, pour chaque triplet (d, p, r) donné, afin de retrouver le volume d'interaction entre deux boules il ne reste plus qu'à référer à la table, trouver d' le réel le plus proche de d existant dans la table pour ensuite chercher la valeur $OL_1(\frac{d'}{r}, p)$ correspondante. On somme ensuite les proportions et on finit en le divisant par $V_{d_c}^p$ qui correspond en fait à $OL_{d_c}(0, p)$, donc le premier élément de la table correspondant à la dimension p.

Dans l'implémentation courante, nous avons calculé et enregistré dans une table les valeurs des OL sur une grille d'un millier de points homogènement répartis sur $[0, 2]$ et ce, pour les 13 premières dimensions spatiales.

Nous sommes conscients que dans le cas de superpositions multiples certaines parts de densité peuvent être comptées plusieurs fois (zone bleue dans la figure 3.2), induisant une sur-estimation de la densité initiale.

Pour compenser ce phénomène qu'on est pour l'instant obligés d'affronter pour des raisons de performances, nous avons décidé de limiter le compte de densité de base à la moyenne de la densité des boules avec qui la nouvelle est en superposition

si jamais cette dernière devait être initialisée avec un compte de densité dépassant cette moyenne. Autrement dit, à la création de la $k^{\text{ème}}$ boule, on calcule la moyenne μ_k des densités des voisins avec qui la nouvelle boule se superpose puis on initialise le compte de densité ρ_k avec la formule 3.3. Si $\rho_k > \mu$ alors ρ_k prend la valeur μ_k à la place.

Le fonctionnement de la procédure pré-CM est résumé dans l'algorithmique 2.

Algorithm 2 Pre-Cover Map

procedure $CM(d_c)$
  $S \leftarrow B_1(x_1, d_c)$ ▷ Create the first ball
  $\rho_1(B_1(x_1, d_c)) \leftarrow 1$
  for all $x_i, i > 1$ **do**
    $d[j] \leftarrow \text{distance}(x_i, S[j]), \forall j$
    $J \leftarrow \{j | d[j] < d_c\}$
    if $J \neq \emptyset$ **then**
      $\forall j \in J, \rho_j(B_j) \leftarrow \rho_j(B_j) + 1$
    else
      $S \leftarrow S \cup B_k(x_i, d_c)$ ▷ Create new ball
      $\rho_{\min} \leftarrow \min(\mu, 1 + \sum_m \frac{OL_1(\frac{d(B_k(x_i,d_c),B_m)}{d_c}, p)}{OL_1(0,p)} . \rho(B_m))$
      $\rho_k(B_k(x_i, d_c)) \leftarrow \rho_{\min}$
    end if
  end for
end procedure

Grâce à cette procédure, on obtient un ensemble de boules, en quantité bien inférieure au nombre d'observations initial, munies de leur compte de densité local ρ_i couvrant entièrement et parcimonieusement le domaine, le tout de manière relativement homogène. En sacrifiant un peu de précision, on obtient l'équivalent des comptes de densités de FDPC mais pour une complexité fortement amoindrie. Les ensembles de boules que fournit cette méthode sur le jeu de données Agrégation, pour quatre rayons d_c différents, sont illustrés dans la figure 3.3.

Cependant, cette méthode n'est pas encore satisfaisante et ce pour plusieurs raisons. Tout d'abord, tout comme pour FDPC, d_c doit être choisi manuellement ce qui, comme nous l'avons déjà dit, est délicat. De son aspect statique pose problème : même Si d_c est bien adapté pour estimer et décrire une zone dense, il ne pourra pas l'être pour une zone moins dense, et inversement.

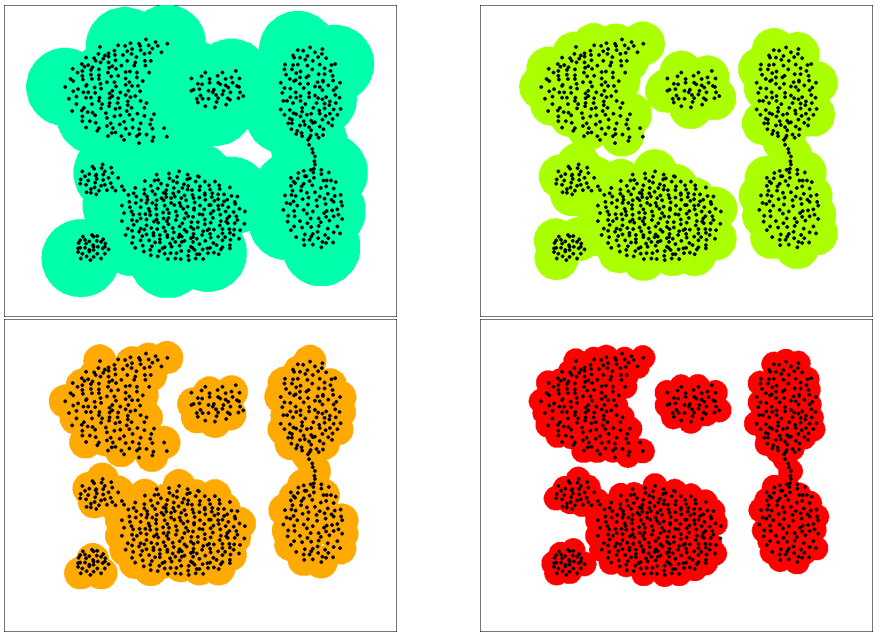


FIGURE 3.3 – Sorties de l'algorithme CM appliqué avec quatre d_c décroissant sur le jeux de données Agrégation.

Enfin, pour chaque tirage d'un nouveau point x_i, on doit calculer les distance entre x_i et toutes les boules existantes, ce qui peut mener à des coûts prohibitifs si ces boules commencent à devenir nombreuses.

3.3 Procédure CM itérée à fenêtre décroissante (ICMDW)

Le sous-algorithme ICMDW itère la procédure CM, la version complétée de la pré-CM décrite dans la section précédente, avec un rayon dégressif $d_c^{g+1} = d_c^g R$ entre chaque itération g successive, où $R \in]0, 1[$ est un paramètre qui va jouer sur la vitesse de décroissance du rayon au fil des itérations.

Une itération utilise les informations résultant de la précédente afin de réduire considérablement le nombre de distances à calculer à chaque tirage de x_i. Cette méthode itérée construit in fine une collection de boules $\{B_j^g\}_{g,j}$ de tailles variables et possédant chacune des informations de proximité avec leur pairs de génération contiguë [1].

Afin de simplifier la compréhension, nous commençons par introduire diverses définitions et décrire de manière informelle la sous-procédure CM ainsi que l'algorithme complet ICMDW, pour finalement les décrire en détail.

1. de même génération g ou écartées d'une seule génération (g, $g+1$ ou $g-1$)

3.3.1 Définitions, notations et présentation intuitive de ICMDW

Terminologie :

Boule Terminale : Une boule est terminale si elle contient moins de $N_{\min}$ points où $N_{\min}$ est un paramètre à choisir mais peu contraignant.

Attachée : On dit qu'une boule $B^g(x, d_c.r)$ est attachée à une boule de la génération précédente $B^{g-1}(y, d_c)$ si $d(x, y) \leq (r+1)d_c$.

Filles de B : Noté $D(B)$, les filles de la boule B sont les boules de la génération suivante qui lui sont attachées.

Mère de $\mathbf{B^g(x, d_c)}$: Dénoté par $M(B^g(x, d_c))$, la mère de la boule $B^g(x, d_c)$, est la boule de la génération précédente la plus proche contenant son centre x.

Mères de x : Lorsque la génération courante est g, les mères d'un point x sont les boules de génération $g-1$ contenant x. Elles sont dénotées par $M(x)$. $M^c(x)$ est la boule la plus proche de x parmi ses mères $M(x)$.

Sœurs de $\mathbf{B^g(x, r)}$: Noté $S(B)$, les sœurs de la boule $B^g(x, r)$ sont les filles de la mère de $B^g(x, r)$.

Maintenant que les notions de bases ont été introduites, nous pouvons exposer la propriété suivante :

Propriété :

Si $x_i \in B_j^g$ et $x_i \in B_k^{g+1}$ alors B_k^{g+1} est attachée à B_j^g.

Preuve :

Soient $B_1(0, r)$ et $B_2(c, r.R)$ deux p-sphères où $c \in \mathbf{R}^p$ et $R \in]0, 1[$ est le ratio de décroissance. Supposons qu'il existe $x \in \mathbf{R}^p$ tel que $x \in B_1$ et $x \in B_2$. Alors :

$$\left\{ \begin{array}{l} x \in B_1 \\ x \in B_2 \end{array} \right. \Rightarrow \left\{ \begin{array}{l} ||x|| \leq r \\ ||c - x|| \leq r.R \end{array} \right. \Rightarrow \left\{ \begin{array}{l} ||x|| \leq r \\ ||c|| - ||x|| \leq r.R \end{array} \right. \Rightarrow \left\{ \begin{array}{l} ||x|| \leq r \\ ||c|| \leq r.R + r \end{array} \right. \tag{3.4}$$

Cette propriété est très importante puisqu'elle implique que, lorsqu'à une itération g on vérifie à quelles boules existantes l'observation courante x_i appartient, on n'a pas besoin de calculer les distances avec toutes les boules mais seulement avec les

quelques boules de génération g attachées à la boule mère de x_i (de génération $g-1$). C'est sur cette propriété que s'appuie ICMDW, algorithme que nous allons désormais pouvoir introduire.

L'idée principale de ICMDW est la suivante. La première étape consiste en une itération de la méthode CM avec un rayon d_c^1 relativement grand, dont le choix sera discuté plus tard. Ensuite la procédure CM est réitéré avec un rayon $d_c^2 = Rd_c^1$ plus petit que le précédent car $R \in]0,1[$.

Comme décrit dans la section traitant de la sous-procédure pré-CM, à chaque itération toutes les observations x_i sont passées en revue une fois afin de savoir si elles appartiennent à une ou plusieurs boules créées lors de l'itération courante. Si l'on effectuait cette étape naïvement, nous aurions à calculer jusqu'à c_g distances pour chaque itération g et chaque observation x_i, où c_g est le nombre de boules créées à l'itération g.

Au lieu de cela la procédure CM se repose sur les informations de liens de parenté entre les boules courantes et antérieures ainsi que de la propriété précédente pour réduire drastiquement le nombre de distances point-boules à calculer lors du tirage de chaque observation x_i.

De plus, chaque nouvelle itération peut juger certaines zones trop peu denses et « geler » les observations s'y inscrivant, c'est à dire interdire leur tirage pour toute itération suivante. Ces zones ne sont alors plus développées en de plus petites boules. Autrement dit les boules continuent génération après génération à donner naissance à de plus petites boules tant qu'elles possèdent plus de $N_{\min}$ points et stoppent lorsque la zone n'est plus assez dense relativement au rayon qui décroît. C'est ce dernier aspect qui rend le rayon d_c^g adaptatif et dynamique.

3.3.2 ICMDW : le sous algorithme CM et son itération

Nous allons maintenant pouvoir décrire plus en détail le fonctionnement de la procédure CM, la méthode qui est au cœur de ICMDW.

Supposons que l'algorithme est à la génération g. Pour chaque observation x_i on cherche en premier lieu à construire J^g, l'ensemble des boules de génération courante

Algorithm 3 Iterated CM With Decreasing Window

procedure ICMDW(d_c^1, $N_{\min}$, R)
    $g \leftarrow 1$
    CM($d_c^1, N_{\min}$)
    while it remains at least one non-terminal ball **do**
        $g \leftarrow g + 1$
        $d_c^g \leftarrow d_c^{g-1}.R$
        CM($d_c^g, N_{\min}$)
    end while
end procedure

g auxquelles x_i appartient. Pour ce faire, on pourrait naïvement calculer les distances entre x_i et toutes les boules de génération courante, mais nous utilisons les liens de parenté obtenus à l'itération précédente pour chercher parmi un sous ensemble contenant bien moins d'éléments. Ce sous ensemble, que l'on note A^g, est construit en chargeant $m^{g-1} := M^c(x_i)$ (la mère la plus proche de x_i) pour ensuite charger les filles de cette mère ($A^g = D(m^{g-1})$), les seules qui peuvent potentiellement contenir x_i d'après la propriété 3.4.

J^g est donc construit en calculant les distances entre x_i et les boules appartenant à A^g uniquement et en retenant celles à une distance inférieure à leur rayon. Si $J^g \neq \emptyset$ alors les boules de J^g augmentent leur compte de densité de 1. Dans le cas contraire, une nouvelle boule $B_k^g(x_i, d_c^g)$ est créée et initialisée avec un compte de densité partiellement hérité des voisines de la même manière qu'expliqué dans la section précédente (Selon la formule 3.3). Enfin, J^g est enregistré afin d'être en mesure de charger les mères de x_i à l'itération suivante.

Après que toutes les observations x_i aient été traitées, on vérifie la densité des boules de la génération qu'on vient d'achever. Celles qui ont un compte de densité inférieur à $N_{\min}$ seront identifiées comme terminales. Ensuite les points n'appartenant qu'à des boules terminales sont gelés.

Les points gelés à une itération ne seront plus tirés lors des itérations suivantes et donc ne pourront plus participer à la création de nouvelles boules. Ceci permet à la fois de réduire le nombre de points à traiter à chaque itération mais surtout d'adapter dynamiquement la taille des boules et leur nombre à la densité locale.

Ces points venant d'être gelés à une itération g sont toutefois utilisés une dernière

Algorithm 4 CM

procedure $CM(d_c, N_{\min})$
  for all not frozen x_i **do**
    $m^{g-1} \leftarrow M^c(x_i)$ ▷ Load closest mother of x_i
    $A^g \leftarrow D(m^{g-1})$ ▷ Load daughters of M
    $\overrightarrow{d} \leftarrow \text{distance}(x_i, A^g)$
    $J^g \leftarrow \{j | \overrightarrow{d}[j] < d_c\}$
    if $J^g \neq \emptyset$ **then**
      $\forall j \in J^g, \rho_j^g(B_j^g) \leftarrow \rho_j^g(B_j) + 1$
    else
      Create the new ball $B_k^g(x_i, d_c)$
      $\rho_{\min} \leftarrow \min(mean(\rho_{\text{inter}}), 1 + \sum_m \frac{OL_1(\frac{d(B_k(x_i,d_c),B_m)}{d_c},p)}{OL_1(0,p)}.\rho(B_m))$
      $\rho_k(B_k(x_i, d_c)) \leftarrow \rho_{\min}$
      Record to which balls $B_k^g(x_i, d_c)$ is attached to
      $J^g \leftarrow k$
    end if
    $M(x_i) \leftarrow J^g$
  end for
  for all x_i frozen at last iteration **do**
    $m \leftarrow M^c(x_i)$ ▷ Load closest mother of x_i
    $s \leftarrow D(m^{g-1})$ ▷ Load daughters of m^{g-1}
    $\overrightarrow{d} \leftarrow \text{distance}(x_i, A^g)$
    $J \leftarrow \{j | \overrightarrow{d}[j] < d_c\}$
    $\forall j \in J, \rho_j^g(B_j^g) \leftarrow \rho_j^g(B_j^g) + 1$
  end for
  $\{B_l^g | \rho_l^g(B_l^g) < N_{\min}\}_l$ are set terminals
end procedure

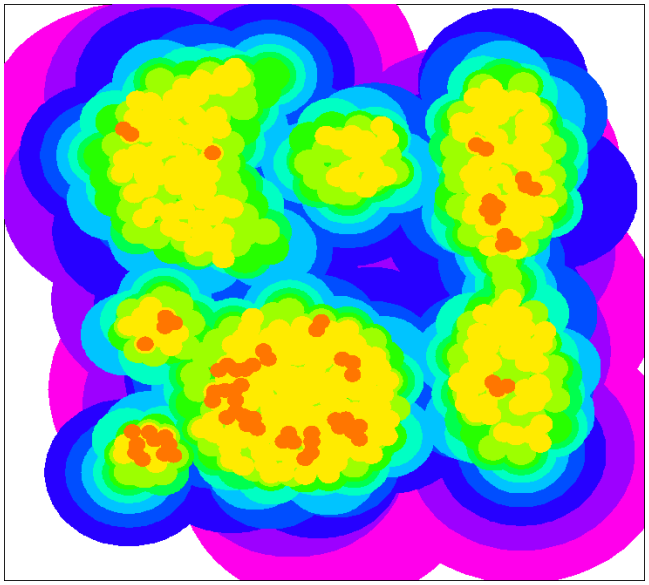
fois à l'itération suivante $g + 1$ pour augmenter la densité des boules auxquelles il appartiennent, mais en revanche jamais pour créer de nouvelles boules. L'algorithme ne s'arrête tant qu'il reste des boules non-terminales ou, de manière équivalente, tant qu'il reste des points non-gelés.

Le réseau de boules obtenu sur le jeu de données Agrégation en appliquant ICMDW avec les paramètres $N_{\min} = 4$ et $R = \frac{3}{4}$ est illustré dans la figure 3.4. Les détails algorithmiques de CM et ICMDW sont donnés respectivement dans les algorithmes 4 et 3.

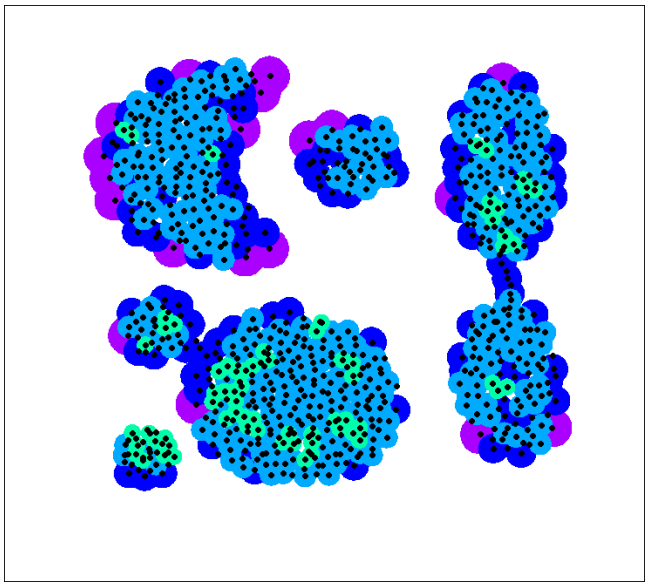
Cet exemple obtenu sur le jeu de données Agrégation permet d'apprécier visuellement la répartition des boules et leur taille relative. Les boules les plus larges, en violet sur la figure (A) sont non-terminales car elles possèdent plus de $N_{\min}$ points. Il faut attendre quelques générations que le rayon devienne suffisamment petit pour créer des boules terminales. Si l'on se concentre sur la deuxième illustration (B), on peut constater qu'en ne conservant que les boules terminales la densité réelle se retrouve relativement bien restituée. De plus, cette illustration a le mérite de mettre en évidence le fait que tous les points appartiennent bien à au moins une boule terminale.

On peut également remarquer que de nombreuses boules (les non-terminales de basse génération) ne servent pas à l'interprétation de la densité. En effet, la répartition n'est au final caractérisée que par les quatre dernières générations sur les neuf. Les non-terminales sont cependant très importantes car elles sont l'expression des états intermédiaires du réseau qui a permis la construction des boules terminales avec un moindre coût. Elles servent également à la phase de partitionnement de Ballstering.

Il est intéressant de remarquer que le réseau de boules ainsi obtenu peut être apparenté à un un arbre hiérarchisé. Contrairement aux arbres binaires, où un père ne peut avoir que deux fils, chaque nœud père (ou boule mère) peut donner naissance à plusieurs fils (boules filles). De plus, là où les arbres binaires séparent définitivement après un nœud, les différentes branches sont inter-connectées puisque une même fille peut avoir plusieurs mères. Ces deux types d'arbres se rejoignent sur l'aspect hiérarchisé puisqu'une boule de génération plus élevée sera obligatoirement située sur un branche inférieure à une boule de génération antérieure. Aussi ce réseau obtenu peut



(A) Toutes les boules



(B) Boules terminales uniquement

FIGURE 3.4 – Affichage des boules obtenues avec ICMDW sur Agrégation. Pour rendre compte de la densité, chaque boule possède une couleur liée à sa génération. Plus une boule est petite, donc de haute génération, plus elle vire vers le rouge.

être assimilé à une carte de densité, ce dont on discute au chapitre suivant.

Chapitre 4

Exploitation du réseau de boules : carte de densité et clustering

On a pu voir dans le chapitre précédent comment la procédure ICMDW produit un réseau de boules de tailles adaptées à la densité locale. Cet ensemble de boules accompagné des informations de parenté/proximité forme une arborescence dont chaque boule est un nœud et peut donc être exploité de plusieurs manières.

Nous avons exploré plusieurs pistes pour l'estimation de la densité à partir de ce réseau de boules et établi deux manières différentes d'en obtenir une. La première façon d'estimer la densité, que nous appelons « surface-grille », produit des valeurs numériques sur une grille tandis que la seconde (générationnelle) conserve son aspect d'arbre hiérarchisé en jugeant la densité uniquement sur la génération (ou profondeur au sein de la hiérarchie) et non pas sur les comptes de densité. Chacune d'elles ont leurs avantages et inconvénients dont on discutera à la fin de ce chapitre.

Le second point est le plus important car il concerne le clustering. En effet, nous allons voir qu'en se basant sur les valeurs numériques ρ_i des boules du réseau obtenu avec ICMDW on peut bâtir un clustering semblable à FDPC, en beaucoup plus rapide que l'original. Nous montrerons ensuite qu'en s'éloignant de la logique de FDPC et en s'appuyant sur les nouvelles possibilités qu'offre la densité générationnelle, nous pouvons produire un clustering de qualité et de robustesse bien supérieures à FDPC.

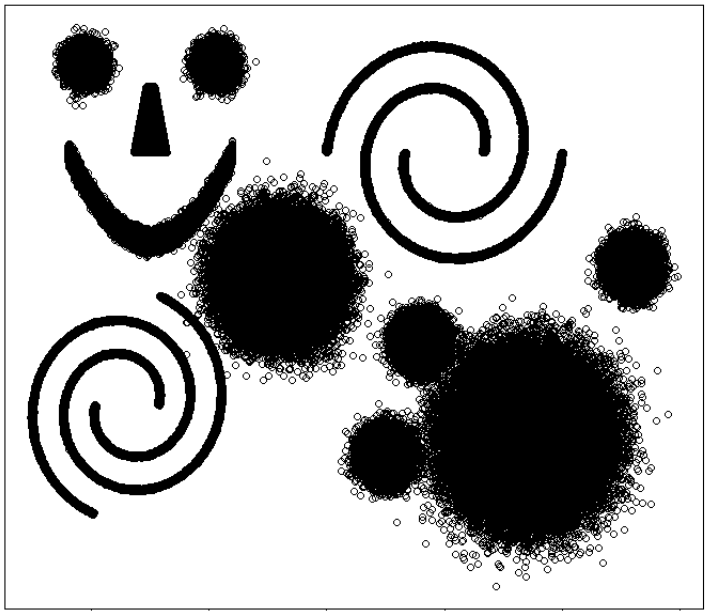


FIGURE 4.1 – Jeu de données artificiel que nous avons appelé Smiley.
Il Contient 13 clusters pour 1 million d'observations.

4.1 Cartes de densité

Le réseau de boules obtenu grâce à ICMDW peut fournir deux sortes de cartes de densités. L'une et l'autre ont leurs avantages, inconvénients et solutions potentielles que nous allons expliciter et illustrer au travers du jeu de données bi-dimensionnel que nous avons généré. Ce jeu de données artificiel que nous appelons *Smiley* en référence à la forme qui y apparaît est mis en lumière dans la figure 4.1. Ce dernier contient 13 groupes dont 7 Gaussiennes, 4 bras de spirales intriqués dans deux spirales distinctes et 2 clusters supplémentaires de formes particulières qui forment le nez et la bouche du smiley.

4.1.1 Densité de type surface-grille

La première façon de représenter la densité que nous allons exposer se base sur les comptes de densités ρ_i et utilise une procédure que nous appelons Tree-Dive car, en quelque sorte, elle plonge dans l'arbre de boules pour récupérer les densités. Dans un premier temps, afin d'obtenir des densités cohérentes entre toutes les générations de boules, on introduit la densité relative ρ'_i qui est calculée en divisant leur compte

par leur volume :

$$\rho'_i = \frac{\rho_i}{OL_{r_i}(0,d)},$$

où d est la dimension du jeu de données et r_i le rayon de la boule qui possède le compte de densité ρ_i. Cette nouvelle mesure de la densité permet de prendre en compte l'envergure de la boule qui contient ρ_i objets.

Pour rendre compte de la densité, seules les boules terminales seront au final indirectement utilisées puisque ces dernières sont les seules à être représentatives, les autres étant souvent trop grandes pour rendre compte fidèlement de la densité.

Afin de construire cette carte on définit en premier lieu une grille de points uniformément répartis sur tout l'espace, en nombre dépendant de la résolution que l'on souhaite. On va ensuite chercher à associer à chaque point de cet espace une des densités relatives ρ'_i, celle là même qui est la plus pertinente compte tenu de la carte obtenue avec ICMDW.

En suivant le choix que nous avons fait, construire cette première carte de densité revient à savoir à quelle boule terminale la plus dense chaque point de la grille appartient, et associer à ce point la densité numérique relative ρ'_i de cette boule. Une fois tous les calculs effectués il en résulte une grille de points renseignant la densité sur tout l'espace dont on peut afficher la surface si le problème est bi-dimensionnel.

Pour ce faire on applique à chaque point x_i de la grille la procédure Tree-Dive qui s'apparente à une recherche en profondeur d'abord. Cette dernière porte ce nom puisqu'elle va récursivement plonger génération après génération au sein de de l'arbre qu'est le réseau de boules, jusqu'à atteindre une feuille de cet arbre, c'est à dire une boule terminale.

Considérons en guise d'illustration le cas concret où l'on veut connaître la valeur de la densité associée à un point arbitrairement choisi de la grille x_i : On part de la toute première génération, et on cherche parmi les boules de première génération laquelle est la plus proche contenant x_i. On charge ensuite les filles de cette dernière pour trouver parmi elles la plus proche contenant x_i.

On continue à plonger récursivement dans l'arbre de cette manière (charger fille ;

trouver la plus proche ; charger fille...) jusqu'à tomber en dehors de toute boule. Parallèlement on retient quelle est la boule terminale de plus haute densité rencontrée au cours du plongeon. Ainsi lorsque le plongeon est terminé on peut renvoyer la densité de cette dernière.

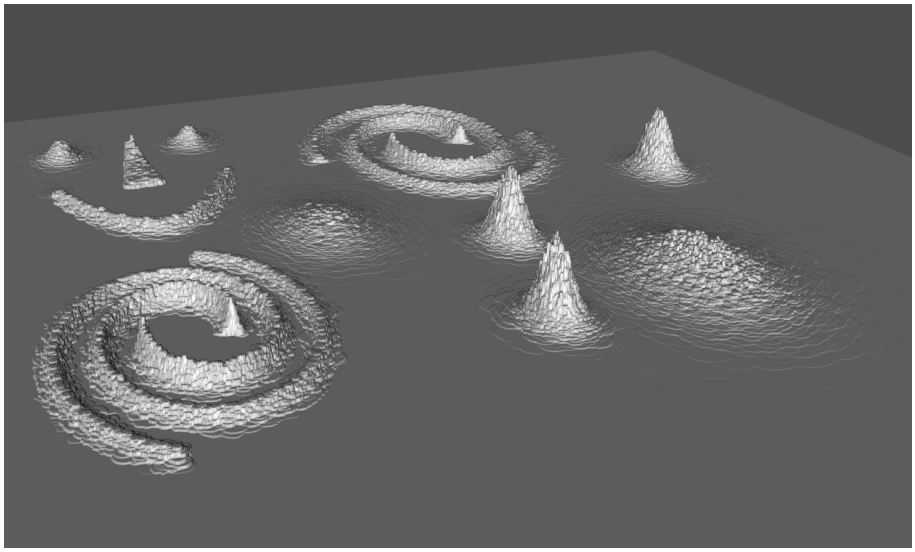
Ainsi en résumé, pour chaque point de la grille on relève la densité numérique ρ'_i de la boule terminale de plus haute densité à laquelle l'observation appartient. Si le jeu de données est bidimensionnel, on est capable afficher en $3d$ la surface que définit cette grille. Un exemple sur Smiley sous deux angles différents est donné dans la figure 4.2. Bien que n'étant pas toute lisse, cette carte de densité reproduit plutôt fidèlement la répartition des données. Pour une meilleure compréhension, les étapes de Tree-Dive sont explicitées dans l'algorithmique 5.

Algorithm 5 Tree-dive

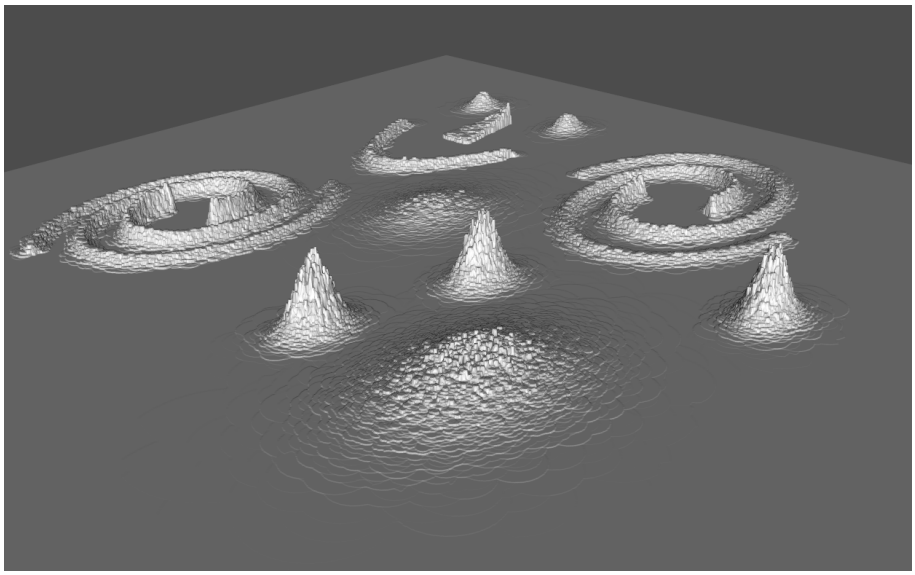
procedure TREE-DIVE(x_i)
  $q \leftarrow 1$ ▷ Begin at first generation
  $Terminal \leftarrow False$
  Load B_M ▷ Virtual original ball (generation 0)
  Load $D^q \leftarrow D(B_M)$ ▷ daughters of B_M
  LastBT$\leftarrow \emptyset$
  while $D^q \neq \emptyset$ **do**
    $D^q_{in} \leftarrow \{B^q_i | distance(x_i, B^q_i) < d^q_0\}$
    $j \leftarrow argmin_i(distance(B, D^q_{in}[i]))$
    $B_M \leftarrow B^q_j$
    $D^q_{in,sup} \leftarrow \{B^q_i | \rho(B^q_i) \geq \rho(LastBT)\}$
    LastBT $\leftarrow$ Closer terminal ball of $D^q_{in,sup}$ to x_i
    $q \leftarrow q + 1$
    Load $D^q \leftarrow D(B_M)$ ▷ daughters of B_M
  end while
  return LastBT
end procedure

4.1.2 Densité générationnelle

La carte de densité générationnelle ne se base pas sur les valeurs numériques ρ_i pour décrire la densité mais sur la génération uniquement. En effet, ICMDW construit des boules de plus en plus petites tant que celles-ci contiennent plus de $N_{\min}$ points. Donc plus la concentration de points (et donc la densité) en une zone est élevée, plus cette zone sera décrite par des boules de haute génération. Ainsi, une boule de génération g est considérée comme moins dense qu'une de génération $g + 1$ et plus



(A) Densité selon le premier angle.



(B) Densité selon le deuxième angle

FIGURE 4.2 – Densité de type surface-grille sur le jeu de données Smiley enregistré sous deux angles différents. Une surface est obtenue en relevant la densité sur une grille régulière, grâce à la procédure Tree-Dive.

dense qu'une autre de génération $g-1$, chaque génération représente donc un niveau de densité. Cette manière de décrire la densité est particulière pour plusieurs raisons.

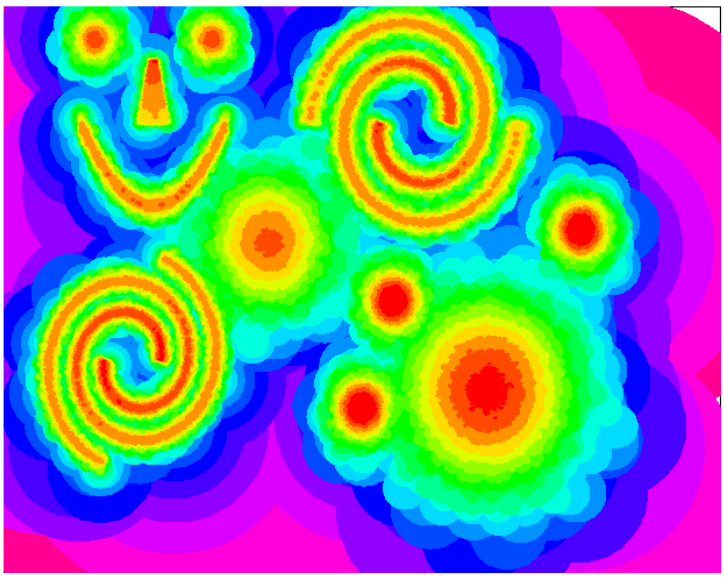
Tout d'abord un écart d'une génération ne veut pas forcément dire que la densité est différente. En effet si on considère l'analogie avec une variable réelle qu'on aurait discrétisée en classes, deux éléments de classes différentes peuvent être au final très semblables si elles se trouvent proches de la borne qui sépare ces deux classes. Il en va de même pour les boules produites par ICMDW car un écart de compte de densité de quelques unités seulement peut faire passer à la génération suivante. En revanche cette observation n'est pas transitive. Bien que deux boules ayant une génération d'écart peuvent être de densité équivalente, un écart de deux générations ou plus est obligatoirement révélateur d'une densité locale significativement différente.

Également, puisque les densités sont rangées en catégories, on peut dénoter une perte de précision dans cette manière de définir la densité. On ne fera plus la différence par exemple entre deux boules pourtant de même génération mais dont une contient deux fois plus de points que l'autre. Heureusement, ceci intervient là où cette précision est très sujette à l'aléa et donc peu fiable. On pourrait presque se risquer à affirmer que cette perte de précision est souhaitable si l'on s'intéresse à la robustesse de l'algorithme.

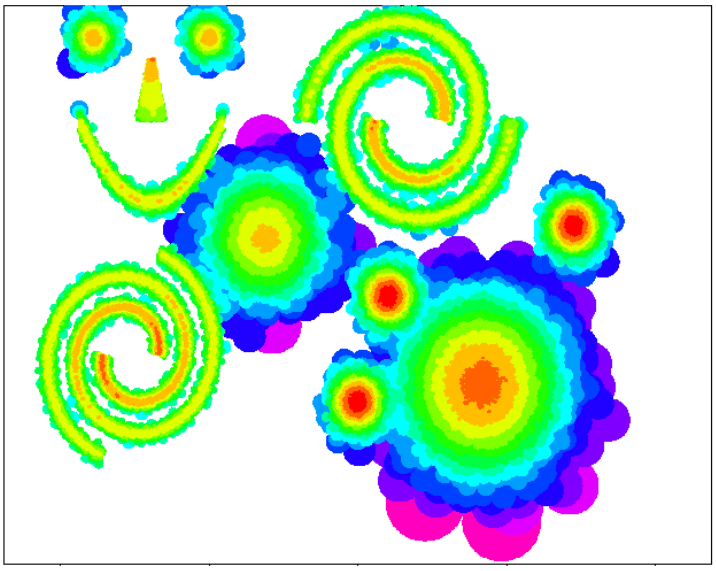
Cette seconde façon de représenter la densité a la particularité d'être directement donnée en sortie de l'algorithme, car l'arbre de boules est en quelque sorte lui-même la carte de densité, et d'être illustré très rapidement en ayant juste à afficher les boules.

Afin d'apprécier graphiquement la densité en $2d$ il suffit de définir une palette de couleurs qui contient autant de couleurs que de générations et d'afficher toutes ces boules avec leur couleur associée, en affichant en priorité les boules de plus haute génération. Si on prend en exemple le jeu bidimensionnel *Smiley* et une palette de couleurs variant du violet au rouge, la carte de densité qu'on obtient est illustrée dans la figure 4.3.

Dans cette figure on peut constater deux cartes de densité équivalentes, l'une représentant uniquement les boules terminales et l'autre toutes les boules sans distinction.



(A) Densité générationnelle où toutes les boules sont représentées.



(B) Densité générationnelle où uniquement les boules terminales sont représentées.

FIGURE 4.3 – Densité générationnelle obtenue avec ICMDW sur le jeu de données Smiley. Dans la première image (A) toutes les boules, non-terminales et terminales, sont indifféremment représentées.

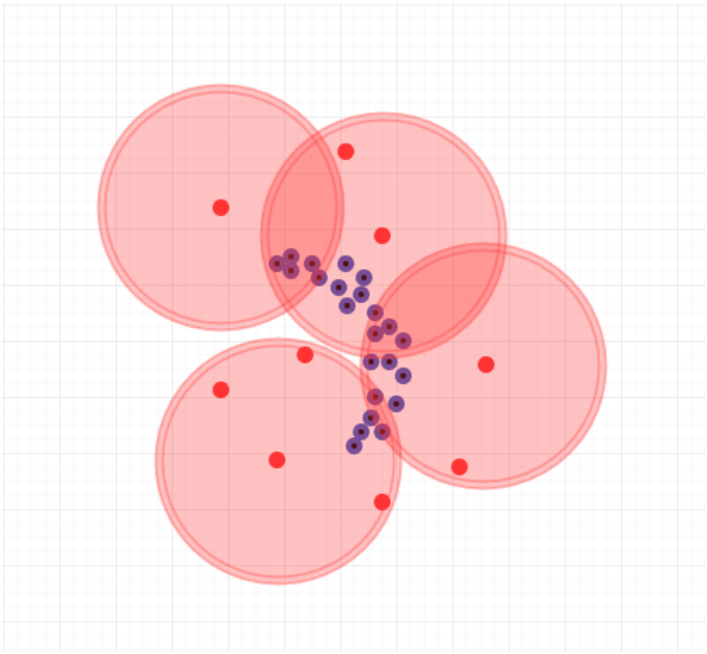


FIGURE 4.4 – Illustration de la mauvaise représentation de la densité lorsque les boules sont construites sur du bruit avoisinant un cluster. Puisqu'elles ne font qu'effleurer le cluster et que cela induit une faible densité, elles peuvent ne pas avoir assez de densité pour donner naissance à de plus petites boules. Elles sont donc terminales et ce sont elles qui représentent la densité. Cet exemple est très largement exagéré et n'arrive presque jamais dès lors que les clusters contiennent un nombre important d'observations.

On pourra trouver des illustrations plus complexes dans la section 6 qui présente de nombreux résultats.

4.1.3 Limitations

Boules terminales non-représentatives

Le premier point gênant que nous abordons est la détérioration occasionnelle de la représentation de la densité qui peut intervenir en zone dense, de structure particulière et en présence de bruit. Considérons la figure 4.4 qui illustre un exemple où les deux conditions nécessaire à cet occurrence sont réunies. On se place dans le cas très peu probable où toutes les boules ont été créées par malchance que sur des points bruités et où la densité forme un demi-arc de cercle relativement fin. S'il se trouve que ces boules contiennent moins de $N_{\min}$ points, elles sont terminales et donc ce sont elles qui représenteront (très mal) cette zone.

Ce cas est tout de même exagéré car une boule a plus de chance d'être créée en zone dense qu'en zone bruitée mais à certains endroits ce genre phénomène peut intervenir. Également, cela ne peut presque arriver que pour les structures fines et contenant peu de points. Si cette structure contenait un nombre de points plus important, alors ces boules seraient non-terminales et ce sont des boules plus petites qui caractériseraient plus fidèlement cette zone.

Le mot « presque » a été employé plus haut car un problème similaire peut arriver même si la structure est dense. Pour que cela intervienne il faut un concours de circonstances particulier et rare en pratique. Le cluster considéré doit être dense et très bien délimité et dans le même temps il faut qu'une boule soit créée sur une zone bruitée proche, de telle manière qu'elle chevauche que très légèrement ce cluster afin d'être terminale.

Dans de telles circonstances, bien qu'étant créée en zone de très faible densité, l'envergure de la boule permet de capter des données appartenant à une zone dense. Lorsqu'elle en capte suffisamment pour ne pas être terminale il n'y a pas de problème car elle restera virtuelle et de plus petites boules rendront fidèlement la zone. En revanche lorsqu'elle n'en capte qu'une petite partie et qu'elle devient terminale tout en ayant une densité non négligeable, ceci induit une estimation de la densité bien plus haute que celle quasiment nulle à laquelle on s'attendrait. Ce phénomène a donc tendance à « étaler » la densité réelle. Ceci dit, ce dernier point est un problème que beaucoup d'algorithmes d'estimation de densité ont, comme les estimateurs par noyau par exemple mais cet étalage est plus régulier, constant, prévisible et contrôlable alors que le notre est ponctuel, imprévisible et irrégulier.

Boules tardives

Aussi la première façon de représenter la densité se base sur des ρ_i qui peuvent être légèrement éloignés de la réalité dans certains cas particuliers. C'est le cas par exemple des boules tardives qui définissent la majeure partie de leur densité avec la formule 3.3 qui n'est qu'un substitut et ne reflète pas avec précision la véritable densité. Ce problème peut être cependant réglé en réinitialisant les ρ_i puis en appliquant pour chaque point du jeu de données un Tree-Dive qui va incrémenter la densité de toutes

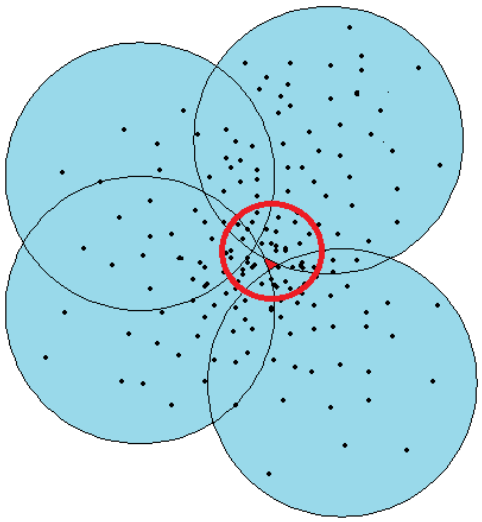


FIGURE 4.5 – Trou dans la densité. Quand bien même les boules bleues seraient très dense, tout Tree-Dive sur un point contenu dans la zone rouge (au milieu) renvoi une densité nulle

les boules terminales auquel le point appartient au cours de son plongeon dans le réseau. Cette étape qui coûte relativement cher en terme de calculs permet tout de même d'avoir les comptes de densité exacts et ainsi régler ce problème.

Cette limitation n'en est pas une pour la carte de densité générationnelle puisqu'elle n'utilise pas les comptes de densité.

Trous dans la densité

Il n'arrive que pour la carte de densité surface-grille, et même en zone très dense, il apparaisse des "trous" dans la densité. Considérez en effet le schéma dans la figure 4.5 dans lequel tous les points appartiennent bien à une boule terminale. Cette zone est relativement dense, pourtant notre évaluation de la densité au milieu entre les 3 boules (en rouge) sera nulle, car aucune boule terminale ne le recouvre.

Afin d'éviter ce problème qui n'a lieu que pour la densité de type surface-grille plusieurs solutions sont possibles. La première, très simple, que nous utilisons est de considérer à posteriori que les boules sont très légèrement plus grandes qu'elles ne le sont lorsqu'on veut représenter la densité-grille. La seconde serait de lisser la carte en utilisant un noyau gaussien qui pourrait s'appuyer sur les informations du réseau de

boules pour être exécuté rapidement. La taille de fenêtre pourrait être dynamique elle aussi et déterminée automatiquement grâce aux informations extraites par ICMDW.

Limitations mineures

Une limitation évidente mais qui doit être mise en lumière est le coût calculatoire d'obtention de la carte de type surface-grille en dimension élevée. En effet, le nombre de points de la grille augmentant de manière exponentielle quand la dimension p s'élève, il peut s'avérer nécessaire de réduire la finesse de la grille en haute dimension.

La carte générationnelle quand à elle est très rapide à obtenir mais sa structure la rend relativement contraignante à l'utilisation. Afin de pouvoir l'exploiter, il faut obligatoirement un algorithme qui prend en compte cette structure, mais sa nature d'arbre enraciné ouvre d'autres possibilités d'exploitation tout aussi intéressantes.

Pour toutes les raisons évoquées dans cette section, il faudrait étudier d'avantage ces limitations avant de prétendre être une alternative réelle aux autres estimations de densité classiques [53].

En revanche ces deux cartes sont au final très rapides à obtenir, ne demandent aucun paramètre à fixer pour de gros volumes de données et la nature de ses imperfections peuvent ne pas impacter sur le travail qu'on essaie d'effectuer, comme c'est le cas de Ballstering.

Cependant elles pourraient éventuellement être améliorées pour être plus fidèles à la densité réelle ce qui, combiné à la rapidité d'obtention et l'automaticité, pourrait fortement concurrencer les autres moyens d'estimation de densité non-paramétriques.

4.2 Du réseau de boules au Partitionnement

Une fois ce réseau de boules obtenu, il convient de chercher à en extraire une partition. Plusieurs façon d'effectuer cette tâche nous sont apparues. La première que nous allons présenter est une adaptation directe de l'esprit de FDPC à la structure d'arbre fournie par ICMDW.

Bien que beaucoup plus rapide, cette première version qui se repose sur les densités numériques ρ_i souffre par héritage de certains problèmes de FDPC comme la mauvaise

détection des cœurs de clusters, les erreurs d'attribution des labels et la non-robustesse aux concentrations dues à l'aléatoire.

Ceci a donc motivé une recherche d'une procédure plus robuste pour extraire les clusters en s'appuyant notamment sur des informations supplémentaires fournies par l'arbre de boules. Cette deuxième version corrige toutes les limitations de FDPC exposées dans la section 1. Elle se repose entièrement sur le graphe qu'est la densité générationnelle.

4.2.1 Adaptation naïve de FDPC à notre structure arborescente

Comme nous l'avons annoncé plus haut, la version que nous allons présenter ici est l'adaptation directe à notre réseau de boules de ce qui est exécuté dans FDPC.

En guise de rappel, ils déterminent pour chaque point x_i la valeur associée δ_i qui est la distance entre x_i et le point le plus proche parmi ceux de plus haute densité (nphd). Pour calculer δ ils se basent sur la matrice des distances $N \times N$ obtenue préalablement, afin d'obtenir les ρ_i.

Dans notre cas, nous n'avons pas construit cette matrice. Au lieu de cela nous allons nous appuyer sur boules de l'arbre et leur densité numérique associée pour produire l'équivalent des δ_i pour les terminales uniquement, les seules représentatives de la densité théorique. Nous allons donc construire les δ_i pour chaque boule terminale. Cependant pour cela il est hors de question de calculer naïvement les distances entre toute paire de boules terminales. Bien qu'elles soient en nombre réduit comparativement à N, elles peuvent rester trop nombreuses pour calculer les distances de manière si brutale.

Or nous disposons des informations de liens de parenté entre les boules proches dans l'espace et en génération ce qui, comme nous allons le voir, nous permet de reconstruire les δ_i d'une manière très efficace.

Il s'agit donc de trouver pour chaque boule terminale la boule la plus proche parmi celles de plus haute densité et de relever la distance δ_i les séparant. Pour ce faire, on définit d'abord la notion de "champion". Afin de percevoir le sens de cette notion, plaçons nous pour l'instant dans le cas simple et fictif où toutes les boules

terminales sont toutes de génération maximum G et considérons la $k^{\text{ème}}$ boule B_k^{G-1} de génération $G-1$ (non-terminale donc).

Considérons maintenant ses filles de génération G que l'on note $A_k^G := D(B_k^{G-1})$. On définit alors le champion c_k^G de B_k^{G-1} comme étant la boule de plus haute densité parmi les filles A_k^G.

L'intérêt de différencier les championnes des autres boules est que, pour toutes les autres boules filles $A_k^G \backslash c_k^G$, on peut se permettre de chercher uniquement parmi les filles de leur mère la plus proche.

Autrement dit, chacune de ces boules non-championnes aura pour NPHD soit une boule plus proche que la championne c_k^G, soit la championne elle même. Pour ces non-championnes, on n'aura donc jamais à chercher plus loin que leur sœurs.

Concernant la championne c_k^G, il en est tout autrement. Aucune de ses sœurs ne peuvent prétendre être son NPHD car elle a une densité plus élevée, on doit donc chercher plus loin.

Pour comprendre comment on peut trouver le NPHD de cette championne, montons d'un étage dans la hiérarchie. plaçons nous donc maintenant à la génération inférieure $G-2$ et supposons que chaque boule de la génération $G-1$ a déterminé son champion. Prenons ensuite en exemple la $k^{\text{ème}}$ boule B_k^{G-2} et considérons ses boules filles A^{G-1}, ces dernières ayant donc chacune leur champion de génération G de renseigné.

Notre champion d'intérêt c_i^G porté par une boule $B_i^{G-1} \in A^{G-1}$ pourra alors trouver une boule terminale de densité plus élevée parmi les filles d'une des boules $B_j^{G-1} \in A^{G-1}$ qui porte un champion de plus haute densité que c_i^G.

En effet, si une boule de génération $G-1$ contient un champion de densité supérieure à la densité de c_i^G cela veut automatiquement dire qu'elle porte des filles dont au moins une (la championne) possède une densité plus élevée, donc candidate à être NPHD.

Puisqu'on cherche la plus proche, on sélectionne la boule appartenant à A^{G-1} la plus proche ayant un champion de densité plus élevé puis on cherche la plus proche parmi ses filles. Son NPHD pourrait se révéler être la championne elle même ou bien

une des sœurs de cette championne qui serait dans ce cas plus proche de c_i^G.

Tous les championnes portées par les boules A^{G-1} trouveront ainsi leur NPHD sauf une, celle de plus haute densité. Pour son cas, on doit donc à nouveau chercher avec le même raisonnement à la génération inférieure $G-2$.

Le principe sera analogue à la différence que lorsqu'on a trouvé la boule B_i^{G-2} de génération $G-2$ la plus proche parmi celles qui portent un champion de densité plus élevée, il ne sera plus suffisant de chercher le nphd parmi les filles de B_i^{G-2}. La raison est que ces filles sont non-terminales puisque sont de génération $G-1$. Il faudra donc à nouveau chercher la plus proche portant un meilleur champion pour charger ses filles qui seront elles de génération G et enfin trouver la boule terminale parmi ces dernières qui déterminera donc le NPHD.

Dans le premier cas nous avons pu tomber directement sur le NPHD dans le deuxième cas on a dû descendre de deux générations dans l'arbre pour trouver le NPHD. Si on se place maintenant à la génération inférieure $G-3$, il faudra descendre de trois générations pour trouver le NPHD et ainsi de suite.

On peut donc constater que plus on monte dans l'arbre, plus on plonge de haut pour retrouver les NPHD. Comme on le verra dans quelques paragraphes, la sous procédure qui se charge de trouver le NPHD d'un champion à partir d'une certaine génération g est un Tree-Dive similaire à celui présenté dans la section 4.1.1. Le même principe est utilisé récursivement en remontant les générations petit à petit afin de trouver tous les NPHD de toutes les boules terminales. On remonte donc ainsi jusqu'à la génération 1 où le champion de tous les champions est la boule de plus haute densité qui, comme pour FDPC, n'a pas de NPHD et prend une valeur δ arbitrairement grande.

Afin de gérer le cas plus général où les boules terminales peuvent appartenir à n'importe quelle génération, il suffit juste de considérer initialement que chaque boule terminale est son propre champion.

Grâce à cet exemple, nous avons pu percevoir l'idée générale de l'algorithme permettant l'établissement des NPHD de chaque boule terminale. Voyons maintenant détail comment cette idée se traduit algorithmiquement.

Élection des champions (voir Algorithme 6) :

Ici on ne suppose plus que toutes les boules terminales appartiennent à la génération G. Dans cette sous-partie on ne s'occupe pas encore de trouver le NPHD mais juste de déterminer les champions, densités de champions et niveaux de champions.

Les densités de champions et niveaux de champions sont deux nouveaux attributs construits pour chacune des boules, terminale ou pas. Pour chaque boule B_k^g on note d'une part $\rho_{champ}(B_k^g)$ la densité de son champion associé, et d'autre part $q_k^g \in \{0, ..., G-1\}$, le niveau de champion de B_k^g (uniquement pour les terminales) qui indique à quel niveau dans la hiérarchie on a dû monter pour trouver son NPHD.

En premier lieu les boules terminales initialisent leur ρ_{champ} avec leur propre densité et voient leur niveau de champion initialisé en se basant sur leur propre génération : une boule de génération g initialisera son niveau de champion avec

$$q_k^g \leftarrow G - g.$$

Puis on commence par chercher pour chaque boule non-terminale B_k^{G-1} de la génération $G-1$ la boule de plus haute densité-champion ρ_{champ} parmi ses filles (donc de génération G), qu'on notera B_c^G.

On modifie alors le niveau de champion de B_c^G pour le fixer à 1 $(= G - (G-1))$ et on met à jour également la densité de champion de B_k^{G-1}, la mère qui l'a promu champion de niveau 1 :

$$\rho_{champ}(B_k^{G-1}) \leftarrow \rho(B_c^G).$$

Une fois toutes les boules de la génération $G-1$ traitées on se retrouve avec certaines boules de l'ultime génération G promues en champion de niveau 1. Toutes les boules non-terminales de génération $G-1$ ont vu leur densité de champion mises à jour avec la densité de leur champion.

On passe à la génération suivante $G-2$: Pour chaque boule B_k^{G-2}, on cherche parmi ses filles la boule B_c^{G-1} qui porte le champion B_d^G de plus haute densité.

On promeut alors le niveau de champion de B_d^G en le passant à 2 ($= G - (G-2)$) et la densité de champion de B_k^{G-2} est mise à jour avec

$$\rho_{champ}(B_k^{G-2}) \leftarrow \rho_{champ}(B_c^{G-1}),$$

qui à ce niveau hiérarchique n'est autre que $\rho(B_d^G)$.

Ces étapes sont effectuées de manière analogue pour chaque génération et jusqu'à la première génération, générant des boules terminales championnes de divers niveaux, d'autant plus élevé qu'elles sont denses et éloignées d'autres points plus denses.

Construction des δ_i

Une fois les champions obtenus on peut déterminer les NPHD. Pour chaque boule terminale, on applique un dérivé de la procédure Tree-Dive que nous appelons Tree-Dive' (voir Algorithme 7) qui permet d'obtenir, à partir des informations des champions (niveau et densité), leur NPHD et donc la distance δ les séparant.

Pour une boule terminale B de niveau de champion q, Tree-Dive' fonctionne comme suit. Considérons la boule B_i^{G-q} qui a promu B au niveau q. Notre procédure s'initialise en chargeant $B_k^{G-q-1} := M(B_i^{G-q})$ la mère de B_i^{G-q} puis en chargeant les filles de la mère $S := D(B_k^{G-q-1})$.

On trouve et retient alors B_j^{G-q}, la sœur la plus proche ayant une densité-champion plus élevée que $\rho(B)$. Si B_j^{G-q} est terminale alors l'algorithme stoppe et c'est cette dernière qui sera son NPHD et servira à définir δ.

Dans le cas contraire, on réitère au niveau inférieur. On va charger les filles de B_j^{G-q}, chercher la plus proche possédant une plus grande densité que $\rho(B)$ puis s'arrêter là si elle est terminale ou continuer à plonger dans l'arbre en chargeant ses filles.

Ce sous algorithme plonge donc dans l'arbre jusqu'à tomber sur une boule terminale qui sera donc son NPHD et servira à définir δ.

Sélection des cœurs et affiliation

A ce stade puisque nous avons les densités ρ_i et les dépendances donnant lieu aux δ_i, nous pouvons utiliser la même façon de faire que FDPC. On trace alors le graphe de décision sur lequel on essaie de discriminer les points ayant à la fois une grande densité et une grande distance δ_i qui seront donc détectés comme cœurs de cluster.

On peut également se baser sur le processus de décision automatisé proposé par [35] où on calcule pour chaque point le produit $\gamma_i = \delta_i.\rho_i$ qui sont ensuite triés dans un ordre décroissant. Puis la règle du « turning point » est appliquée sur les γ_i.

Tout comme FDPC, les cœurs sélectionnés prennent un label unique, définissent un nouveau groupe chacun, et les autres boules obtiennent récursivement le même label que leur NPHD. Enfin, chacune des observations est attribuée au cluster auquel a été attribué sa boule mère.

Algorithm 6 Sélection des champions

procedure ChampionSelect(
)
  for $g = G-1, ..., 1$ **do**
    for all $B_l^g, l \in \mathbf{L}^g$ **do**
      if B_l^g is terminal **then**
        $\rho_{champ}(B_l^g) \leftarrow \rho(B_l^g)$
        $q_l^g \leftarrow G - g$ ▷ Upgrade champions level
      else
        $B_{temp}^{g+1} \leftarrow argmax_{B_i^{g+1} \in D(B_l^g)}(\rho_{champ}(B_i^{g+1}))$
        $\rho_{champ}(B_l^g) \leftarrow \rho_{champ}(B_{temp}^{g+1})$
        $q_j^{g+1} \leftarrow G - g$ ▷ Upgrade champions level
      end if
    end for
  end for
end procedure

Cette partie est très rapide comparé à l'établissement du réseau de boules mené par la procédure IMCDW. En l'état, notre algorithme fournit des résultats très similaires à ceux de FDPC tout en ayant une complexité de $O(nlog(n))$ alors qu'on partait de $O(n^2)$.

On s'est également affranchis du paramètre d_0 qui était difficile à choisir et très influant sur les résultats, en le remplaçant par un paramètre beaucoup plus robuste et dont on peut se passer du choix pour des données volumineuses.

Algorithm 7 Tree-dive'

procedure TREE-DIVE'(B, q)
  $Terminal \leftarrow False$
  Load B_M ▷ mother of the ball that promoted B to level q
  while $Terminal == False$ **do**
    Load $D^q \leftarrow D(B_M)$ ▷ daughters of B_M
    $D^q_{sup} \leftarrow \{B^q_i | \rho_{champ}(B^q_i) > \rho(B)\}$
    $j \leftarrow argmin_i(distance(B, D^q_{sup}[i]))$
    if B^q_j is a terminal Ball **then**
      $Terminal \leftarrow True$
    else
      $B_M \leftarrow B^q_j$
      $q \leftarrow q + 1$
    end if
  end while
  $\delta(B) \leftarrow dist(B, B_M)$
end procedure

De plus le rayon des boules d_0^g est adaptatif ce qui permet d'avoir de meilleurs résultats lorsque les partitions à discriminer sont non homogènes, de densités très différentes, car pour être bien représentés l'une nécessite un grand rayon tandis que l'autre zone nécessite un petit rayon.

Un point faible potentiel que notre méthode induit est la position des boules qui peut varier suivant l'ordre de tirage des observations. En effet, alors que FDPC calcule les densités locales pour chaque point, notre algorithme ne le fait que pour un certain nombre. Pour de petit jeux de données, le placement des boules peut influencer sur les résultats.

Aussi notre façon de faire peut ne pas donner exactement le NPHD. Considérez l'exemple illustré dans la figure 4.6 où on essaie de déterminer le NPHD de la championne rouge. Dans cet exemple le véritable NPHD est la championne bleue sauf que la mère de cette dernière est plus éloignée que la mère de la championne verte. Or notre procédure Tree-Dive cherche d'abord la boule mère la plus proche (verte) pour seulement ensuite chercher parmi ses filles.

Dans le cas de données volumineuses ce problème est négligeable lorsqu'il intervient proche des feuilles terminales de l'arbre car il n'y a pas besoins d'une précision très importante à un grain aussi fin. En revanche ce problème pourrait être très impactant s'il intervenait un peu plus haut dans l'arbre et se répétait plusieurs fois au

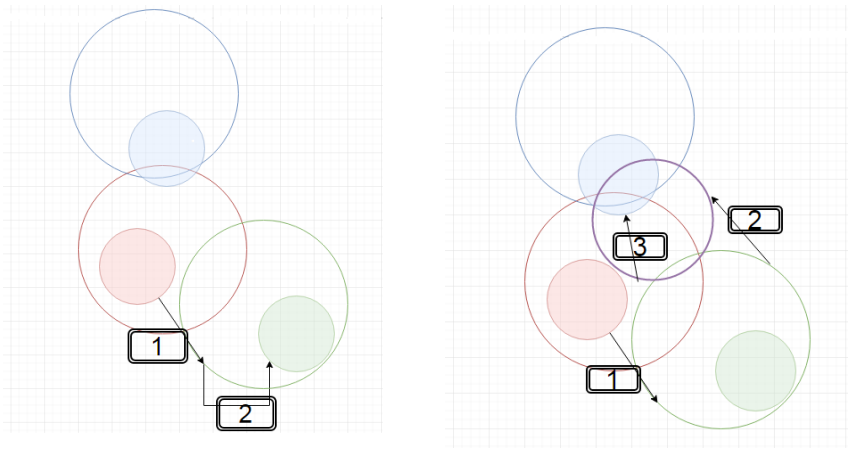


FIGURE 4.6 – Illustration de l'erreur potentielle sur la détermination du nphd.

cours du plongeon Tree-Dive'.

Heureusement, un phénomène le compense et le rend impossible à une échelle plus grande. En effet, si on considère la deuxième figure de 4.6 où une génération intermédiaire a été ajoutée, la mauvaise direction prise initialement vers la boule grand-mère verte revient automatiquement sur la boule mère violette car cette dernière est rattachée à la boule verte et est plus proche de la rouge. Puisque cette boule violette est attachée à la bleue le NPHD va se retrouver affecté correctement. Pour résumer donc, ce phénomène néfaste ne peut intervenir qu'à petite échelle, uniquement au niveau des feuilles de l'arbre car compensé sinon au cours de la descente.

En outre il hérite de la plupart des problèmes de FDPC qu'on a exposés dans un chapitre précédent à savoir la mauvaise détection des NPHD et des cœurs qui ne prennent pas en compte la densité avoisinante. Sans oublier que puisqu'il se base sur les comptes de densité il est sensible aux concentrations aléatoires et d'autant plus que le rayon est petit. Enfin la sélection des cœurs doit être faite manuellement pour être correcte et que la règle automatique du turning point que nous avons essayé est loin d'être optimale ni même robuste.

4.2.2 Partitionnement robuste de Ballstering

Conscient de ces problèmes évoqués précédemment, on a cherché à trouver une nouvelle manière d'effectuer le clustering. Après avoir exploré plusieurs pistes peu

satisfaisantes qui souffraient du bruit sur les densités ρ_i, nous avons décidé d'abandonner ces valeurs numériques.

Au lieu de compter sur les densités ρ_i, on a préféré se servir uniquement de la génération de chaque boule pour les discriminer, tout comme la carte de densité générationnelle. Bien sûr de l'information est perdue avec cette approche mais cette précision n'est pas nécessaire et s'en passer permet une approche très efficace et robuste qu'on va pouvoir développer plus bas.

Tout d'abord on redéfinit la notion de cœurs de cluster. Nous remanions celle de FDPC en ajoutant une condition sur la densité joignant un cœur à un autre de sorte à rejoindre la définition classique de cluster des méthodes à densité : un cœur de cluster est considéré comme étant à la fois un maxima local de la densité mais également séparé par une dépression de densité significative des autres maximums locaux.

Tout comme dans la version précédente chaque cœur définit un nouveau cluster mais l'attribution des autres boules ne se fait plus avec le concept de NPHD. On décide d'attribuer une boule à un des clusters existants en fonction de leur accessibilité, de la force du chemin depuis le cœur associé, cette force d'accessibilité étant maintenant basée sur la densité les séparant et non plus sur la distance brute.

Pour effectuer la sélection des cœur de clusters, on considère initialement toutes les boules terminales comme des cœurs potentiels. Une des boules de plus haute génération est d'abord choisie comme premier cœur et définit donc le premier cluster.

On va ensuite chercher et retenir toutes les boules terminales que l'on peut atteindre depuis ce nouveau cœur en se déplaçant de proche en proche dans le réseau en s'autorisant à descendre en génération mais sans jamais remonter. Nous appellerons cette recherche de boules atteignables depuis un cœur RS, pour « Reachability Searching » .

Les boules ainsi atteintes au cours de la recherche ne sont plus considérées comme des cœurs potentiels. Puis, parmi les boules candidates restantes, on prend à nouveau une de plus haute génération pour encore appliquer une procédure RS et ainsi de suite jusqu'à ce qu'il ne reste plus aucun cœur potentiel. La phase de détection des cœurs, résumée dans la très simple algorithmique 8, repose sur la RS qui elle est plus

compliquée et qu'on explicite maintenant.

Algorithm 8 Détection des cœurs

Initialisation : $L \leftarrow$ toutes les boules terminales
Initialisation : $C_l \leftarrow \emptyset$
while $L \neq \emptyset$ **do**
    Sélectionner B_c l'élément de plus haute génération de L
    Ajouter B_c à la liste des cœurs C_l
    Appliquer la RS à B_c et retenir S, les boules couvertes
    Éliminer les boules présentes dans S de L
end while

On avait énoncé que dans la RS on s'autorise à descendre autant qu'on veut en génération (donc en densité) mais jamais à remonter. Or en pratique, même une zone de densité uniforme (plate) peut être représentée avec des boules variant sur deux générations. Donc, afin d'absorber les petites variations de générations dues à l'aléatoire, on autorise la procédure RS à passer par des boules de génération $g_{\min} + 1$ où $g_{\min}$ est la plus petite génération rencontrée sur le chemin. Autrement dit on s'autorise à remonter d'une génération de plus que la plus basse génération par laquelle on a du passer.

Rentrons dans les détails. Une boule B_2 est atteignable depuis B_1 par une instance RS si B_2 peut être atteinte de proche en proche depuis B_1 en suivant les règles de déplacement suivantes :

— **Proche en terme de distance** : Depuis une boule $B(x, r)$ on ne peut directement explorer que les boules $\{B(x_i, r_i) | d(x, x_i) \leq r + r_i\}$, celles qui intersectent $B(x, r)$.

— **Proche en terme de génération** : Depuis une boule B^g de génération g, on ne peut se déplacer que vers des boules de génération contiguë ($g, g-1$ et $g+1$).

— **Ne pas remonter significativement en génération** : Considérons une boule B et $G_{\min}$ la plus petite génération $G_{\min}$ rencontrée dans le chemin qui

atteint B. Alors, depuis la boule B, on ne pourra plus atteindre des boules de génération strictement supérieure à $G_{\min} + 1$.

La recherche des cœurs et l'affiliation des cœurs sont faites en même temps. Pour cette raison on prévient que les détails algorithmiques de la sélection des cœurs sera explicitée plus loin.

Lorsqu'une boule est atteinte par une procédure RS instanciée par un coeur de cluster, elle est désormais affiliée à ce cluster. Cependant pour effectuer à bien ces deux opérations on introduit une nouvelle règle de déplacement basée sur une notion de force de chemin qui est ajoutée aux trois précédentes. Tout au long du processus, chaque boule conserve l'information de la force du chemin par lequel elle a été atteinte.

Ainsi, au cours d'une RS, une boule pourra être atteinte si elle vérifie les trois conditions primaires mais aussi si on arrive jusqu'à elle avec un meilleur chemin que toutes les instances RS qui ont pu l'atteindre. De plus, lorsqu'une boule déjà atteinte auparavant par une procédure RS est atteinte à nouveau avec un chemin équivalent à l'instance précédente, elle est n'est plus attribuée à aucun cluster.

On mesure la force ou longueur d'un chemin P dans l'arbre de boules par un procédé à mémoire courte se basant sur deux valeurs, $G_{\min}$ et c.

On définit $G_{\min}$ comme étant la génération de $B_{\min}$ qui est la première boule rencontrée dans le chemin P parmi celles de plus petite génération. On définit ensuite c comme étant le nombre de déplacements restants pour atteindre la destination B_p depuis $B_{\min}$.

De manière plus formelle, soit $P := (B_0, B_1, ..., B_{p-1}, B_p)$ un chemin constitué de p déplacements reliant deux boules B_0 et B_p. Si on définit j par :

$$j := \text{argmin}_k\{k|\text{gen}(B_k) \leq \text{gen}(B_i), \forall i \in \{0, ..., p\}\},$$

on peut alors définir les deux variables $G_{\min}$ et c comme suit :

$$G_{\min} := \text{gen}(B_j)$$

et

$$c := p - j.$$

On peut alors donner une mesure de la force d'un chemin qui a atteint B grâce aux deux variables $G_{\min}$ et c. Un chemin $P(B) := \{G_{\min}, c\}$ est strictement inférieur (plus court, meilleur) que $P'(B) := \{G'_{\min}, c'\}$ si :

$$\Big(\text{gen}(G_{\min}) > \text{gen}(G'_{\min})\Big) \textbf{ ou } \Big((\text{gen}(G_{\min}) = \text{gen}(G'_{\min})) \textbf{ et } (c < c')\Big).$$

Un chemin $P(B)$ est égal à $P'(B)$ si :

$$\Big(\text{gen}(G_{\min}) = \text{gen}(G'_{\min})\Big) \textbf{ et } \Big(c = c'\Big).$$

Pour décrire la RS de manière plus détaillée, supposons qu'on applique la procédure RS à un coeur B_{core} venant d'être identifié. La procédure RS avec affiliation appliquée à ce nouveau cœur fonctionne comme suit. Cette procédure maintient entre autres deux éléments au cours des itérations. Le premier est la liste des boules en cours d'exploration L. Cette liste L contient les boules qui ont été atteintes mais pour lesquelles on a pas encore exploré, évalué, leurs voisins. Le second élément est la boule courante B_c, celle qui a été temporairement choisie parmi la liste L pour évaluer ses voisins à explorer. Voici son fonctionnement.

On initialise d'abord L et B_c toutes les deux avec la même boule B_{core}, celle qui vient d'être élue cœur. Ensuite, on itère les mêmes instructions suivantes tant que L n'est pas vide :

En premier lieu, on charge $V := \{V_i\}_i$, l'ensemble des voisins inexplorés de B_c vérifiant les trois conditions primaires de déplacement. Pour chacune des boules V_i, si le chemin courant est plus petit que (ou égal) le chemin de la dernière instance de RS l'ayant atteinte alors on autorise V_i à être exploré. Son label devient celui de B_{core}, son chemin est mis à jour avec le chemin courant et V_i est ajouté à la liste L afin d'évaluer plus tard ses voisins. Dans tous les cas, le nouveau nœud V_i est marqué comme explorée, la boule courante B_c est supprimée de la liste L et la nouvelle boule courante devient le meilleur élément de L, c'est à dire, la boule qui porte le meilleur chemin. Cette procédure est répétée tant qu'il reste des candidats à développer dans

la liste L.

Comme on peut le voir dans les détails algorithmiques donnés dans l'algorithme 9, on a autorisé la procédure RS à affilier les boules qui ont déjà été atteintes par une autre instance avec un chemin équivalent. La raison derrière est que, lorsque la recherche courante tombe sur une boule déjà découverte par un autre cluster avec un chemin égal, ce dernier peut avoir utilisé une boule intermédiaire pour l'atteindre alors qu'il ne la possède plus a posteriori car cette boule intermédiaire a pu être atteinte entre temps par l'instance courant avec un chemin strictement meilleur.

En effet, les premières instances peuvent établir des chemins facilement car les autres instances n'ont pas encore réclamé les boules qu'ils peuvent atteindre avec un meilleur chemin. Une boule B qui a été atteinte de cette manière possède un chemin qu'il ne serait plus en mesure d'effectuer maintenant que des boules de ce chemin ont été réclamée avec un chemin strictement meilleur. Et puisque la force d'un chemin est à mémoire courte (dernière modification de $G_{\min}$ et c) on ne peut pas savoir facilement si ces chemins sont encore valides.

De manière plus formelle, considérons l'instance RS courante I_1 qui vient juste d'atteindre la boule B avec un chemin $P(G_{\min}, c)$. Supposons également que B a été atteinte dans le passé par une autre instance I_2 avec un chemin $P(G'_{\min}, c')$ égal à $P(G_{\min}, c)$ mais que la boule $G'_{\min}$ a été explorée entre temps par l'instance courante I_1 avec un chemin strictement meilleur. Dans ce cas là le chemin $P(G'_{\min}, c')$ emprunté par I_2 pour atteindre B n'est plus valide car $G'_{\min}$ n'appartient plus au cluster associé à I_2. Ainsi l'égalité apparente due au fait que l'évaluation de la force d'un chemin est à mémoire courte est en fait une fausse égalité. L'instance courante I_1 doit donc être prioritaire.

Bien évidement, toutes les égalités observées ne seront pas forcément de fausses égalités, mais dans un premier temps on préfère contraindre à recouvrir les cas d'égalité comme si l'instance courante possédait un chemin strictement meilleur.

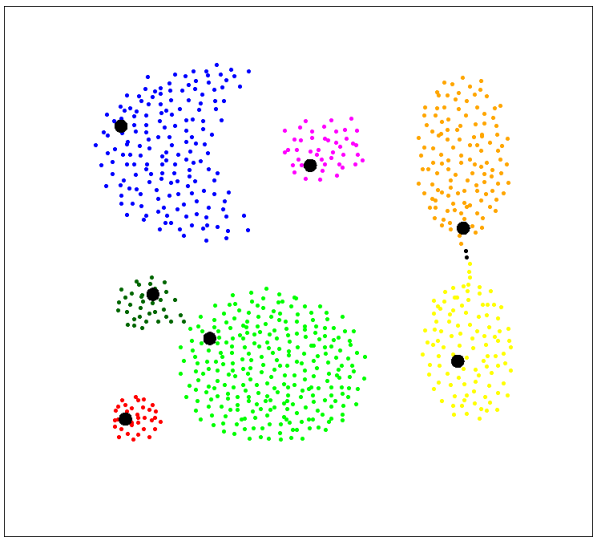
La nuance entre chemin supérieur ou égal et supérieur strict est faite au cours d'un seconde recherche RS pour chacun des cœurs. La différence entre ces deux passes consécutives, c'est que pour la deuxième tous les clusters ont eu l'occasion de réclamer toutes les boules qu'ils pouvaient. Dès lors, tous les prochains cas d'égalité seront forcément vrais. Les boules concernées par ce cas de figure seront quand même explorées

mais seront marquées par un booléen. Les boules marquées seront alors attribuées à aucun cluster à la fin de cette deuxième passe. Comparé à ICMDW, la recherche RS ne demande que très peu de calculs. En effectuer deux reste très raisonnable et n'augmente pas de manière très significative le temps d'exécution total.

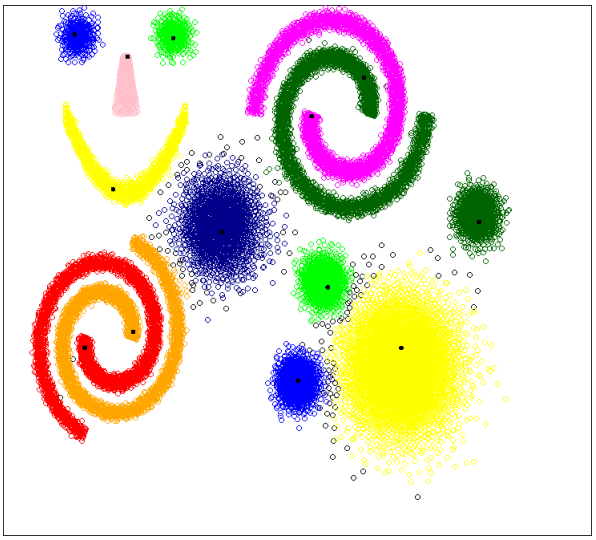
Finalement les points de l'échantillon obtiennent le même label que leur dernière boule mère terminale connue, ou de manière équivalente, le même label la boule terminale de plus haute génération à qui chaque x_i appartient respectivement. Les résultats de clustering obtenus par Ballstering sur les jeux de données Agrégation et Smiley sont illustrés dans la figure 4.7.

Algorithm 9 Reachability search (premier passage)

Initialisation de L et B_c avec le cœur courant
while $L \neq \emptyset$ **do**
  $V \leftarrow$ voisins de B_c vérifiant les trois conditions primaires
  Supprimer B_c de L
  for all $V_i \in V$ **do**
    Retenir que V_i a été exploré
    if $\text{gen}(V_i) < G_c$ **then** ▷ Où G_c provient de $P(B_c) := (G_c, c_c)$
      $\text{path}_c \leftarrow (V_i, 0)$
    else
      $\text{path}_c \leftarrow (G_c, c_c + 1)$
    end if
    if $\text{path}_c \leq \text{path}(V_i)$ **then**
      $\text{path}(V_i) \leftarrow \text{path}_c$
      Affilier V_i au cluster associé au cœur courant.
      Ajouter V_i à L
    end if
  end for
  $B_c \leftarrow$ l'élément de L possédant le plus court chemin
end while



(A) Résultats sur Agrégation ($N_{\min} = 4$)



(B) Résultats sur Smiley ($N_{\min} = 200$)

FIGURE 4.7 – Résultats de Ballstering sur les jeux de données Agrégation et MultiShapes. Les gros points noirs sont les cœurs de clusters et les plus petits sont des observations non-affectées.

Chapitre 5

Paramétrage et Complexité théorique

5.1 Choix des paramètres

Dans cette sous-partie on discute plus en détail des trois paramètres de Ballstering. Deux d'entre eux, le coefficient de décroissance R et le rayon initial d_c^0 sont respectivement fixés et calculés automatiquement et ne sont donc pas à choisir par l'utilisateur. Seul $N_{\min}$ est personnalisable par l'utilisateur et il n'est nécessaire de le choisir que lorsque les clusters sont décrits par peu de données ce qui n'est pas la cible de notre algorithme. On justifie ci-dessous les valeurs des deux premiers paramètres fixés et discutons des différents choix de $N_{\min}$.

5.1.1 Coefficient de décroissance R

Le paramètre $R \in]0, 1[$ contrôle la vitesse de décroissance du rayon. Lorsque R est petit, la taille des boules décroit rapidement ce qui va induire peu d'itérations mais beaucoup de boules créées à chaque itération (Et donc le nombre de distances à calculer par itération). A l'inverse, lorsque R est proche de 1, on obtient peu de boules par itération mais énormément d'itérations.

Concernant la qualité du clustering, R doit être suffisamment petit pour induire une différence de densité significative entre deux boules de génération consécutives. S'il est proche de 1 par exemple on s'expose fortement au risque de détecter de faux pics de plusieurs générations dans des zones où la densité est pourtant uniforme.

Cependant il doit également être suffisamment grand pour pouvoir décrire correctement les relativement faibles variations de densité. En effet, s'il est proche de 0, le rayon des boules chute extrêmement vite entre deux générations consécutives. Une zone où la densité ne varie pas de manière extrême se retrouverait alors décrite par des boules terminales appartenant toutes à la même génération. Toutes les nuances dans la variation de densité ne sauraient être représentées.

Selon notre expérience, fixer R à $\frac{3}{4}$ nous a apparu être un choix robuste qui à la fois permet de bien rendre compte de la densité mais aussi assure une différence de densité significative entre deux générations différentes.

5.1.2 Le rayon initial d_c^0

Le rayon initial est le rayon utilisé lors de la toute première itération de Ballstering. Il doit être suffisamment grand pour couvrir tous les points en quelques boules seulement mais également de taille parcimonieuse afin de ne pas se retrouver avec plusieurs itérations inutiles dans lesquelles une seule boule englobant tout le domaine est créée à chaque fois.

N'étant pour autant pas un paramètre crucial, il peut être choisi grâce à l'heuristique suivante. K paires de points (x_i, x_j) ($K << N$) sont choisis aléatoirement et les valeurs absolues des différences dans chaque dimensions $a_{i,j}^m = |x_i[m] - x_j[m]|$ sont calculées. On retient la différence la plus grande et on la divise par trois pour obtenir d_c^0.

Quand on est en présence de pôles très denses et très éloignés les un des autres ou que du bruit très éloigné de la plupart des points est tiré parmi les K points pour déterminer d_c^0, ce dernier se retrouve initialisé avec une très grande valeur. d_c^0 qui sera alors suffisamment grand pour englober les pôles en quelques boules devra être réduit à travers un nombre potentiellement grand d'itérations avant d'arriver à une taille discriminante, qui reflète bien la densité locale. Même si dans ce cas là peu de calculs sont nécessaires à chaque itérations car peu de boules sont créées, la relecture successive de ces données peut induire en temps perdu non négligeable surtout lorsque qu'elles sont très volumineuses.

Le choix du rayon initial pourrait donc être amélioré en repérant au préalable si des zones denses sont isolées par exemple. Ceci dit, et bien qu'une amélioration de ce

point de vue serait appréciable, cette limitation reste tout de même marginale.

$N_{\mathbf{min}}$

Souvenons nous que $N_{\mathbf{min}}$ était le paramètre définissant la terminalité d'une boule ou au contraire de sa décomposition en de plus petites.

Un $N_{\mathbf{min}}$ trop faible risque d'induire des sauts de générations significatifs dû uniquement aux concentrations aléatoires de points. Avec un paramètre $N_{\mathbf{min}}$ trop petit, l'algorithme sera très sensible au bruit et risque donc de détecter des faux cœurs de clusters. Au contraire, attribuer une grande valeur à $N_{\mathbf{min}}$ rendrait l'algorithme plus robuste face aux concentrations aléatoires mais diminuerait le grain de précision de la carte de densité. En effet, un $N_{\mathbf{min}}$ élevé implique qu'on stoppe la décomposition des boules alors qu'il reste beaucoup de points à l'intérieur qui pourraient décrire une densité aux formes plus complexes, qu'on ne retrouvera pas si on ne décompose pas plus.

Tout comme bon nombre d'algorithmes basés sur la densité, notre algorithme n'est pas fait pour l'étude de petit jeux de données ni pour identifier des clusters contenant peu de points. On recommande fortement de choisir un $N_{\mathbf{min}} > 50$ et la valeur basique $N_{\mathbf{min}} = 100$ marche sans problèmes : aucun faux cœur ne peut être détecté et dès qu'on s'intéresse à des clusters décrits par un nombre points relativement conséquent (de l'ordre du millier en basse dimension) l'algorithme discriminera bien deux clusters distincts. Il a pu fonctionner correctement sur le l'exemple illustrant Agrégation avec une si faible valeur $N_{\mathbf{min}} = 4$ grâce à l'homogénéité artificielle du jeu de données. Ballstering ne devrait ni être appliqué à un si petit jeu de données ni avec un $N_{\mathbf{min}}$ si faible.

Afin de soutenir le choix de $N_{\mathbf{min}} > 50$ nous avons mené des expériences sur données artificielles. Nous avons généré trois échantillons consistant en une même loi uniforme contenant respectivement 10 000, 20 000 et 100 000 points. Nous avons appliqué Ballstering sur chacun de ces jeux de données 40 fois en faisant varier $N_{\mathbf{min}} \in \{50, 49, 48, ..., 10\}$. Pour chaque $N_{\mathbf{min}}$ nous avons calculé le nombre de fois sur les 30 essais où le clustering final n'était pas exact, c'est à dire quand deux cœurs clusters ou plus ont été détectés.

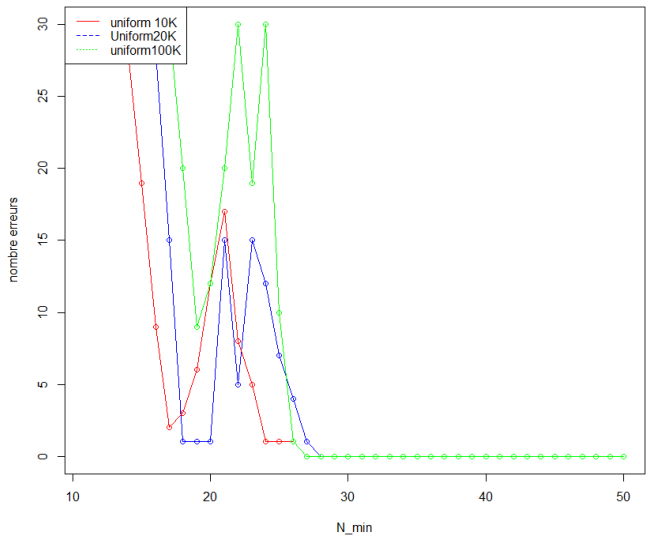


FIGURE 5.1 – Évolution en fonction de $N_{\min}$ (décroissant) du nombre d'échecs de clustering (sur 30 essais) pour trois échantillons répartis selon la même loi uniforme mais dont le nombre d'observations est 10 000, 20 000 et 100 000.

Les résultats qui sont résumés dans le graphique figure 5.1 indiquent que le risque d'erreur est nul pour ce type de lois et pour les tailles d'échantillon considérées tant que $N_{\mathbf{min}}$ est grand et devient sujet aux détections de faux cœurs lorsque $N_{\mathbf{min}}$ approche de la valeur 30. Plus exactement la première erreur est détectée pour $N_{\mathbf{min}} = 27$ lorsque appliqué à l'échantillon de 20 000 observations et $N_{\mathbf{min}} = 28$ pour le plus petit et le plus grand échantillon. Le risque d'erreur intervient donc au même seuil peu importe le nombre d'observations N. Seule la proportion d'échecs croît avec N et cela s'explique en considérant les grands jeux de données comme une concaténation des plus petits : si chaque sous domaine possède une probabilité p d'échouer alors la probabilité que la concaténation de ces sous-domaines échoue correspond à la probabilité qu'au moins un de ces sous-domaines échoue qui est supérieure à p. Pour les valeurs précises des expérimentations voir tableaux en annexe, tableau B.

Des travaux similaires mais appliqués à des données distribuées selon une loi normale (selon plusieurs variances σ^2 différentes) sont également présentés et montrent la même tendance avec une première erreur observée pour $N_{\mathbf{min}} = 24$ sur la Gaussienne de paramètre σ le plus faible.

5.2 Complexité de Ballstering

Chaque itération de ICDMW passe en revue chaque observation non-gelée et applique une série d'instructions qu'on appellera pour des raisons pratiques *proc*1 et dont on évaluera la complexité plus loin. Le nombre de points à qui on applique cette procédure diminue donc en fonction du nombre d'observations gelées. Cependant puisque le nombre d'observations gelées dépend directement de la répartition des données, on se placera dans le pire cas où toutes les observations sont gelées à la toute dernière itération ce qui correspond à N applications de *proc*1 par itération.

Tâchons maintenant d'évaluer la complexité de *proc*1 qui, pour une observation x_i, crée une boule si x_i n'appartient à aucune boule de la génération courante et incrémente les comptes de densités dans le cas contraire. La première opération de *proc*1 est de calculer la distance aux sœurs de la mère de x_i qui est donc proportionnel au nombre de sœurs. Pour évaluer le nombre de sœurs on se place dans le pire cas où toute boule possède un nombre fixe x de boules filles qui correspond au maximum de boules filles que peut avoir une boule mère.

Ce problème de recouvrement est purement géométrique et ne dépend donc pas du nombre d'observations N. En revanche le nombre de sœurs x dépend de la dimension d. Le calcul exact de x est délicat, mais il est facile d'en avoir une bonne approximation en imposant que la somme des volumes des boules filles soit supérieur ou égal au volume de la boule mère étendue à son rattachement ($r' = r(1 + R)$) corrigé par un facteur multiplicatif γ pour tenir compte des recouvrements entre sœurs. Plaçons nous à une génération arbitraire g et considérons une boule mère $B^{g-1}(c, r)$ de cette génération ainsi que V'_m, le volume de la boule $B(c, r(1 + R))$ de même centre que $B^{g-1}(c, r)$ mais rayon étendu à son rattachement, qui correspond donc à l'espace où les boules filles B^g_i rattachées à la boule mère $B^{g-1}(c, r)$ peuvent prendre leur centre. Dans ce cas on a :

$$\sum^{n} V_i \geq \gamma V'_m, \tag{5.1}$$

où V_i est le volume de la boule fille B^g_i. Or le volume d'une d-sphère de rayon r est égal à $K(d)(r^d)$, où K est une fonction de d dont l'expression n'est pas importante

ici. Ainsi,

$$\sum^{x} V_i \geq \gamma V'_m \tag{5.2}$$

$$\Rightarrow K(d)(r.R)^d \geq \gamma K(d) r^d \tag{5.3}$$

$$\Rightarrow xR^d \geq \gamma \tag{5.4}$$

$$\Rightarrow x \geq \frac{\gamma}{R^d} \tag{5.5}$$

$$\tag{5.6}$$

Le nombre de boules filles maximum avec lesquelles il faut calculer la distance est donc lié exponentiellement à d.

Une fois la distance aux sœurs calculées, deux cas sont possibles. Soit on incrémente les densités des sœurs à qui appartient x_i auquel cas seules quelques additions élémentaires sont induites, soit on crée une nouvelle boule qui elle induit des calculs supplémentaires pour savoir à quelles boules mère cette dernière est rattachée, ce qui induit des opérations du même ordre que ce que nous venons de développer.

La complexité de l'algorithme complet s'obtient à partir des résultats précédents en évaluant le nombre de niveaux nécessaires dans l'arbre de partitionnement en fonction de la taille N de l'échantillon, donc le nombre d'itérations de la procédure CM (noté n).

Afin de déterminer ce nombre d'itérations n et pour des raisons pratiques, nous supposons que la distribution sous-jacente est uniforme et que toutes les boules terminales se situent au plus bas de la hiérarchie. Dans ces conditions l'algorithme se termine lorsque le nombre de points présents dans une boule est inférieur à une valeur seuil notée P. Toujours sous l'hypothèse d'uniformité, ceci se traduit par une

contrainte sur les volumes des boules terminales. En admettant que les boules terminales sont contenues dans le support de la distribution sous-jacente on a :

$$\frac{P}{N} \propto r_{\min}^{d} \tag{5.7}$$

$$\Rightarrow r_{\min} \propto (\frac{P}{N})^{\frac{1}{d}} \tag{5.8}$$

$$\Rightarrow d_c^0 R^{\text{nb}} \propto (\frac{P}{N})^{\frac{1}{d}} \tag{5.9}$$

$$\Rightarrow nb \log(R) + \log(d_c^0) \propto \frac{1}{d}(\log(P) - \log(N)) \tag{5.10}$$

en supprimant les constantes on a : (5.11)

$$nb \log(R) \propto -\frac{1}{d} \log(N) \tag{5.12}$$

$$\Rightarrow nb \propto \frac{1}{d} \frac{\log(N)}{\log(1/R)} \tag{5.13}$$

Ce qui induit une complexité générale de ICMDW de

$$O(\frac{\frac{1}{d}\log(N)N}{\log(\frac{1}{R})}\frac{1}{R^d})$$

La complexité en terme de nombre d'observations N est donc de $O(N \log(N))$. En revanche ce que nous venons d'établir implique que le nombre d'itérations décroit quand d augmente. Ceci se traduit par une compression de la hiérarchie lorsque d augmente, empêchant l'algorithme d'être discriminant, de détecter correctement les cœurs. Il apparaît donc nécessaire de faire dépendre le paramètre R de la dimension pour compenser ce phénomène, mais l'algorithme est déjà limité par la dimension d car le nombre de filles par boule mère augmente de manière géométrique avec d.

La phase partitionnement de Ballstering consiste en deux séries de procédures RS (une RS par cœur) successives. Supposons que le jeu de données considéré contienne k clusters où k est fixé et $k << N$. Puisque deux RS doivent être appliquées par cœur, il y en aura $2k$ au total.

Intéressons nous donc à la complexité de la procédure RS. D'après nos observations, une grande partie des boules ne sont généralement explorées que par un quelques cœurs seulement, voir souvent un seul. Cependant puisque le nombre de

FIGURE 5.2 – Illustration des trois répartitions différentes sur lesquelles on applique Ballstering afin de voir l'impact de la répartition des données sur le temps d'exécution. De haut en bas : Cassini, uniforme et spirale.

boules couvertes par chaque RS est variant et difficile à contrôler, nous considérons le pire cas hautement improbable où chaque boule (mis à part les cœurs) est couverte par toutes les procédures RS. Dans ce cas, chaque procédure RS induit pour les $N_B - k$ (où N_B est le nombre de boules total) boules couvertes de simples opérations élémentaires ne dépendant pas de N ni de d à savoir le calcul de la force du chemin et potentiellement remplacement du chemin et affiliation au cluster.

la complexité de la phase de partitionnement de Ballstering se mesure donc par rapport au nombre de boules formant l'arbre et est donc équivalente à celle de ICMDW, donc $O(N \log(N))$.

Nous avons étudié théoriquement la complexité, en ayant recours au scénario du pire cas. Pour avoir un indicateur de la complexité en moyenne, nous avons généré un ensemble de jeux de données bidimensionnels qui varient selon la taille (nombre d'observations) et la répartition sous-jacente. Ces jeux de données sont issus de trois types de répartitions. On y dénote la loi uniforme, une densité de forme de Cassini partielle et une spirale à deux bras dont la particularité est de concentrer sa densité sur des bras relativement fins (figure 5.2).

Chacun de ces trois archétypes ont été générés 7 fois dans des fichiers différents en augmentant le nombre d'observations à chaque simulation. Les plus petits échantillons

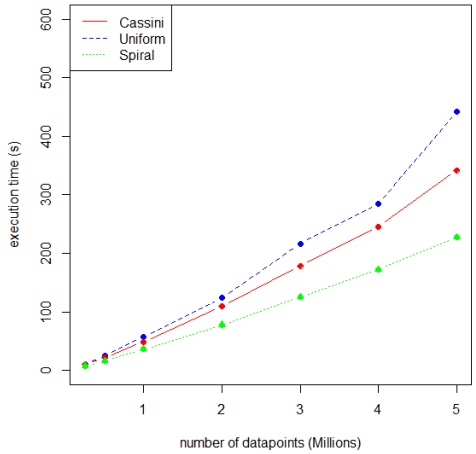


FIGURE 5.3 – Temps d'exécution obtenus par Ballstering sur les trois répartitions différentes (Cassini, spirale et uniforme), avec nombre croissant d'observations.

contiennent 250 000 points et les plus grands jusqu'à 5 millions de points maximum. Nous avons obtenus les temps de calculs pour chacun des 21 jeux de données en exécutant Ballstering avec comme paramètre $N_{\min}$ égal à 100. Les résultats obtenus sont résumés dans la figure 5.3.

Ce qui ressort de cette expérience est d'une part que la répartition des données impacte très fortement le temps d'exécution. En effet ce dernier varie presque du simple au double lorsqu'on passe de la répartition uniforme à la spirale (pour 5 millions de points). Bien que le cas de la spirale induise plus d'itérations que le cas uniforme, une grande partie des observations sont gelées ce qui induit moins de calculs par itération, ce qui explique la différence de temps.

A l'échelle de l'exemple considéré l'algorithme Ballstering montre une tendance presque linéaire vis à vis du nombre d'observations N ce qui est proche de l'idéal.

Chapitre 6

Résultats empiriques

6.1 Résultats expérimentaux sur données simulées

Dans cette section nous allons évaluer la qualité et la complexité temporelle de notre algorithme Ballstering notamment en le comparant à deux autres algorithmes. Nous avons choisi de le comparer à k-means [36] pour se rapidité et HDBSCAN [15] pour ses performances appréciables mais surtout pour sa qualité de clustering.

Fondé sur l'utilisation des k plus proches voisins, HDBSCAN présente une très bonne qualité de clustering, peu de paramètres à choisir et une complexité finale réduite à $O(Nlog(N))$ grâce à l'utilisation de $kd-$tree [4], un algorithme auxiliaire qui partitionne récursivement le domaine des observations pour mieux localiser les données, ce qui permet de récupérer rapidement les k plus proches voisins.

Ces algorithmes sont en open source et disponibles grâce à Scikit-learn [42] qui supporte une collection d'algorithmes de machine learning développés en python, permettant un accès facile à de puissants outils pour l'analyse de donnée. Il est important de noter que les deux algorithmes de comparaison ont été optimisés et parallélisés alors que notre version de Ballstering n'est qu'à l'état de prototype et non parallélisé.

Comparé à HDBSCAN, la version disponible de FDPC est très lente, consomme une quantité de mémoire conséquente, demande de choisir un paramètre non trivial et pourtant crucial pour au final fournir une qualité de clustering très moyenne, bien inférieure à celle de HDBSCAN. A cela vient s'ajouter simple fait que la version disponible de FDPC ne peut pas être utilisée sur les jeux de données de cette section car trop volumineux pour lui et notre ordinateur.

6.1.1 Évaluation de la qualité

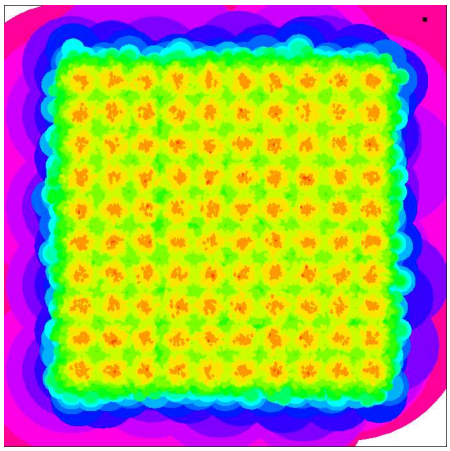
Pour démontrer la robustesse de Ballstering et son habileté à gérer les structures complexes, la qualité de clustering sera évaluée sur trois jeux de données synthétiques présentant une difficulté croissante.

Les deux premiers, Birch1 et Birch3 ont été récupérés sur [17], un répertoire qui héberge une collection de jeux de données qui ont tous été utilisés dans des articles traitant de clustering ou de classification. Nous avons simulé le dernier jeu de données qui s'appelle Multishapes.

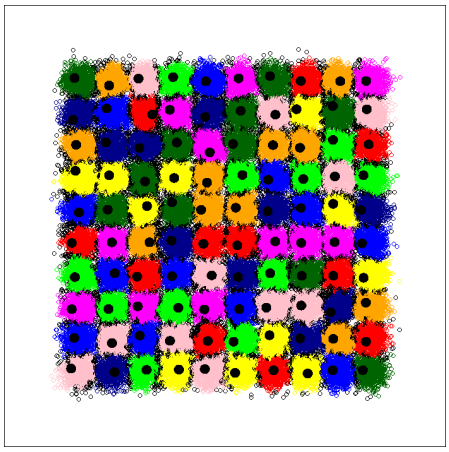
Concernant le choix des paramètres, nous avons toujours choisi le k de k-means de telle manière qu'il soit égal au nombre de groupes contenus dans le jeu de données correspondant.

Pour HDBSCAN, le choix est un peu plus difficile à effectuer. Ce dernier algorithme possède deux paramètres principaux, min_cs qui définit le nombre minimum de points que doit contenir chaque cluster pour être considéré en tant que tel, et min_s qui lui contrôle la détection du bruit. Si min_s est grand alors une grande proportion de points sera considérée comme du bruit. Puisqu'on a observé sa tendance à attribuer trop de points à du bruit, ce dernier paramètre sera dans ces trois exemples toujours fixé à son minimum. Pour choisir min_cs nous avons essaie plusieurs valeurs et choisi le meilleur pour chacun des exemples.

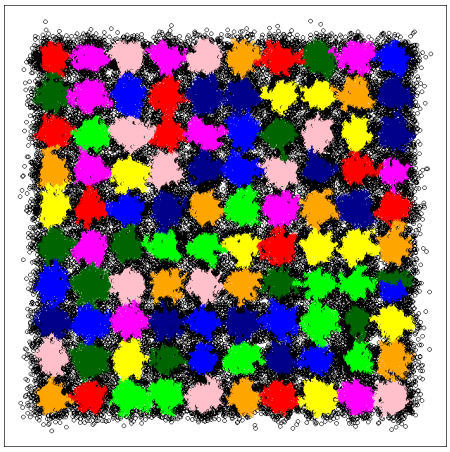
Afin d'avoir un aperçu de la répartition des points, chaque exemple sera illustré par la carte de densité générationnelle formée par la collection de boules établie par Ballstering. Nous sommes conscients que nos résultats de clustering sont directement liés à notre carte de densité. C'est donc pourquoi nous allons nous appuyer sur ces cartes pour critiquer et discriminer les algorithmes uniquement dans des situations évidentes. Les situations plus ambiguës pourront être commentées avec la carte de densité mais avec une grande prudence.



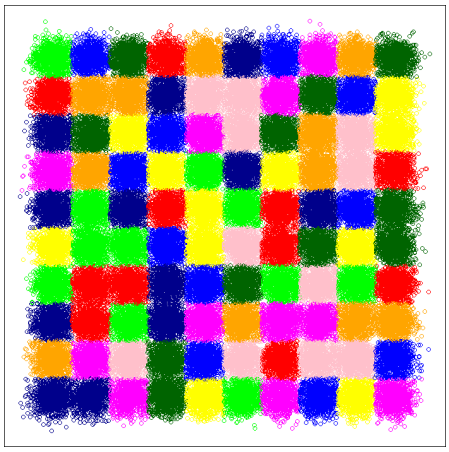
(A) Ballstering : Densité générationnelle



(B) Ballstering : Clustering



(C) HDBSCAN



(D) $k-$means

FIGURE 6.1 – Résultats de Ballstering ($N_{\min} = 15$), HDBSCAN (min_cs= 500,min_s= 1) et $k-$means ($k = 100$) sur le jeu de données Birch1.

Birch1

Birch1 est composé de 100 Gaussiennes qui partagent la même variance et le même nombre de points (1000 points chacun), répartis sur une grille de 10×10. Sa difficulté réside en la proximité des 100 clusters qui se superposent fortement entre voisins. Les résultats sont illustrés dans la figure 6.1 où les points noirs sont les cœurs trouvés par Ballstering.

HDBSCAN ainsi que Ballstering ont trouvé le bon nombre de clusters mais HDBSCAN a séparé en deux parties la Gaussienne localisée à la la ligne 7 et colonne 10 et dans le même temps a fusionné deux autres situées en $(2, 4)$ et $(3, 4)$. De plus, et bien que min_s ait été fixé à son minimum, HDBSCAN s'est montré beaucoup trop conservateur en attribuant beaucoup de points à du bruit. Aussi, certains des clusters qu'elle a restitué sont un peu trop étendus et vont mordre sur les Gaussiennes avoisinantes. k-means a présenté de plutôt bon résultats sur cet exemple. Le seul reproche qu'on peut réellement lui faire est la brutalité de son clustering, avec ses clusters très rectilignes ne reflétant pas bien les Gaussiennes. Ballstering ne souffre d'aucun des problèmes cités ci-dessus. Chacun des clusters a été correctement détecté, avec un rendu Gaussien et la non affiliation de certains points est parcimonieuse et adaptée.

Birch3

Birch3 est lui aussi composé de 100 Gaussiennes contenant 1000 points chacune mais cette fois-ci de variance différentes et placées aléatoirement. Il est difficile d'apprécier précisément la qualité de clustering sur ce jeu de données, mais il a l'avantage de mettre en avant des comportements intéressants pour chacun des algorithmes comparés.

Concernant HDBSCAN, le problème d'attribution au bruit non pertinent est encore plus flagrant sur cet exemple avec parfois de larges zones très denses qui sont considérées comme du bruit. De plus, il ne semble pas être sensible aux variation de densités progressives mais pourtant très claires.

C'est le cas par exemple sur la figure 6.2 où les clusters situés en (1) et les trois Gaussiennes en (2) sont fusionnés à tort. C'est également le cas en (3) et (4) mais Ballstering n'a pas réussi non plus. Ce dernier point est d'ailleurs un cas limite de

Ballstering : Bien que la densité estimée par notre algorithme laisse apparaître distinctement deux poles, les deux clusters sont considérés comme un seul par Ballstering à cause de l'absorption des variations d'une génération. Cependant, si nous avions un échantillon un peu plus fournit, la carte de densité obtenue aurait été plus précise avec plus de générations intermédiaires crées, ce qui discriminerait les clusters qui ne le sont pas dans les conditions actuelles.

Également certains clusters qui ont été fusionnés à tort par Ballstering on bien été détectés par HDBSCAN comme en (5) et (6), correspondant à (A) et (B) pour Ballstering. Ceci est probablement dû à leur paramètre k (fixé automatiquement à 4), bien inférieur à $N_{\min}$ (égal à 15), qui permet donc de détecter de plus petites variations de densité que notre algorithme.

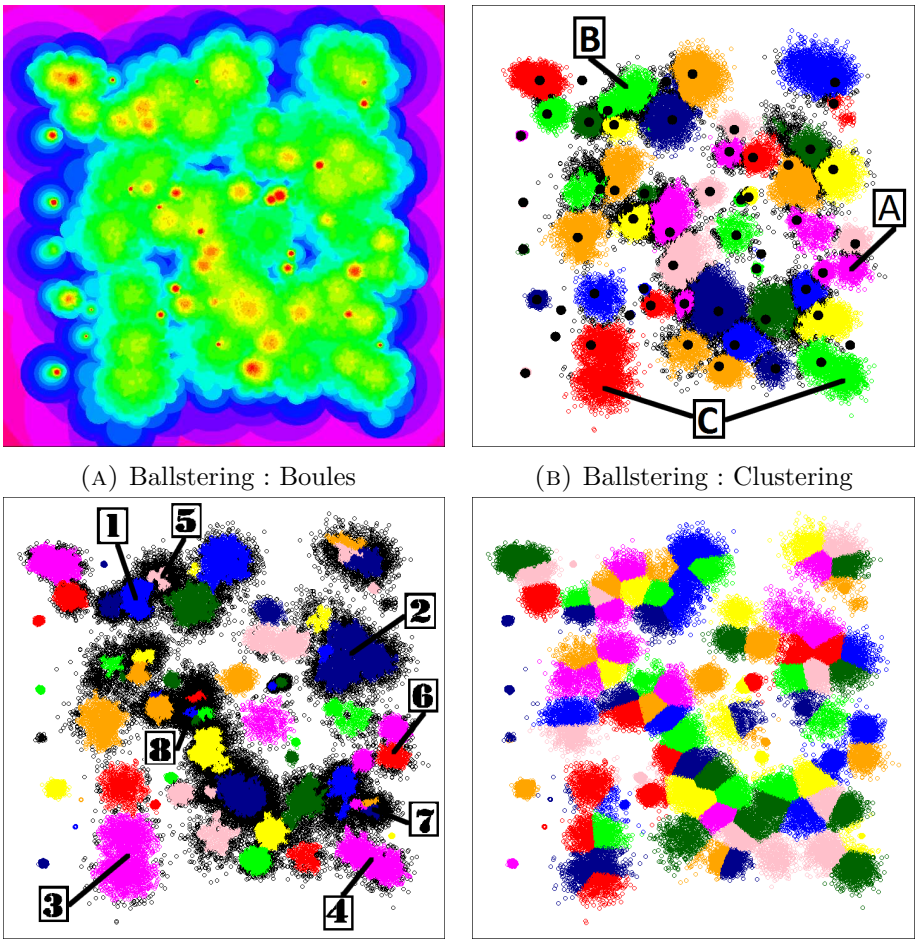
Malheureusement pour HDBSCAN, bien que la présence des deux cluster soit bien déterminée, cet avantage est gâché car une très large partie de ces clusters se retrouve considérée encore comme du bruit.

k-means souffre toujours d'un clustering brutal basé sur la distance et non sur la densité, et force beaucoup de séparations non appropriés.

Multishapes

MultiShapes est un jeu de données bidimensionnel que nous avons créé contenant 3 360 000 points distribués sur 32 clusters de taille et de forme variées. On peut y trouver des clusters concentriques (les spirales) ou plats (les carrés), tous ces clusters formant un ensemble de groupes de densité très variées. Certains d'entre eux sont proches, voir superposés l'un sur l'autre et nous avons en plus ajouté un bruit homogène sur tout le domaine.

Comme on peut le voir dans la figure 6.3, HDBSCAN a obtenu des résultats pertinents mis à part encore une fois sur la détection du bruit qui n'est pas assez conservateur notamment pour les Gaussiennes. Aussi la Gaussienne au centre et un bras de la spirale à sa droite ont été détectés comme un seul cluster. Cela confirme la tendance qu'a HDBSCAN à ne pas discriminer deux clusters qui sont pourtant reliés par une dépression de densité mais progressive. On peut cependant admettre que le vrai bruit est très bien détecté.



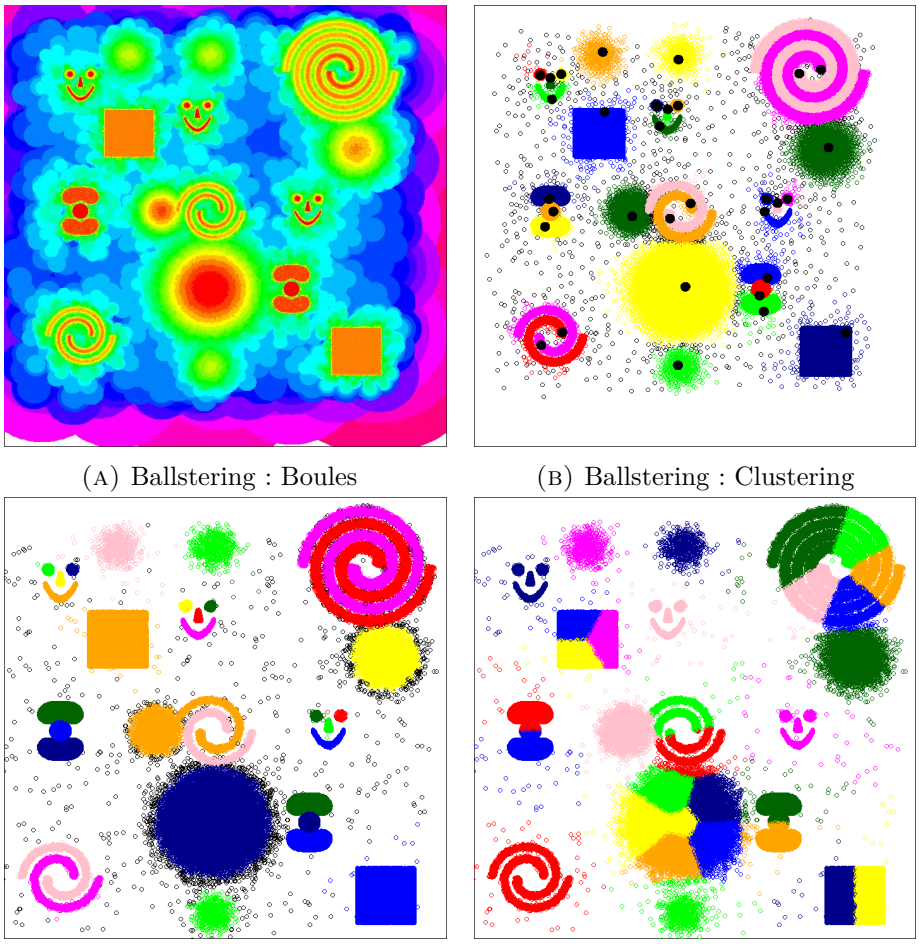
(A) Ballstering : Boules

(B) Ballstering : Clustering

(C) HDBSCAN

(D) $k-$means

FIGURE 6.2 – Résultats de clustering de Ballstering ($N_{\min} = 15$), HDBSCAN (min_cs= 150,min_s= 1) et $k-$means ($k = 100$) sur le jeu de données Birch3.



(A) Ballstering : Boules (B) Ballstering : Clustering

(C) HDBSCAN (D) $k-$means

FIGURE 6.3 – Résultats de clustering de Ballstering ($N_{\min} = 100$), HDBSCAN (min_cs= 1000,min_s= 1) et $k-$means ($k = 32$) sur le jeu de données Multishapes.

Comme on l'attendait, $k-$means a montré une qualité de clustering très faible sur les clusters non-convexes que sont les spirales. De plus il a estimé que chaque smiley ne sont qu'un seul cluster alors qu'ils sont clairement composés de 4 clusters. En parallèle il a divisé les plus grands clusters comme la gaussienne et les uniformes en plusieurs plus petit clusters pour pouvoir atteindre les 32 clusters demandés en paramètre.

Puisque tous les clusters possèdent un nombre de points conséquent, on a pu exécuter Ballstering avec comme paramètre initialisé avec la valeur basique $N_{\min} = 100$. Il a correctement détecté les 32 clusters, la détection du bruit est quasiment équivalente à celle d'HDBSCAN mais notre algorithme a mieux agrégé le contour des Gaussiennes.

Évaluation via critères externes

Afin de se faire une idée plus précise de la qualité du clustering de notre algorithme, nous l'avons appliqué à un ensemble de 10 échantillons artificiels et évalué son partitionnement à l'aide de critères de qualité externes. Ces 10 jeux de données sont semblables à birch3, évoqué précédemment. En effet, ils contiennent chacun 20 clusters qui suivent une loi normale bidimensionnelle $N(\mu, \sigma)$ dont la moyenne μ est choisie uniformément sur le carré unité et la variance uniformément sur $[0.001, 0.05]$.

Nous avons donc appliqué Ballstering avec $N_{\min} = 50$ sur ces 10 jeux de données et calculé à chaque fois la valeur des dix critères de qualité puis moyenné les valeurs obtenues.

Ces critères ne peuvent être évalués si le partitionnement contient des points non-attribués. Nous avons décider d'attribuer à posteriori les non-classés de Ballstering et HDBSCAN au cluster dont le barycentre est le plus proche.

Dans la figure 6.4 nous pouvons voir les valeurs moyennes de 10 critères externes calculés en fonction des partitionnements proposés par les trois algorithmes Ballstering, HDBSCAN ($\min_c s = 150$, $\min_s = 1$) et k-means (k=20) ainsi que pour une partition aléatoire. Ces critères qui ont été calculés grâce au package R "ClusterCrit" [12] numérotés de 1 à 10 et sont nommés respectivement "czekanowski dice", "folkes mallows", "jaccard" "kulczynski", "precision", "recall", "rogers tanimoto", "russel rao", "sokal sneath1" et "sokal sneath2". Ils ont tous la particularité d'être inférieurs à 1 et

de prendre cette valeur lorsque le clustering est parfait.

Sur ce graphique k-means et Ballstering ont des résultats très roche, l'un dépassant l'autre selon le critère. En revanche HDBSCAN montre des réponses aux critères bien inférieurs.

En parallèle il convient de remarquer que Ballstering et HDBSCAN souffrent de leur brutale affiliation de ce qu'ils n'ont pas classé ce qui se traduit par une baisse de la valeur des critères. C'est d'ailleurs sûrement à cause de cela que la qualité de HDBSCAN s'effondre, car comme nous l'avons vu cet algorithme a tendance à attribuer au bruit beaucoup trop d'observations en cas de superposition. Pour cette raison les résultats obtenus par HDBSCAN dans cette expérience ne sont pas représentatifs.

Cependant notre algorithme lui arrive, malgré l'affiliation brutale, à rester compétitif avec k-means qui est pourtant dans des conditions optimales : les clusters sont gaussiens et le nombre de clusters est donné.

6.2 Évaluation de la complexité et scaling

On se concentre maintenant sur les comportement de ces trois algorithmes lorsqu'on passe à grand échelle. Pour ce faire nous avons généré six jeux de données, en $3d$ cette fois, partageant la même répartition mais qui diffèrent sur le nombre d'observations. Chaque jeu contient 32 normales $3d$ contenant le même nombre de points et même variance, distribuées selon une grille $4 \times 4 \times 2$. Le premier jeu de données a été construit avec des Gaussiennes possédant $n_i = 20000$ points chacune (640 000 au total). Le second double son nombre de points avec $n_i = 40000$ et ainsi de suite, le jeu de données suivant doublant le nombre de points du précédent. Le $6^{\text{ème}}$, le dernier et plus gros jeu de données, contient 20.480 millions d'observations. Les résultats on été obtenus avec un ordinateur 64 bits équipé d'un processeur Intel 5-4690K CPU (3.50GHz, quad core) et qui possède 16Gb de RAM.

On a lancé Ballstering avec la valeur de base pour le seul paramètre $N_{\min} = 100$, HDBSCAN avec (min_cs=150, min_s=1) et on a aidé en paramétrant k-means avec le nombre exact de clusters $k = 32$. Tous les trois ont eu de très bon résultats avec un léger point négatif pour HDBSCAN qui trouve encore une fois un peu trop de bruit.

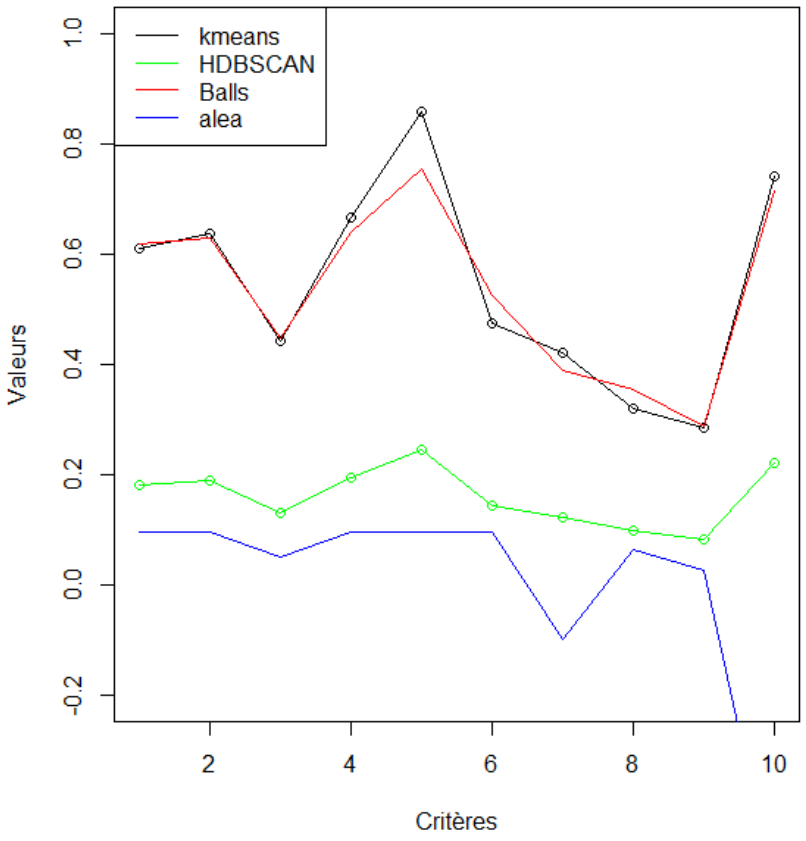


FIGURE 6.4 – Valeurs moyennes de 10 critères externes calculés en fonction des partitionnements proposés par les trois algorithmes Ballstering, HDBSCAN et k-means ainsi que pour une partition aléatoire.

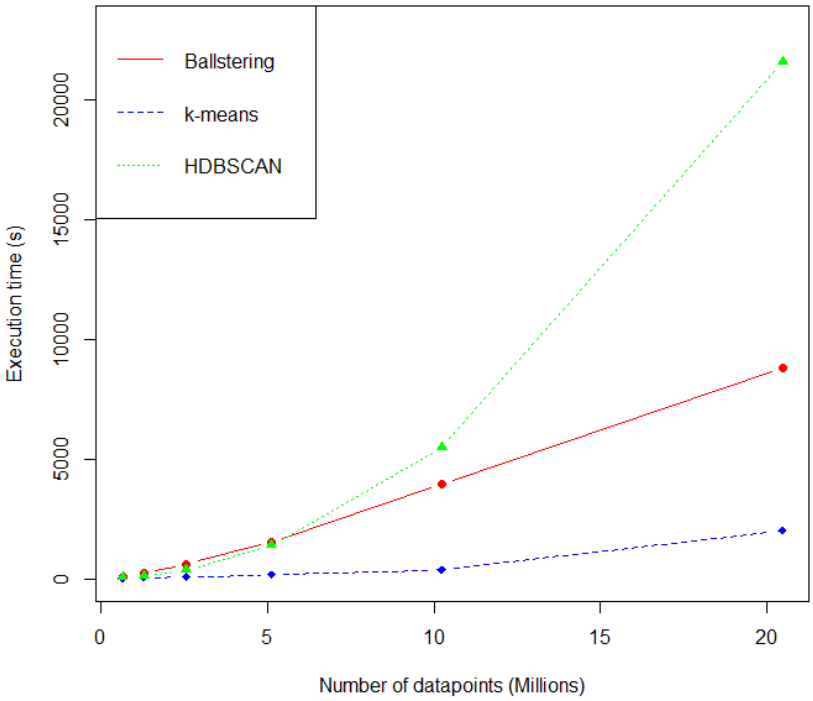


FIGURE 6.5 – Temps d'exécution de Ballstering, HDBSCAN et $k-$means lorsque le nombre d'observation augmente.

Les relevés de temps d'exécution pour les trois algorithmes et pour les six jeux de données sont illustrés dans le graphique de la figure 6.5.

En premier lieu nous devons parler de l'anomalie qui apparaît pour k-means sur le dernier jeu de données. Elle est due au fait que notre ordinateur a atteint la capacité maximum de la RAM et a du utiliser la mémoire SWAP, ce qui a pour conséquence de ralentir l'algorithme de manière conséquente. Ainsi, le temps d'exécution qu'on aurait du observer pour le sixième cas devrait être trois fois plus petit. Ceci dit cette anomalie met en évidence que sa consommation de mémoire est importante, du moins plus que ses deux concurrents.

Puisqu'on compare un prototype avec deux algorithmes à la fois parallélisés et optimisés, c'est sans aucune surprise que notre algorithme se révèle plus lent que les deux autres sur le premier jeu de données. En revanche il fait preuve d'un scaling asymptotique linéaire pendant que HDBSCAN montre une tendance quadratique ce qui mène Ballstering à être plus rapide que HDBSCAN à partir du quatrième jeu de données et plus de deux fois plus rapide sur le dernier.

Ballstering reste toujours bien plus lent que k-means pour tous les jeux de données. Cependant, k-means et Ballstering doublent tous les deux leur temps d'exécution lorsque le nombre d'observations double. Ainsi, l'écart de temps d'exécution peut être grandement réduit une fois notre algorithme optimisé et parallélisé. En outre la complexité de $k-$means (de l'ordre de $O(kNT)$) dépend linéairement de son paramètre k et du nombre d'itérations T qui est indirectement impacté par k. Ainsi une augmentation du nombre de clusters aggraverait significativement le temps d'exécution de $k-$mean tandis que cela n'aurait aucun impact pour Ballstering. L'écart de temps d'exécution serait ainsi bien plus réduit.

6.3 Application aux relevés du trafic aérien français

Nous allons décrire les résultats de clustering obtenus par Ballstering sur les données de trafic aérien français dans sa représentation quadri-dimensionnelle obtenue via le pré-traitement décrit dans la section 2.2.

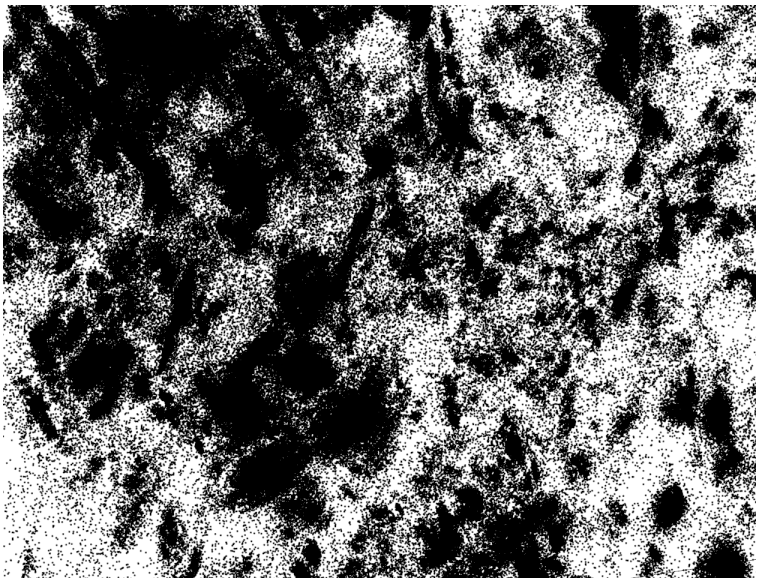


FIGURE 6.6 – Capture d'écran des données issues du trafic français dans leur représentation 3D engendrée par les trois premières composantes principales.

Le jeu de données considéré possède donc 877 705 trajectoires, chacune représentée par un unique point en quatre dimensions. La figure 6.6 permet d'apprécier graphiquement la répartition des trajectoires dans l'espace tri-dimensionnel engendré par les trois premières composantes principales. Les trois autres représentations 3D par combinaison des axes principaux sont donnés en annexe A. Dans cette représentation de très nombreux amas gaussiens de variance et de densité variées se démarquent visuellement. Certains sont très allongés, beaucoup sont en situation de superposition et un nuage de points bruités constelle l'ensemble du domaine.

Nous avons appliqué notre algorithme Ballstering avec le paramètre $N_{\min}$ égal à 50 et avons obtenu les résultats en minutes et 16 secondes. Parmi les 780 groupes qui ont été identifiés, les plus volumineux possèdent jusqu'à 16 287 observations et le moins dense possède 60 éléments. 135 628 objets n'ont été classés dans aucun groupe en raison des fortes superpositions.

La figure 6.7 illustre non pas les données mais le centre de chaque boule terminale colorées selon le cluster auquel elles ont été affiliées. Cette représentation permet de gagner un peu en clarté puisque les points sont moins nombreux en zone peu dense. Ainsi es clusters ressortent donc mieux et on accentue cet effet n'affichant pas les boules non-classées.

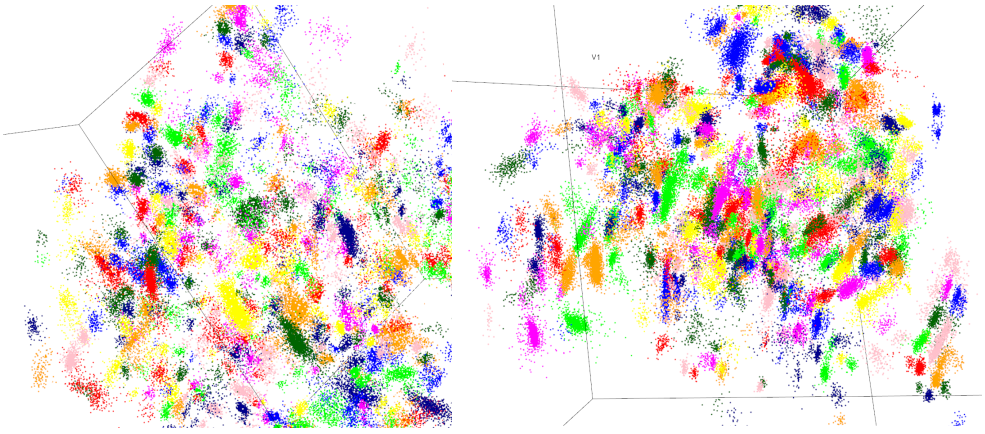


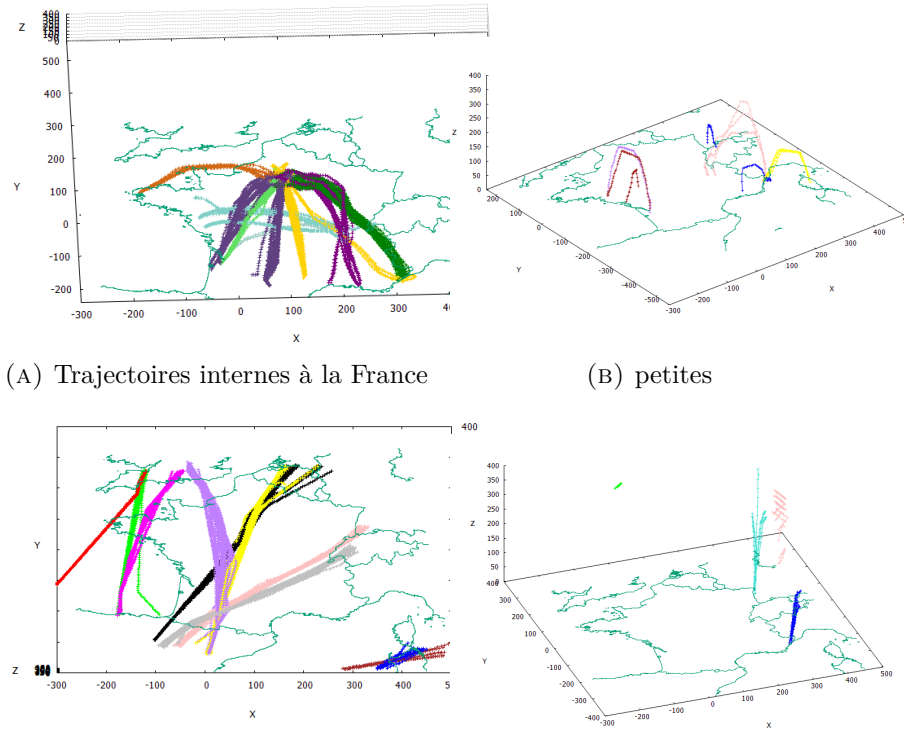
FIGURE 6.7 – Résultats de clustering des trajectoires d'avions dans la représentation 3D engendrée par les trois premières composantes principales, selon deux angles différents.

Puisque on décrit un espace 3D à travers des captures d'écran 2D, on ne peut parler que des clusters en périphérie ou les clusters denses parmi ceux au premier plan. Les amas gaussiens qui sont mis en évidence grâce à leur couleur différente laissent penser que petits et gros clusters, de variances différentes sont bien détectés. Certains clusters semblent mêlés à d'autres mais ce n'est qu'une question de perspective, l'un étant à une position différente de l'autre dans la troisième dimension.

Bien que les résultats apparaissent plutôt cohérents, il est difficile d'en dire plus que ce que nous avons fait dans cette représentation. Nous allons donc tâcher de les décrire en s'intéressant aux données affiliées à leur groupe sous leur forme initiale de trajectoire.

Les résultats sous leur forme de trajectoires ne sont pas évidentes non plus à interpréter mais nous pouvons tout de même mettre en évidence certains aspects du bon fonctionnement de notre algorithme. Pour des questions de lisibilité nous ne pouvons bien évidement pas illustrer de manière exhaustive les trajectoires en un seul graphique. Nous illustrerons plutôt quelques représentants de trajectoires types que notre algorithme a permis de mettre en évidence.

En explorant les groupes identifiés par Ballstering on peut donc retrouver plusieurs trajectoires types dont certaines illustrées dans la figure 6.8, où on affiche que quelques groupes par catégorie. On y trouve la catégorie des vols internes à la France en haute altitude (6.8a), les vols courts à basse altitude (6.8b), les trajectoires qui ne font



(A) Trajectoires internes à la France (B) petites

(C) survols (D) Trajectoires anormales

FIGURE 6.8 – Résultats de partitionnement de Ballstering sur les trajectoires d'avions. Quelques trajectoires types sont illustrées.

que survoler la france (6.8c) et des trajectoires anormales (6.8d). Deux ensembles de trajectoires types supplémentaires sont illustrés en annexe (figure A.3).

Les résultats semblent pertinents dans leur ensemble. Les groupes identifiés sont bien séparés, distincts et ne se ressemblent pas ou alors ont au moins une caractéristique flagrante qui les discriminent. De plus cette application nous a permis de mettre en évidence des groupes de trajectoires étranges comme de minuscules sections de trajectoires et des enregistrements de trajets tronqués alors qu'encore situés dans les secteurs couverts par les radars français.

Conclusion

Nous avons au cours de ce mémoire présenté en détail FDPC, une innovante et récente méthode de clustering. Une attention particulière a été apportée à ses défauts que sont sa grande complexité, sa dépendance à un paramètre d_c difficile à choisir, très impactant et non dynamique. De plus nous avons mis en lumière des défaillances de FDPC, impactant lourdement la qualité du clustering et intervenant dans des circonstances pourtant communes.

Afin de rendre sa première phase apte à analyser des volume de données conséquents, nous avons proposé ICMDW, une méthode itérative peu complexe par rapport au nombre d'observation N, permettant d'obtenir les densités sous une forme arborescente.

De la structure arborescente fournie par ICMDW nous avons dans un premier temps produit deux travaux. La construction de deux types de cartes de densités non-paramétriques de faible complexité. Ensuite nous avons construit une méthode accomplissant la deuxième phase de FDPC de manière homologue à cette dernière en exploitant habilement les sorties de ICMDW pour produire finalement une méthode équivalente à FDPC, plus efficace sur des données volumineuses et dont le choix crucial de d_c est évité. Cette dernière ne corrige cependant pas les défaillances identifiées.

Afin de passer outre tous les autres points faibles de FDPC que nous avons mis en évidence et dont l'algorithme précédent hérite, nous avons entièrement repensé la deuxième phase qui se juxtapose à ICMDW. Pour son établissement la définition de cœur de cluster a été remaniée pour y inclure la prise en compte de la densité avoisinante. La phase 2 est non plus effectuée par comparaison de distances Euclidiennes mais par propagation au sein du graphe formé par le réseau de boules fournit par ICMDW, permettant ainsi un clustering rapide, robuste et corrigeant tous les problèmes identifiés de FDPC.

La qualité de partitionnement de Ballstering a pu être mise en évidence à la fois sur des données simulées et réelles issu du trafic aérien français, en le comparant notamment aux deux célèbres méthodes que sont k-means et HDBSCAN. Par ce biais sa faible complexité a pu être mise en lumière empiriquement après l'avoir étudiée théoriquement.

Quant aux perspectives, nous pourrions améliorer les performances cet algorithme en optimisant et parallélisant le code et le comparer à nouveau à HDBSCAN et k-means à niveau égal.

Pour améliorer sa qualité en particulier on pourrait se pencher sur le problème de surestimation de la densité initiale en proposant un calcul exact ou une réponse à la surestimation plus fine.

Aussi notre algorithme est pour l'instant limité à des données appartenant à un espace euclidien car interviennent des calculs de volumes d'interaction entre hypersphères. Nous pourrions nous passer de cette condition en remplaçant le calcul du volume d'interaction par une fonction de la distance entre les points considérés, moins précise mais exportable à un cadre plus large.

Également les deux cartes de densités que nous obtenons ont du potentiel du fait de la rapidité et automaticité de la méthode mais ne sont pas parfaites et nécessitent un travail supplémentaire pour mieux comprendre leur comportements les étendre à un cadre plus formel.

En outre il serait intéressant de mener de plus amples études afin d'affiner la validation des paramètres et de la qualité du partitionnement et éventuellement la mise en place d'un paramètre R dépendant de la dimension d.

Enfin il paraît possible de dériver une RS fonctionnant en une seule passe et non deux comme c'est le cas actuellement. Aussi de toutes nouvelles manières d'exploiter l'arbre fournit par ICMDW pour extraire des partitions pourraient être envisagées.

Annexe A

Diverses représentations des trajectoires d'avions

Dans cette annexe nous apportons des illustrations supplémentaires vis-à-vis du traitement des données issues du trafic aérien. Cela comprend donc sa transformation initiale, notamment sur l'ACP qui lui est appliqué en fin de pré-traitement (section 2.2), ainsi qu'aux résultats de clustering présentés dans la section 6.3.

Nous commençons avec le tableau soutenant via valeurs numériques l'histogramme des éboulis des valeurs propres (figure 2.11), résultats de l'ACP appliquée aux données issues du trafic aérien français sur trois mois qui étaient dans leur représentation spline à 60 dimensions.

Ce tableau à 8 colonnes nous informe sur la répartition de la variabilité du jeu de données sur les 8 axes principaux. Si on se réfère à la troisième ligne "Cumulative proportion" on peut constater que les quatre composantes principales réunissent 91% de la variabilité initiale.

L'illustration suivante (figure A.1) permet de mieux apprécier la répartition des données dans l'espace engendré par les trois premières composantes principales. L'illustration qui suit (figure A.2) suit la même logique mais pour les données colorée par leur label

Enfin la figure A.3 contient deux illustrations supplémentaires de trajectoires types nous avons pu isoler des autres grâce au clustering. Nous comptons dans la première illustration des groupes de trajectoires transatlantiques, et des trajectoires en direction du au sud-Ouest dans la seconde

	Comp.1	Comp.2	Comp.3	Comp.4
Standard deviation	0.0004203871	0.0003076177	0.0002853066	0.0002180515
Proportion of Variance	0.4021063351	0.2153102410	0.1852106076	0.1081832890
Cumulative Proportion	0.4021063351	0.6174165761	0.8026271837	0.9108104727
	Comp.5	Comp.6	Comp.7	Comp.8
Standard deviation	0.0001342613	0.0001172288	5.333466e-05	4.314744e-05
Proportion of Variance	0.0410151296	0.0312687888	6.472338e-03	4.235962e-03
Cumulative Proportion	0.9518256023	0.9830943910	9.895667e-01	9.938027e-01

TABLE A.1 – Résultats de l'ACP appliquée aux données issues du trafic aérien français sur trois mois, dans sa représentation spline à 60 dimensions. Si on se réfère à la troisième ligne "Cumulative proportion" on peut constater que les quatre composantes principales réunissent déjà 91% de la variabilité initiale.

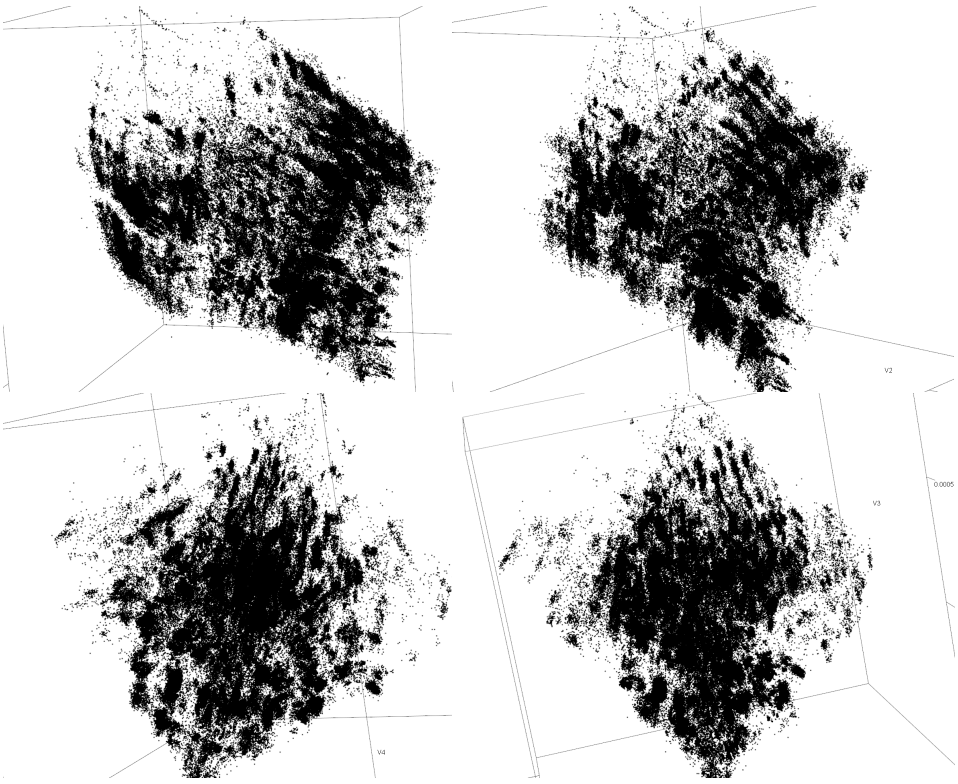


FIGURE A.1 – Représentation des trajectoires d'avions dans l'espace défini par les trois premières composantes principales, sous quatre angles différents. A lire dans le sens horaire.

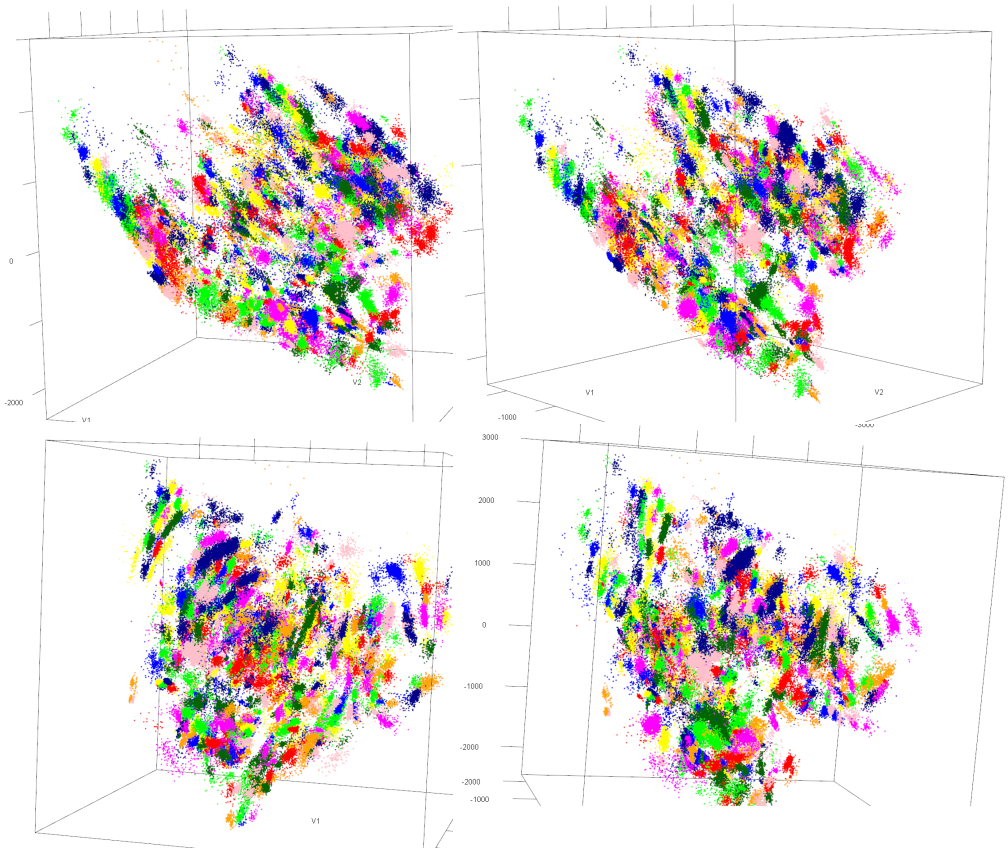


FIGURE A.2 – Représentation des trajectoires d'avions partitionnées dans l'espace défini par les trois premières composantes principales. A lire dans le sens horaire

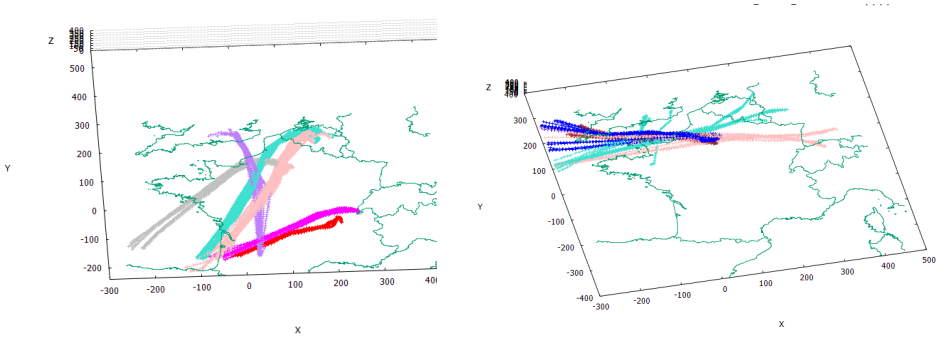


FIGURE A.3 – Deux illustrations supplémentaires de trajectoires types nous avons pu isoler des autres grâce au clustering. Nous comptons dans la première illustration des groupes de trajectoires en direction du sud-Ouest et des trajectoires transatlantiques dans la seconde

Annexe B

Compléments empiriques : choix du paramètre $N_{\min}$ et qualité du clustering

Cette annexe présente en premier deux tableaux qui donnent les valeurs numériques de l'évolution, en fonction de $N_{\min}$ (décroissant), du nombre d'échecs de clustering (sur 30 essais) pour plusieurs échantillons.

Le premier tableau (figure B) contient les résultats pour trois échantillons répartis selon la même loi uniforme mais dont le nombre d'observations est 10 000, 20 000 et 100 000. Le second (figure B) contient les résultats pour trois échantillons répartis selon une loi normale dont la variance $\sigma \in \{0.1, 0.5, 1\}$ varie selon l'échantillon.

Le dernier tableau contient tout simplement les valeurs numériques de la figure 6.4.

$N_{\min}$	Unif10K	Unif20K	Unif100K
30	0	0	0
29	0	0	0
28	0	0	0
27	0	1	0
26	1	4	0
25	1	7	1
24	1	12	10
23	5	15	30
22	8	5	19
21	17	15	30
19	12	1	20
18	6	1	12
17	3	1	9
16	2	15	20
15	9	28	30
14	19	30	30
13	28	30	30
12	30	30	30
11	30	30	30
10	29	30	30

$N_{\min}$	$\sigma = 0.1$	$\sigma = 0.5$	$\sigma = 1$
30	0	0	0
29	0	0	0
28	0	0	0
27	0	0	0
26	0	0	0
25	0	0	0
24	1	0	0
23	2	0	0
22	1	0	1
21	2	0	0
19	2	0	0
18	3	1	1
17	4	6	7
16	9	4	8
15	6	10	9
14	10	11	10
13	6	17	16
12	20	19	22
11	24	29	22
10	30	30	29

names	czekanowski dice	folkes mallows	jaccard
km	0.61099	0.6380	0.4413
bs	0.61736	0.6288	0.4474
hdb	0.18121	0.1878	0.1298
alea	0.09488	0.09488	0.04980

names	kulczynski	precision	recall
km	0.6663	0.8579	0.4747
bs	0.6406	0.7548	0.5264
hdb	0.1947	0.2458	0.1435
alea	0.09488	0.0948	0.09488

names	rogers tanimoto	russel rao	sokal sneath1
km	0.4223	0.3196	0.2842
bs	0.3891	0.3545	0.2889
hdb	0.1217	0.0967	0.0828
alea	-0.0984	0.06385	0.02553

names	sokal sneath2
km	0.7424
bs	0.7142
hdb	0.2195
alea	-0.5585

Annexe C

Algorithmique de DBSCAN

```
DBSCAN (SetOfPoints, Eps, MinPts)

// SetOfPoints is UNCLASSIFIED
  ClusterId := nextId(NOISE);
  FOR i FROM 1 TO SetOfPoints.size DO
    Point := SetOfPoints.get(i);
    IF Point.ClId = UNCLASSIFIED THEN
      IF ExpandCluster(SetOfPoints, Point,
            ClusterId, Eps, MinPts) THEN
        ClusterId := nextId(ClusterId)
      END IF
    END IF
  END FOR
END; // DBSCAN
```

FIGURE C.1 – Algorithmique de DBSCAN (Partie 1).

```
ExpandCluster(SetOfPoints, Point, ClId, Eps,
                MinPts) : Boolean;
  seeds:=SetOfPoints.regionQuery(Point,Eps);
  IF seeds.size<MinPts THEN // no core point
    SetOfPoint.changeClId(Point,NOISE);
    RETURN False;
  ELSE   // all points in seeds are density-
         // reachable from Point
    SetOfPoints.changeClIds(seeds,ClId);
    seeds.delete(Point);
    WHILE seeds <> Empty DO
      currentP := seeds.first();
      result := SetOfPoints.regionQuery(currentP,
                                        Eps);
      IF result.size >= MinPts THEN
        FOR i FROM 1 TO result.size DO
          resultP := result.get(i);
          IF resultP.ClId
              IN {UNCLASSIFIED, NOISE} THEN
            IF resultP.ClId = UNCLASSIFIED THEN
              seeds.append(resultP);
            END IF;
            SetOfPoints.changeClId(resultP,ClId);
          END IF; // UNCLASSIFIED or NOISE
        END FOR;
      END IF; // result.size >= MinPts
      seeds.delete(currentP);
    END WHILE; // seeds <> Empty
    RETURN True;
  END IF
END; // ExpandCluster
```

FIGURE C.2 – Algorithmique de DBSCAN (Partie 2).

Bibliographie

[1] Hervé Abdi et Lynne Williams. “Principal Component Analysis”. In : 2 (juil. 2010), p. 433 –459.

[2] Mihael Ankerst et al. “OPTICS : ordering points to identify the clustering structure”. In : *ACM Sigmod record*. T. 28. 2. ACM. 1999, p. 49–60.

[3] David Arthur et Sergei Vassilvitskii. “k-means++ : The advantages of careful seeding”. In : *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial et Applied Mathematics. 2007, p. 1027–1035.

[4] Jon Louis Bentley. “Multidimensional binary search trees used for associative searching”. In : *Communications of the ACM* 18.9 (1975), p. 509–517.

[5] John Adrian Bondy. *Graph Theory With Applications*. Oxford, UK, UK : Elsevier Science Ltd., 1976. isbn : 0444194517.

[6] Horst Bunke et al. “Graph clustering using the weighted minimum common supergraph”. In : *GbRPR* 2726 (2003), p. 235–246.

[7] Ricardo JGB Campello, Davoud Moulavi et Joerg Sander. “Density-based clustering based on hierarchical density estimates”. In : *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 2013, p. 160–172.

[8] Dongxia Chang et al. “A dynamic niching clustering algorithm based on individual-connectedness and its application to color image segmentation”. In : *Pattern Recognition* 60 (2016), p. 334 –347. issn : 0031-3203. doi : http://dx.doi.org/10.1016/j.patcog.2016.05.008.

[9] Yizong Cheng. “Mean shift, mode seeking, and clustering”. In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17.8 (1995), p. 790–799. issn : 0162-8828. doi : 10.1109/34.400568.

[10] Charles K Chui. *Multivariate splines*. SIAM, 1988.

[11] Arthur P Dempster, Nan M Laird et Donald B Rubin. “Maximum likelihood from incomplete data via the EM algorithm”. In : *Journal of the royal statistical society. Series B (methodological)* (1977), p. 1–38.

[12] Bernard Desgraupes. “clusterCrit : Clustering indices”. In : *R package version* 1.3 (2013), p. 4–5.

[13] H. Du, S. zhao et D. zhang. “Robust Local Outlier Detection”. In : *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*. 2015, p. 116–123. doi : 10.1109/ICDMW.2015.114.

[14] J. C. Dunn. “A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters”. In : *Journal of Cybernetics* 3.3 (1973), p. 32–57. doi : 10.1080/01969727308546046. eprint : http://dx.doi.org/10.1080/01969727308546046. url : http://dx.doi.org/10.1080/01969727308546046.

[15] Martin Ester et al. “A density-based algorithm for discovering clusters in large spatial databases with noise.” In : *Kdd*. T. 96. 34. 1996, p. 226–231.

[16] Thomas S Ferguson. “A Bayesian analysis of some nonparametric problems”. In : *The annals of statistics* (1973), p. 209–230.

[17] P. Fränti. *Clustering datasets*. 2015. url : http://cs.uef.fi/sipu/datasets/.

[18] Luiz Flavio Autran Monteiro Gomes et al. “3rd International Conference on Information Technology and Quantitative Management, ITQM 2015 Image Segmentation via Improving Clustering Algorithms with Density and Distance”. In : *Procedia Computer Science* 55 (2015), p. 1015 –1022. issn : 1877-0509. doi : http://dx.doi.org/10.1016/j.procs.2015.07.096.

[19] Robert Gray. “Vector quantization”. In : *IEEE Assp Magazine* 1.2 (1984), p. 4–29.

[20] Marti A. Hearst et al. “Support vector machines”. In : *IEEE Intelligent Systems and their applications* 13.4 (1998), p. 18–28.

[21] Sebastiano Impedovo. “More than twenty years of advancements on Frontiers in handwriting recognition”. In : *Pattern Recognition* 47.3 (2014), p. 916–928.

[22] Sen Jia, Guihua Tang et Jie Hu. “Band Selection of Hyperspectral Imagery Using a Weighted Fast Density Peak-Based Clustering Approach”. In : *Intelligence Science and Big Data Engineering. Image and Video Data Engineering : 5th International Conference, IScIDE 2015, Suzhou, China, June 14-16, 2015, Revised Selected Papers, Part I*. Sous la dir. de Xiaofei He et al. Cham : Springer International Publishing, 2015, p. 50–59. isbn : 978-3-319-23989-7. doi : 10.1007/978-3-319-23989-7_6.

[23] Xin Jin et Jiawei Han. “K-Medoids Clustering”. In : *Encyclopedia of Machine Learning*. Sous la dir. de Claude Sammut et Geoffrey I. Webb. Boston, MA : Springer US, 2010, p. 564–565. isbn : 978-0-387-30164-8. doi : 10.1007/978-0-387-30164-8_426. url : https://doi.org/10.1007/978-0-387-30164-8_426.

[24] Stephen C Johnson. “Hierarchical clustering schemes”. In : *Psychometrika* 32.3 (1967), p. 241–254.

[25] Samuel Kaski, Janne Sinkkonen et Arto Klami. “Discriminative clustering”. In : *Neurocomputing* 69.1 (2005), p. 18–41.

[26] Leonard Kaufman et Peter Rousseeuw. *Clustering by means of medoids*. North-Holland, 1987.

[27] Teuvo Kohonen. “Self-organized formation of topologically correct feature maps”. In : *Biological cybernetics* 43.1 (1982), p. 59–69.

[28] Bernhard Korte et Jens Vygen. *Combinatorial Optimization : Theory and Algorithms*. 4th. Springer Publishing Company, Incorporated, 2007. isbn : 3540718435, 9783540718437.

[29] “La régression linéaire simple”. In : *Régression : Théorie et applications*. Paris : Springer Paris, 2007, p. 1–32. isbn : 978-2-287-39693-9. doi : 10.1007/978-2-287-39693-9_1. url : https://doi.org/10.1007/978-2-287-39693-9_1.

[30] S. Li et al. “An efficient clustering method for medical data applications”. In : *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2015 IEEE International Conference on*. 2015, p. 133–138. DOI : 10.1109/CYBER.2015.7287923.

[31] Shengqiao Li. “Concise formulas for the area and volume of a hyperspherical cap”. In : *Asian Journal of Mathematics and Statistics* 4.1 (2011), p. 66–70.

[32] Zhou Liang et Pei Chen. “Delta-density based clustering with a divide-and-conquer strategy : 3DC clustering”. In : *Pattern Recognition Letters* 73 (2016), p. 52 –59. ISSN : 0167-8655. DOI : http://dx.doi.org/10.1016/j.patrec.2016.01.009.

[33] D. Liu, S. F. Cheng et Y. Yang. “Density Peaks Clustering Approach for Discovering Demand Hot Spots in City-scale Taxi Fleet Dataset”. In : *2015 IEEE 18th International Conference on Intelligent Transportation Systems*. 2015, p. 1831–1836. DOI : 10.1109/ITSC.2015.297.

[34] P. Liu et al. “A Text Clustering Algorithm Based on Find of Density Peaks”. In : *2015 7th International Conference on Information Technology in Medicine and Education (ITME)*. 2015, p. 348–352. DOI : 10.1109/ITME.2015.103.

[35] C. Ma, T. Ma et H. Shan. “A new important-place identification method”. In : *Computer and Communications (ICCC), 2015 IEEE International Conference on*. 2015, p. 151–155. DOI : 10.1109/CompComm.2015.7387558.

[36] J. MacQueen. “Some methods for classification and analysis of multivariate observations”. In : *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1 : Statistics*. Berkeley, Calif. : University of California Press, 1967, p. 281–297.

[37] Geoffrey McLachlan et Thriyambakam Krishnan. *The EM algorithm and extensions*. T. 382. John Wiley & Sons, 2007.

[38] Rashid Mehmood et al. “Clustering by fast search and find of density peaks via heat diffusion”. In : *Neurocomputing* (2016), p. –. ISSN : 0925-2312. DOI : http://dx.doi.org/10.1016/j.neucom.2016.01.102.

[39] Rashid MEHMOOD et al. “Fuzzy Clustering by Fast Search and Find of Density Peaks”. In : *2015 International Conference on Identification, Information, and Knowledge in the Internet of Things (IIKI)*. IEEE. 2015, p. 258–261.

[40] Li MIAO. “Comparative Analysis of Two Clustering Algorithms : K-means and FSDP (Fast Search and Find of Density Peaks)”. Master’s Project, Paper 427. Mém.de mast. San Jose State University, 2015.

[41] A. PAPOULIS. *Signal analysis*. McGraw-Hill electrical and electronic engineering series. McGraw-Hill, 1977. ISBN : 9780070484603. URL : https://books.google.fr/books?id=Re5SAAAAMAAJ.

[42] F. PEDREGOSA et al. “Scikit-learn : Machine Learning in Python”. In : *Journal of Machine Learning Research* 12 (2011), p. 2825–2830.

[43] Dzung L PHAM, Chenyang XU et Jerry L PRINCE. “Current methods in medical image segmentation”. In : *Annual review of biomedical engineering* 2.1 (2000), p. 315–337.

[44] Yan QIN, Chunhui ZHAO et Furong GAO. “An iterative two-step sequential phase partition (ITSPP) method for batch process modeling and online monitoring”. In : *AIChE Journal* 62.7 (2016), p. 2358–2373. ISSN : 1547-5905. DOI : 10.1002/aic.15205.

[45] J. O. RAMSAY. “Functional Data Analysis”. In : *Encyclopedia of Statistical Sciences*. John Wiley & Sons, Inc., 2004. ISBN : 9780471667193. DOI : 10.1002/0471667196.ess3138. URL : http://dx.doi.org/10.1002/0471667196.ess3138.

[46] Carl Edward RASMUSSEN. “The infinite Gaussian mixture model”. In : *Advances in neural information processing systems*. 2000, p. 554–560.

[47] Antonio ROBLES-KELLY et Edwin R HANCOCK. “Graph edit distance from spectral seriation”. In : *IEEE transactions on pattern analysis and machine intelligence* 27.3 (2005), p. 365–378.

[48] Alex RODRIGUEZ et Alessandro LAIO. “Clustering by fast search and find of density peaks”. In : *Science* 344.6191 (2014), p. 1492–1496. ISSN : 0036-8075.

DOI : 10.1126/science.1242072. eprint : http://science.sciencemag.org/content/344/6191/1492.full.pdf.

[49] Haşim SAK et al. “Fast and accurate recurrent neural network acoustic models for speech recognition”. In : *arXiv preprint arXiv :1507.06947* (2015).

[50] David W SCOTT. *Multivariate density estimation : theory, practice, and visualization*. John Wiley & Sons, 2015.

[51] Maxim SHEVTSOV, Alexei SOUPIKOV et Alexander KAPUSTIN. “Highly Parallel Fast KD-tree Construction for Interactive Ray Tracing of Dynamic Scenes”. In : *Computer Graphics Forum*. T. 26. 3. Wiley Online Library. 2007, p. 395–404.

[52] Yong SHI et al. “A novel clustering-based image segmentation via density peaks algorithm with mid-level feature”. In : *Neural Computing and Applications* (2016), p. 1–11. ISSN : 1433-3058. DOI : 10.1007/s00521-016-2300-1.

[53] Bernard W SILVERMAN. *Density estimation for statistics and data analysis*. T. 26. CRC press, 1986.

[54] Mingjun SONG et Daniel CIVCO. “Road extraction using SVM and image segmentation”. In : *Photogrammetric Engineering & Remote Sensing* 70.12 (2004), p. 1365–1371.

[55] Robert Endre TARJAN. *Data structures and network algorithms*. SIAM, 1983.

[56] Naftali TISHBY, Fernando C PEREIRA et William BIALEK. “The information bottleneck method”. In : *arXiv preprint physics/0004057* (2000).

[57] G. WANG et Q. SONG. “Automatic Clustering via Outward Statistical Testing on Density Metrics”. In : *IEEE Transactions on Knowledge and Data Engineering* 28.8 (2016), p. 1971–1985. ISSN : 1041-4347. DOI : 10.1109/TKDE.2016.2535209.

[58] Shuliang WANG et al. “Clustering by Fast Search and Find of Density Peaks with Data Field”. In : *Chinese Journal of Electronics* 25.3 (2016), p. 397–402.

[59] Xiao-Feng WANG et Yifan XU. “Fast clustering using adaptive density peak detection”. In : *Statistical Methods in Medical Research* (2015). DOI : 10.1177/0962280215609948.

[60] Y. Wang et al. “Clustering of Electricity Consumption Behavior Dynamics toward Big Data Applications”. In : *IEEE Transactions on Smart Grid* PP.99 (2016), p. 1–1. ISSN : 1949-3053. DOI : 10.1109/TSG.2016.2548565.

[61] John H Wolfe. “Pattern clustering by multivariate mixture analysis”. In : *Multivariate Behavioral Research* 5.3 (1970), p. 329–350.

[62] Juanying Xie et al. “Robust clustering by detecting density peaks and assigning points based on fuzzy weighted K-nearest neighbors”. In : *Information Sciences* 354 (2016), p. 19 –40. ISSN : 0020-0255. DOI : http://dx.doi.org/10.1016/j.ins.2016.03.011.

[63] Linli Xu et al. “Maximum margin clustering”. In : *Advances in neural information processing systems*. 2005, p. 1537–1544.

[64] J. Yu et al. “A density peak clustering approach to unsupervised acoustic subword units discovery”. In : *2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*. 2015, p. 178–183. DOI : 10.1109/APSIPA.2015.7415498.

[65] Charles T Zahn. “Graph-theoretical methods for detecting and describing gestalt clusters”. In : *IEEE Transactions on computers* 100.1 (1971), p. 68–86.

[66] R. Zhang et al. “An Improved Fast Search Clustering Algorithm Based on Kernel Density”. In : *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*. 2015, p. 689–693. DOI : 10.1109/SmartCity.2015.149.

[67] Wenyi Zhao et al. “Face recognition : A literature survey”. In : *ACM computing surveys (CSUR)* 35.4 (2003), p. 399–458.